




ERJU SYSTEM PILLAR

System Requirements Specification_TCCS - Part 3 Service Function Configuration and Configurable BuildingBlock (SERA Version)



System Requirements Specification_TCCS - Part 3 Service Function Configuration and Configurable BuildingBlock (SERA Version)

Author(s)	Karl-Albrecht Klinge , Bolz, Gert (SMO RI R&D IXL IL) , RICHTER Robert , Ghielmetti Cirillo (I-NAT-GST-CCS)
Abstract	This document is the output of system requirement phase activities (phase 4) as specified in SPPRAMSS-349 - EN 50126-1:2017 for the Service Function Configuration (SFC). The objective of this document is to specify a comprehensive and identified set of requirements for the Service Function Configuration (SFC).
Config Item	System Requirements Specification
Document ID	TCCS Service Function Configuration _SFC_ L5/TCCS SRS ServiceFunctionConfiguration and Configurable BuildingBlock#723853  System Requirements Specification_TCCS - Part 3 Service Function Configuration and Configurable BuildingBlock (SERA Version)
Classification	Public
Status	In Review by System Pillar
Version	1.0
Revision	723853
Last Change Date	02.10.2025
Copyright	Brussels: Europe's Rail Joint Undertaking, 2025

© Europe's Rail Joint Undertaking, 2025

This document is drafted by and belongs to EU Rail.

EU Rail encourages the distribution and re-use of this document, the technical specifications and the information it contains. EU Rail holds several intellectual property rights, such as copyright and trade mark rights, which need to be considered when this document is used.

EU Rail authorises you to re-publish, re-use, copy and store this document without changing it, provided that you indicate its source and include the following: EU Rail trade mark, title of the document, year of publication, version of document.

EU Rail makes no representation or warranty as to the accuracy or completeness of the information contained within these documents. EU Rail shall have no liability to any party as a result of the use of the information contained herein. EU Rail will have no liability whatsoever for any indirect or consequential loss or damage, and any such liability is expressly excluded.

You may study, research, implement, adapt, improve and otherwise use the information, the content and the models in the this document for your own purposes. If you decide to publish or disclose any adapted, modified or improved version of this document, any amended implementation or derivative work, then you must indicate that you have modified this document, with a reference to the document name and the terms of use of this document. You may not use EU Rail's trade marks or name in any way that may state or suggest, directly or indirectly, that EU Rail is the author of your adaptations.

EU Rail cannot be held responsible for your product, even if you have used this document and its content. It is your responsibility to verify the quality, completeness and the accuracy of the information you use, for your own purposes.


This work is currently a work in progress. The content presented is subject to change as it undergoes further review, refinement, and development. Please do not consider this version as final or authoritative.

INFO: History table is not displayed, because this document is in status **doc_contentApproval**.


RULE: History table is not displayed, in statuses: { draft doc_open doc_inprogress doc_contentApproval doc_contentDecision }

CONTACT: For more information contact Administrator

Review description

Attachments	REMINDER_ [ERJU SP] Request to review SC2.4 List of deliverables - Task 2_ Transversal Systems .pdf , Review and Approval Jens Kilian.pdf , Review and Approval Virgil Lostun.pdf
Comments	#1 Approval comment by Golebniak, Udo (SMO RI ML ADC I&C) on 2025-10-01 12:27 The current TCCS Design is focused on IP Needs. The usage in Traffic CS Environment is still to be discussed and must follow top-down design. Traffic CS has currently not reached the necessary level in the design. Target date for the Traffic CS Specification work is 2027.
Approvals	LOSTUN Virgil : Waiting , SANGO Marc (SNCF / DIR TECHNOLOGIES INNOVATION ET PROJETS GROUPE / IR DIR RECHERCHE - PSF) : Waiting , DE NICOLA, Giuseppe : Waiting , KEFALAS, Georgios : Waiting , Julien Bois : Waiting , Oliver Knapp : Waiting , Wischy, Markus Alexander (SMO RI R&D F IL) : Waiting , HENON Frédéric : Waiting , TEKE, Emre : Waiting , Renato Rodrigues : Waiting , IOVINO, Salvatore : Waiting , Davinder Bhatia : Waiting , BITSCH Friedemann : Waiting , Roman R Treydel : Waiting , Golebniak, Udo (SMO RI ML ADC I&C) : Waiting , Mirko Blazic : Waiting , Benameur, Malik (SMO NEE RC-CH RI PLM SYS) : Waiting , MOTTOLA, Giuseppe Diodato : Waiting , Jack Schneider : Waiting , Zeeshan Z Ansar : Waiting , Patrick Konix : Waiting , NANNI Marco : Waiting , DE MARCO TELESE Giancarlo : Waiting , Tione, Roberto : Waiting , Andreeva-Moschen Emilia (HOLDING) : Waiting , Kilian Jens : Waiting
Type of Approval	 Document Review

Approval description

Attachments	REMINDER_ [ERJU SP] Request to review SC2.4 List of deliverables - Task 2_ Transversal Systems .pdf , Review and Approval Jens Kilian.pdf , Review and Approval Virgil Lostun.pdf
Approvals	LOSTUN Virgil : Waiting
Type of Approval	 Document Approval

1 Preamble	4
1.1 Scope and intended audience	4
1.2 Purpose	6
1.3 Glossary	7
1.3.1 General abbreviations	7
1.3.2 Document-specific abbreviations	7
1.3.3 Term definition	7
2 Assumptions and dependencies	10
3 Constraints	10
4 System overview	10
4.1 System context	10
4.2 System interfaces	19
4.2.1 REPO	19
4.2.2 SMI (v3)	19

4.3 System modes and states	19
5 System requirements	20
5.1 General Requirements	20
5.2 Configuration Repository interaction and validation requirements	21
5.3 Dependency Tree validation requirements	22
5.4 Distribution requirements	23
5.4.1 Stages and Actors	23
5.4.2 Sequences	24
5.4.2.1 Configuration distribution process stages	24
5.4.2.1.1 Preconditions	24
5.4.2.1.2 Start Distribution Job	25
5.4.2.1.3 Preloading	25
5.4.2.1.4 Safe Top Level BBC Activation	26
5.4.2.1.5 Deactivation	27
5.4.2.1.6 Activation	28
5.4.2.1.7 Confirmation	29
5.4.2.2 Example: Top-Level BuildingBlockConfiguration version switch	31
5.4.3 Overall orchestration	35
5.4.4 Safety Attestation Requirements	37
5.4.5 Safe Configuration Authority Functions	39
5.5 OPC UA Security Requirements	41
5.6 Non-functional requirements	41
5.6.1 SFC	41
5.6.2 BuildingBlock	41
5.7 Functional requirements	41
5.7.1 SFC	41
5.7.2 BuildingBlock	42
6 Risk analysis	42
7 Application conditions	45
7.1.1 Functional overview	46
7.1.2	46

Figure 1. Logical architecture Service Function Configuration

1 Preamble

1.1 Scope and intended audience

ToDo: Focus on SFC

The new generation of railway equipment, like vehicles and TPS or ATO-TS, consists of digital components with the capability to be configured via software / data update mechanisms. This flexibility applies to the safety and the non-safety related software / data parts of these components.

This configuration management process automates the following configuration tasks, provided that the BuildingBlock is initialised to allow the communication as described in this document:

- base configuration of complex systems with explicit dependency management
- update of configurations
- initial setup of replaced BuildingBlocks

A typical configuration management process relies on different stakeholders and responsibilities whereby suppliers, integrators and maintenance operators (RU/IM) are the main actors.

To ensure that these components work together safely and effectively in systems of arbitrary size and complexity, there is the need to define and standardise the configuration management process.

This process must define explicit configuration management coordination principles. This requires (at least) standardised meta-information for automated distribution and orchestration of artifacts, verification, and a validation process. This configuration management process must ensure the integrity and authenticity of all configuration items (artifacts) and their dependencies. It is important that the process allows each supplier and integrator to take over responsibility in line with their competence.

DRAFT




For example:

We have an area of control with three TPS, EAL and ATO-TS (which may collectively contain 1000 field elements from different suppliers) which requires a software configuration update.

Without explicit, safe management of dependencies and automated distribution, this is hard to manage, error prone and may result in a misconfiguration. The same applies to a vehicle with all its components (e.g., from the safe computer to door systems) that could be in different hardware revisions in different homologated versions within one train series.

[SPT2TS-129847]

1.2 Purpose

This document is the output of system requirement phase activities (phase 4) as specified in  SPPRAMSS-349 - [EN 50126-1:2017] for the Service Function Configuration (SFC). The objective of this document is to specify a comprehensive and identified set of requirements for the Service Function Configuration (SFC).

1.3 Glossary

1.3.1 General abbreviations

Abbreviation	Title
BB	Building Block
BBC	Building Block Configuration
HW	Hardware
JWT	JSON Web Token
SFC	Service Function Configuration
SW	Software

6 items found 

1.3.2 Document-specific abbreviations

Abbreviation	Title
BB	Building Block
BBC	Building Block Configuration
HW	Hardware
JWT	JSON Web Token
SFC	Service Function Configuration
SW	Software

6 items found 

1.3.3 Term definition


Title	Description
Building Block	<p>A BuildingBlock is a logical unit of the system that is bound to a sourceable physical equipment by means of a Basic Data Identifier having:</p> <ul style="list-style-type: none"> • standardised functionality or aggregates standard functionality it depends on • may have standardised PRAMS requirements (including Tolerable Functional Failure Rate [TFFR]) • may have Safety Integrity Levels [SIL] for functions within the system border and Safety Related Application Conditions [SRAC]) • standardised cyber security requirements (including Security Level [SL] based on the security requirements, and Security Related Application Conditions [SRAC]) • may have (on lower levels) standardised interfaces (on all OSI Layers) towards other Building Blocks and/or external systems.

Title	Description
	<p>One equipment can host several BuildingBlocks (e.g in the case of a MultiObjectController) and may be separately sourceable from different suppliers and capable of being integrated by a third party (integrator). A BuildingBlock is configured by one or more BuildingBlockConfigurations.</p> <p>A BuildingBlock must have an unique identifier called bbld (that could be a technical system or subsystem identifier). The bblds are assigned by the integrator and are transferred to another physical unit in case of replacement. Each bbld must be unique.</p>
Building Block Configuration	<p>A BuildingBlockConfiguration (BBC) is a node on a layer within the configuration dependency tree. It must be uniquely identifiable within the system as bbcld. On the lower levels of the dependency trees the bbcls are assigned by the supplier - for example by imprinting the structure of configurable items into a sourceable physical unit. A BBC may contain a configurationFile artifact and dependencies to other BBCs. One Building BlockConfiguration (BBC) has exactly one configuration.json file (and an additional configurationSafe.json if it is a safe BBC). BBCs that itself have no further dependencies in their configuration.json file are the Lowest Updatable Units (LUU - can be updated on its own). BBCs that are updatable must provide a corresponding configurationFile (payload). BBCs that are updatable need an endpoint described in the "configuration.json" file. That BBC endpoint can be accessed using a protocol capable of file transfer (e.g. opc ua).</p>
CCS Deployment	<p>CCS Deployment refers to one physical deployment of a CCS System, that is uniquely identifiable with bbld, bbcld and bbcVersion. A CCS Deployment consists of the CCS hardware running the BBCs that are defined in a Top-Level BuildingBlockConfiguration (BBC) and its dependencies.</p>
Configuration Management	<p>Configuration Management refers to the management of all Configuration Items of a physical instance of a System (e.g. trains or trackside CCS). From a process point of view, it covers activities along the complete chain, from the Building Block Supplier(s) to the Integrators, the Operator, and the actual Deployment being operational.</p>
Configuration Management Concept	<p>The Configuration Management Concept is a solution oriented description of the Configuration Management Process: it defines all the technical solutions needed for the concrete implementation of the Configuration Management Process.</p>
Configuration Management Processes	<p>The Configuration Management Process captures all concrete activities along the complete configuration management chain: from the Building Block realisation (by the Building Block Supplier(s)), to the integration into a specific System Instance (by the Integrator(s)), to the distribution to the field (by the Operator), and the actual activation (going into operation) in the field (it includes also on a train). The Configuration Management Process is a solution agnostic term that collects all activities needed for the Configuration Management.</p>
DistributionJob "distribution-job.json"	<p>A DistributionJob is defines the time and conditions when one or more CCS deployments with the same homologation receive their software configuration update (preload and activation). Examples for conditions when the distribution-job is started are approvals from a train-driver, or someone from operation or a GOA4 system that need to approve the configuration update within the preloading and activation time windows.</p>


Title	Description
n" document	<p>The distribution-job.json document references the Top Level BuildingBlockConfiguration (BBC), that has recursive dependency tree.</p> <p>The distribution-job is defined within a distribution-job.json document that is validated by the distribution-job.schema.json.</p>
Hardware	<p>Hardware refers to the physical components of a computer system or electronic device. It encompasses all the tangible parts that can be seen and touched, such as the central processing unit (CPU), memory modules, storage devices (hard drive, solid-state drive), possible input/output devices (e.g. keyboard, mouse, monitor, etc), motherboard, graphics card, sound card, and various other components that make up a computation unit or electronic device. Hardware is responsible for executing and processing instructions and data, providing the necessary functionality for software to run and perform tasks.</p>
High level BuildingBlocks	<p>Due to the dependency concept the BuildingBlocks can form units of unlimited size. The recursive dependency tree can have any depth.</p> <p>Every level of the dependency tree links to the next level - that allows to assign clear responsibilities for each of the levels.</p> <p>As an example a BuildingBlock "area" might refer to a number of "interlocking interiors" and "filedelements" in its next downstream dependency level.</p> <p>The Top-Level BuildingBlockConfiguration is the root where all dependencies start from.</p>
JSON Web Token	<p>JSON Web Token is an Internet standard for creating data with signature and/or encryption whose payload holds JSON that asserts some number of claims.</p>
Service Function Configuration	<p>The Service Function Configuration (SFC) is the implementation of the Configuration Management System.</p> <p>The SFC is a central location technical system that is responsible for managing the BuildingBlock Configurations.</p> <p>Each BuildingBlock deployment is managed by exactly one SFC.</p>
Software	<p>Software is software that encompasses both : the operating system and the computer application that runs the computer. They are both essential for the operation of a device.</p>
Static (or semi-static) data	<p>Static (or semi-static) data refers to information or data that remains unchanged or constant over time. It is data that does not require frequent updates or modifications. Static data typically includes reference data, constants, configuration settings, or any other data that remains consistent throughout the operation of a system or application. This type of data is often used as a foundation or reference point for various processes or calculations within a system. Static data is generally stored in a read-only format and is not subject to frequent modifications or user interactions. This is often data that requires a homologation process before it can be applied on a system going in operation. However, it can also include data that does not require a homologation like IP-address, security patches, etc.</p> <p>Example of statical data: software, firmware, parametrisation file, data related to the topology stored in an interlocking, braking curves stored in the ETCS on-board, etc.</p>

13 items found 

2 Assumptions and dependencies

Note to author: List any assumptions and dependencies applicable to the system requirements that should be taken into account in the allocation and derivation of lower-level system requirements. If applicable assumptions are covered by other documents such as  SPPR-7906 - System Definition, these can be referenced here instead.

3 Constraints

Note to author: List any constraints applicable to the system requirements. If applicable constraints are covered by other documents such as  SPPR-7906 - System Definition, these can be referenced here instead.

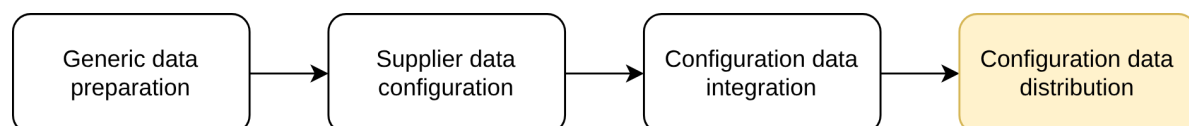
4 System overview

For the initial and subsequent configuration of interlockings, trackside elements and further field elements, RBCs, and onboard, a configuration process is proposed. System Pillar is in the process of standardising the secure distribution of safe and non-safe configuration data, and software. The dependencies of the configuration are explicitly defined in the form of a “dependency tree”. That is IT best practice for complex projects, e.g. using Maven or Gradle . A dependency tree represents the hierarchical structure of all the dependencies a system requires. It shows how each configuration item in a specific version depends on other configuration items in a specific version. The configuration items are stored in a Configuration Repository that is capable of resolving and downloading transitive dependencies in other Configuration Repositories. Managing dependencies can become complex, especially in large systems. The dependency tree helps to visualise these relationships and to identify configuration items that have a version change, when a new top-level version is chosen for distribution including potential conflicts. The data-driven and non-safe configuration interface is called SMI – Standardised Maintenance Interface. SMIs enable the distribution of safe and non-safe configuration data (including software / firmware) according to the dependency tree. The dependency tree is a result of the data preparation process and is the only way to configure the operational systems consistently. [SPT2TS-130478]

4.1 System context

The Service Function Configuration is part of a four phases-process starting from the data preparation. Each phases can have constraints imposed by the next ones. The four phases are the following :

1. **Generic data preparation** : this concerns the generic BuildingBlockConfiguration, with the associated default parametrisation. This is an input from the supplier.
2. **Supplier data configuration** : this phase is performed by the **supplier**. The configuration files are generated based on the generic model provided in the data preparation.
3. **Configuration data integration** : this phase is performed by the **integrator**. All the dependencies between the different compatible configurations are defined in the dependency tree.
4. **Configuration data distribution** : this phase is covered by the **Service Function Configuration**.



Notice : this SRS focuses on the phase 4 : configuration data distribution.
[SPT2TS-130816]

Environment and systems for the configuration management process

The following figure shows the environment and the different systems involved in the configuration management process including their actors (stakeholders):

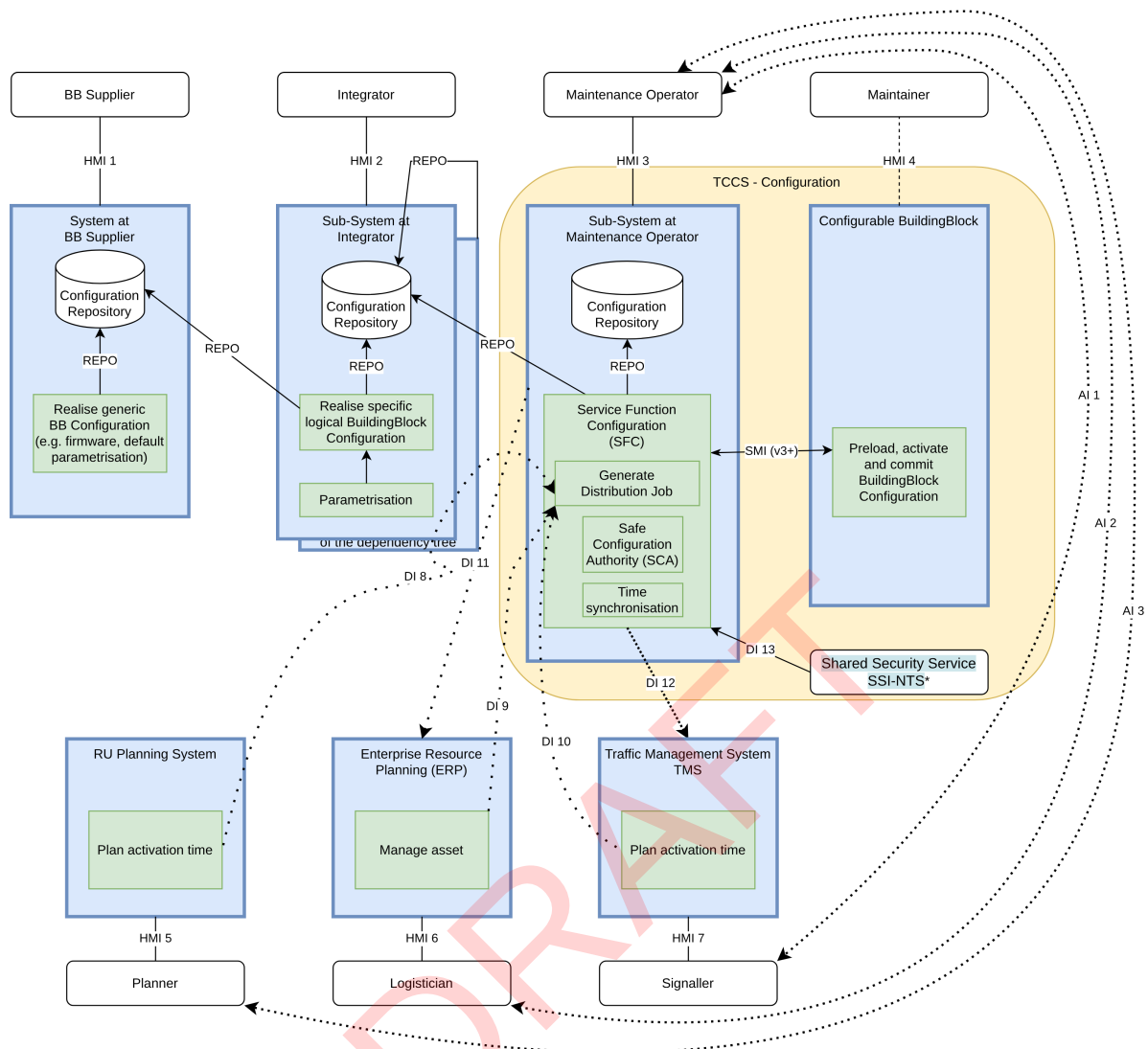


Figure 1 Logical architecture Service Function Configuration

System	Description	Function
System at Building Block (BB) Supplier	<p>From configuration management perspective, this is the system used by the BB Supplier as interface to the Integrator. It is used by the BB Supplier to publish generic BuildingBlockConfigurations.</p> <p>Some of these generic BuildingBlockConfigurations will remain unchanged until installation (typically executable like software, firmware, etc.). The supplier will also create default parametrisation files whose content is adjusted to the specific logical application by an Integrator. The supplier will provide the limits for all the parameter values.</p> <p>This includes a repository to be used to distribute both safety relevant and non-safety relevant BuildingBlockConfigurations (interface REPO).</p>	The system publishes (binary) configurationFile items and additional files identifying the BuildingBlockConfiguration (BBC) to be accessed in a repository (see below).

System	Description	Function
	<p>REPO is the access to a repository that stores BBC artifacts. That is applicable for BBC artifacts that a supplier provides to an integrator, integrators to integrators or integrators to maintenance operators.</p>	
(Sub-)system at Integrator	<p>From configuration management perspective, this is the data preparation subsystem used by the Integrators on different levels of the system.</p> <p>On the lower levels it depends on the generic BuildingBlockConfigurations published in the repositories of the BuildingBlock Suppliers and defines specific BuildingBlockConfigurations that contain the parametrisation for an instance of a logical railway (sub-)system.</p> <p>On higher levels the Integrators create logical BuildingBlockConfigurations that describe the composition of BuildingBlockConfigurations one level down. The number of levels is not restricted.</p> <p>- As an example for trackside: if an area of control is described that contains a number of TPS, EAL and ATO-TS to be configured as a combined system: BBC area (==Top Level BBC) (depends on a number of) BBC interlocking (depends on a number of) field element logical parametrisation (depends on) field element firmware</p> <p>- As an example for on-board: if a vehicle is described that contains an ETCS on-board, a cabin radio, and an ATO-OB to be configured as a combined system: BBC vehicle (==Top Level BBC) (depends on a number of) ETCS on-board (depends on a number of) DMI (depends on) DMI firmware (depends on) DMI parametrisation</p> <p>The Integrators on different levels of the system need not necessarily be in one company. The interface REPO between the different levels is a BuildingBlockConfiguration repository.</p> <p>The BuildingBlockConfigurations form a dependency tree (see below).</p> <p>It is useful to differentiate between safe and non-safe BuildingBlockConfigurations to reduce the effort for non-safe configuration updates. For this reason the integrity of safe BBCs is secured by a sidecar file that contains hashes of the dependencies one level down recursively, this ensures that the dependency between different safe BBCs is preserved.</p> <p>For non-safe BBCs the dependency is not documented allowing for a leaner process. Primarily, the integrator does not have to create all the dependency hashes.</p> <p><i>We do not define the binary configurationFile that contains</i></p>	<p>The following functional range is provided:</p> <ul style="list-style-type: none"> • Management of (generic) BuildingBlockConfigurations published by BuildingBlock Suppliers. • Management of BuildingBlockConfigurations (BBC) that are dependent on other BBCs, e.g. a parametrisation BBC dependent on the generic BBCs of the supplier including the logical parametrisation files with parametrisation values adjusted to the specific application. • Produce the configuration.json and configurationSafe.json (see below) to describe the dependencies on exact versions of BBCs one level down. • Publish the Top-Level BuildingBlockConfigurations including all BuildingBlockConfigurations in a repository accessible by the Maintenance Operator.

System	Description	Function
	<i>e.g. firmware or parametrisation. If data models are needed e.g. for the parametrisation of the interlocking topology we assume that the SD1 data models will be used.</i>	
Sub-system at Maintenance Operator	<p>From configuration management perspective, this is the system used by the Maintenance Operator to select the Top-Level BuildingBlockConfiguration provided in the Integrator(s) repositories REPO and define which target shall when preload and activate a BuildingBlockConfiguration dependency tree based on the distribution-job provided to repository using REPO as well.</p> <p>The ServiceFunctionConfiguration (SFC) stores which BuildingBlock is using which exact BuildingBlockConfiguration (version). It is also capable of analysing and comparing different versions of the BuildingBlockConfiguration dependency trees to know which BuildingBlockConfiguration needs to be updated on a Top-Level BuildingBlockConfiguration version switch. It manages the activation and Confirmation of all BuildingBlockConfigurations that need a version change. For the confirmation of safe BuildingBlockConfigurations it contains an add-on function called SafeConfigurationAuthority (SCA).</p> <p>For asset management in an Enterprise Resource Management (ERP) system it could be helpful to offer an interface to read the current BuildingBlockConfiguration versions from this system.</p>	<p>The following functional range is provided:</p> <ul style="list-style-type: none"> • Management of Top-Level BuildingBlockConfigurations published by Integrators. • Producing the distribution-job defining the target and when to preload and activate a safe/non-safe BuildingBlockConfiguration. • Process the distribution-jobs to update the BuildingBlocks (target components).
Building Block	From configuration management perspective, this is the target of the single configuration items. This component is commanded to preload and activate the BuildingBlockConfigurations needed. This component reports back the status of preloading and activation.	The BuildingBlock preloads and activates new BuildingBlockConfigurations as commanded.
RU Planning System	<p>From configuration management perspective, the maintenance operator recognises, based on the information provided by this system, the appropriate point in time for the activation of the BuildingBlockConfigurations on vehicles.</p> <p>The information is used for the definition of the distribution-job.</p> <p>From the data provided by the RU Planning System the maintenance operator identifies when and for which period a specific train is not planned for operation.</p>	The system provides the operational planning information of relevant on-board instances.
Enterprise Resource Planning (ERP) - optional	<p>The ERP system can read the current BuildingBlockConfigurations installed on a physical BuildingBlock for asset management purposes.</p> <p>It may link this information to the equipment model (see equipment model in SDI).</p>	optional - read only to Sub-system at Maintenance Operator
Traffic Management	From configuration management perspective, the maintenance operator recognises, based on the information provided by this system, the appropriate point in time for the activation of trackside	The system provides the operational planning information of relevant

System	Description	Function
System (TMS)	BuildingBlockConfigurations. The information is used for the definition of the distribution-job. From the data provided by the TMS Management System the maintenance operator identifies when and for which period specific trackside equipment (a specific track area) is not planned for operational activities. TMS should also be informed about non-availabilities due to configuration changes.	trackside logical and physical instances.

Table 1 Description of the different involved systems

Interface ID	Description	Exchanged data
HMI 1	Human Machine Interface 1: Interface between a person in the BB supplier organisation (human being) and the system at BB supplier (system being described in a table above). <i>This interface is proprietary to the specific BB supplier and will not be addressed by Transversal CCS domain now. It could be useful to standardise some aspects later to reduce training costs.</i>	Information about the contents of available BuildingBlockConfigurations in the suppliers repository.
HMI 2	Human Machine Interface 2: Interface between the person in the integrator organisation (human being) and the system at integrator (system being described in a table above). <i>This interface is proprietary and will not be addressed by Transversal CCS domain now. It could be useful to standardise some aspects later to reduce training costs.</i>	Information relevant for the following activities: <ul style="list-style-type: none"> • Parametrisation of BuildingBlockConfigurations. • Ensure integrity of the dependencies of different BuildingBlockConfigurations between each other on different aggregation levels. • Publication of BuildingBlockConfigurations in a repository.
HMI 3	Human Machine Interface 3: Interface between the person in the operator organisation (human being) and the system at maintenance operator (system being described in a table above). <i>This interface is proprietary to the specific maintenance operator and will not be addressed by Transversal CCS domain now. It could be useful to standardise some aspects later to reduce training costs.</i>	Information relevant for the following activities: <ul style="list-style-type: none"> • Creation of distribution-job. • Show valid distribution-jobs • Select a valid distribution-job for execution ("will" of the Maintenance Operator). • Visualise the execution of a distribution-job: which BuildingBlockConfiguration versions need to change; preload status,

Interface ID	Description	Exchanged data
		<p>the activation and commit status of BuildingBlockConfigurations on BuildingBlocks</p> <ul style="list-style-type: none"> • Visualise the BuildingBlockConfiguration version history including all state changes (e.g. PreloadStatus and ActivationStatus).
HMI 4	<p>Human Machine Interface 4: Interface between the human Maintainer and the BuildingBlock.</p> <p><i>This interface is proprietary to the specific BuildingBlock and will not be addressed by Transversal CCS domain now. It could be useful to standardise some aspects later to reduce training costs.</i></p>	Information relevant for maintenance activities.
HMI 5	<p>Human Machine Interface 5: Interface between the human planner and the RU planning system.</p> <p><i>This interface is proprietary to the specific RU planning system and will not be addressed by Transversal CCS domain now. It could be useful to standardise some aspects later to reduce training costs.</i></p>	Information relevant for the operational planning of the different trains.
HMI 6	<p>Human Machine Interface 6: Interface between the human logistician and the enterprise resource planning (ERP).</p> <p><i>This interface is proprietary to the specific enterprise resource planning (ERP) and will not be addressed by Transversal CCS domain now.</i></p>	Information relevant for the management of the CCS asset.
HMI 7	<p>Human Machine Interface 7: Interface between the human signaller and the traffic management system (TMS).</p> <p><i>This interface is to a certain extent addressed by the SD4 team of Transversal CCS domain. The SD3 team (configuration management process) will not look to the details of this interface now.</i></p>	Information relevant for the traffic planning of the area of control covered by the specific system.
REPO	<p>Digital Interface REPO: This interface is used to publish and provide the BuildingBlockConfigurations to Integrators and the Maintenance Operator (repository).</p> <p>It will use the identical structure on all levels of the BuildingBlockConfiguration dependency tree.</p> <p><i>Comment: If e.g. data from a Digital Register need to be transferred in a specific version the respective integrator will create a BuildingBlockConfiguration (a binary</i></p>	<p>A repository URI will contain:</p> <ul style="list-style-type: none"> • configuration.json • configurationSafe.json • configurationFile (binary, optional) • hashes and signing information <p>See below for further information.</p>

Interface ID	Description	Exchanged data
	<p><i>configurationFile including the contents like the segment profiles, the configuration.json and configurationSafe.json files according to the chapters below). That BuildingBlockConfiguration is transferred to a Trackside Protection System as a BuildingBlock according to the processes described below.</i></p> <p>This interface is also used at the maintenance operator to release a distribution-job that contains the specific targets and the preloading and activation information.</p> <p>The distribution-job will only link the BuildingBlockConfigurations provided in the REPO interfaces.</p>	
SMI	<p>Digital Interface SMI (>= v3): This interface is used by the SFC to preload, activate and confirm BuildingBlockConfigurations on physical BuildingBlocks.</p> <p>The preload, activation and confirmation states must be available for visualisation to the maintenance operator during an update and must also be historised.</p>	<p>configurationJson configurationFile (binary, optional) hashes</p> <p>preload, activation and confirmation states</p> <p>see below for more details including sequence diagrams.</p>
DI 8	<p>Digital Interface 8: The information transferred over this interface is used by the maintenance operator to adjust the distribution-job to the operational plans of the single affected train units. Maintenance Operator needs to ensure that a specific train is not in operation when BuildingBlockConfigurations are activated.</p> <p><i>Task: at first there is no intention to standardise this interface as it has a lower priority compared to other interfaces involved in the configuration management process. Furthermore, the involved systems can be technologically quite different between different maintenance operators and planners. The workaround is to use "AI 3". At a later date this decision can be revised.</i></p>	Information relevant for defining the distribution- job of CCS on-board instances.
DI 9	<p>Digital Interface 9: Optional: the asset management system can read which BuildingBlockConfigurations are activated on which physical BuildingBlock.</p> <p>Task in SD3: at first there is no intention to standardise this interface as it has a lower priority compared to other interfaces involved in the configuration management process. Furthermore, the involved systems can be technologically quite different between different maintenance operators and logisticians. The workaround is to use "AI 2". At a later date this decision can be revised.</p>	needed for seamless integration of asset management systems (ERP)
DI 10		

Interface ID	Description	Exchanged data
	<p>Digital Interface 10: This interface is used by the maintenance operator to adjust the distribution-job to the operational plans of affected tracks. Maintenance Operator needs to ensure that tracks affected by the BuildingBlockConfiguration updates are not planned for operational functions when BuildingBlockConfigurations is activated.</p> <p><i>Task in SD3: at first there is no intention to standardise this interface as it has a lower priority compared to other interfaces involved in the configuration management process. Furthermore, the involved systems can be technologically quite different between different maintenance operators and signalers. The workaround is to use "AI 1". At a later date this decision can be revised.</i></p>	Information relevant for defining the distribution-job of CCS trackside instances.
DI 11	<p>Digital Interface 11: This interface is used by the logistician to update the enterprise resource planning (ERP) system based on the definition and installation result of the specific BuildingBlockConfigurations of a BuildingBlock.</p> <p><i>Task in SD3: currently there is no intention to standardise this interface as it has a lower priority compared to other interfaces involved in the configuration management process. Furthermore, the involved systems can be technologically quite different between different maintenance operators and logisticians. The workaround is to use "AI 2". At a later date this decision can be revised.</i></p>	Information relevant for managing the enterprise resource planning (ERP) system.
DI 12	<p>Digital Interface 12: This interface is used by the maintenance operator to distribute and publish BuildingBlockConfiguration dependency tree and the distribution-job information to the traffic management system (TMS).</p> <p>This interface is used to ensure alignment of topology data between the trackside BuildingBlockConfigurations and the TMS</p> <p><i>Task in SD3: currently the configurationFile (binary) that contains such information is not standardised. Probably it could be a solution to treat the TMS to be a BuildingBlock and use the same interface as SMI v3+.</i></p>	Topology data information that is used by CCS Building Blocks as well as by the TMS.
DI 13	<p>Digital Interface 13: Process the distribution-jobs to update the BuildingBlocks (target components). The time will finally be used for activation of a CCS configuration.</p> <p><i>Please note that for security and certificate validation the time service is required at all configuration management systems.</i></p> <p><i>This topic is also addressed by the Security domain that</i></p>	Time synchronisation information, being typically a time synchronisation protocol (Shared Cybersecurity Service SSI-NTS).

Interface ID	Description	Exchanged data
	<i>might already standardise this interface as a shared service.</i>	
AI 1	<p>Analogue Interface 1: This interface is used by the maintenance operator to synchronise with the signaller the operational plans of tracks affected by the installation of a CCS configuration. The interface is an exchange that can be in person, verbally by telephone or by other means.</p> <p>In the future this interface might be replaced by "DI 10".</p>	Information relevant for defining the distribution-job (activation information) of CCS trackside instances.
AI 2	<p>Analogue Interface 2: It can be used by the logistician to update the ERP based on the activation state of the BuildingBlockConfigurations on physical BuildingBlocks. The interface is an exchange that can be in person, verbally by telephone or by other means.</p> <p>In the future this interface might be replaced by "DI 9" and "DI 11".</p>	Information relevant for managing the enterprise resource planning (ERP) system.
AI 3	<p>Analogue Interface 3: This interface is used by the maintenance operator to synchronise with the planner the operational plans of single vehicles affected by the BuildingBlockConfiguration activations. The interface is an exchange that can be in person, verbally by telephone or by other means.</p> <p>In the future this interface might be replaced by "DI 8".</p>	Information relevant for defining the distribution-job (activation information) of CCS on-board instances.

Table 2 Description of the different interfaces

Actor	Description	Remarks
BB Supplier	<p>From configuration management perspective, this is the person within the BB Supplier organisation responsible to prepare and release a new building-block configuration (BBC) version to the Integrator. He manages within the BB Supplier organisation all configuration versions for all different building-blocks (products) sold by a specific supplier.</p> <p>Each BB Supplier shall be responsible for having tools and processes in place to guarantee the correctness (in terms of consistency of the delivered BBC artifacts and configuration versions) of a released building-block configuration (BBC).</p>	The different configuration versions are delivered in the Configuration Repository.
Integrator	<p>From configuration management perspective, this is the person within the Integrator organisation responsible to prepare and release a new top level building-block configuration (BBC) version to the Operator. For a specific CCS instance deployment, the Integrator shall be responsible for the overall BBC identified with a specific top level BBC version.</p> <p>Even though there are multiple integration steps that could be handled by different Integrators, for simplicity, this concept currently summarises all entities involved as Integrator. Typically, there could be an Integrator responsible for the safety relevant configuration and,</p>	The different configuration versions are delivered in the Configuration Repository.

Actor	Description	Remarks
	in a subsequent step, an Integrator adding some non-safety relevant configuration.	
Maintenance Operator	From configuration management perspective, this is the person within the Operator organisation (IM or RU) responsible to manage, distribute and install new building-block configuration (BBC) versions. He manages within the Operator organisation all configuration versions for all different building-blocks (products) that are in operation. This activity is performed remotely (where the different building-blocks are out of sight) out of a centralised maintenance operation centre.	The different configuration versions are managed in the Configuration Repository.
Maintainer	From configuration management perspective, this is the person within the Operator organisation (IM or RU) that installs a building-block configuration (BBC) locally (not remotely) while interacting directly with the building-blocks (building-blocks are in sight) where the installation occurs.	
Planner	From configuration management perspective, this is the person within the RU organisation responsible to plan the operational use of all available vehicles. He will have to plan the point in time when a new BBC can be installed on a specific vehicle, as this vehicle is not planned to be used for operation during this time period.	
Logistician	From configuration management perspective, this is the person within the IM or RU organisation responsible to centrally manage the whole asset (including the different BBC versions).	
Signaller	From configuration management perspective, this is the person within the IM organisation responsible to plan the efficient use of all available tracks. He will have to plan the point in time when a new BBC can be installed on a specific track (area of control), as this track is not planned to be used for operation during this time period.	

Table 3 Description of the different actors

[SPT2TS-130477]

4.2 System interfaces

4.2.1 REPO

See TCCS System Interface REPO

4.2.2 SMI (v3)

The SFC shall ensure that configuration operations are strictly isolated from operational runtime functions. Only configuration interfaces shall be used for deployment processes. [SPT2TS-130451]

See TCCS System Interface SMI (v3) [SPT2TS-130449]

4.3 System modes and states

ToDo: this content will be developed in the next remit phase.

5 System requirements

5.1 General Requirements

Support for minimal SIL footprint Automation

The SFC shall implement configuration and deployment logic with a minimal SIL footprint, leveraging non-safe (basic integrity) components whenever possible, while reserving safety logic only for safety-relevant checks and handovers.

[SPT2TS-130453]

Decoupling of configuration and operational functions

The implementation of the Service Function Configuration shall use configuration functions and interfaces only. There may be an inclusion of operational interfaces "behind the scenes" of the configuration process, but these are domain specific and not part of the System Requirement Specification.

Remark 1: it should be checked, if such a domain specific extension is really needed. A safety-related component can possibly fail safe at each point in time. So this is an availability topic more than a safety issue. And it should be taken into account, that the maintenance operator has checked the point in time for the update. If the domain comes to the conclusion that a domain specific "behind the scenes" is needed, then existing functions could be reused. E.g. in Trackside Assets the SCI interface could add a request from the field element to go into maintenance mode, and the existing `pdiForMaintenance` could be used as return.

Remark 2: E.g. the EULYNX approach uses both SMI and SCI for configuration. SCI is used to have a SIL4 function that the configuration versions of a field element and a the interlocking are consistent.

For safety and non-safety related configurations

This update mechanism shall be applicable for the update of safety-related and non-safety-related data / software. This means that the update mechanism of safety-related and non-safety-related data / software follows the same principles whereby specific (process) extensions (add-ons) for safety-related updates are required.

Minimum human interaction

The SFC implementation shall allow an update process with minimum human interaction. This means that the update process is basically initiated and verified by a human with automatic update processing.

The update process for non-safety-related data / software shall be embedded in the common update mechanism. This means that this update is basically initiated and verified by a human with automatic update processing based on non-safety system components.

The update process for safety-related data / software shall be embedded in a the common update mechanism. This means that this update is basically initiated and verified by a human with automatic update processing based on non-safety system components and (in addition) safety-system components.

Central Orchestration Support

The SFC shall support central orchestration of configuration updates. Local preloading may be supported, but all distribution processes must be controlled and initiated from a centralised SFC instance.

SFCs may cascade being responsible for a subtree of the dependency tree.

The implementation shall ensure the integrity and confidentiality and that authenticated configurations from identified sources are applied only.

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 SafeConfigurationAuthority (safe functions) or

ConfigurationManagementSystem (non safe functions) entity, and that the BuildingBlockConfiguration is targeted to the correct receiving entity.

The technical activation of BuildingBlockConfigurations as per the BuildingBlockConfiguration configuration.json documents shall be handled by each BuildingBlock independently.

The internal procedure(s) potentially strongly vary between different BuildingBlock suppliers. Hence, the actual activation mechanisms are not to be standardised.

The partitioning of the components in the logical architecture can depend on the availability and reliability of the network.

Example: for vehicles it could be appropriate to partition the ServiceFunctionConfiguration (SFC) incl. the SafeConfigurationAuthority (SCA) add-on on-board. For trackside a more centralised topology might be appropriate. [SPT2TS-129761]

5.2 Configuration Repository interaction and validation requirements

These requirements define the functions needed on the **Service Function Configuration (SFC)** to interact with the **Configuration Repository** during validation, distribution, and activation of BuildingBlockConfigurations (BBCs) and DistributionJobs. The SFC must access immutable, cryptographically signed documents (BuildingBlockConfigurations) and payloads to ensure safe and authorised configuration deployment.

Local Repository Update

The SFC shall use a **CREATE-only** operation to insert newly retrieved configurations into the local repository. It must never modify or delete existing versions but maintain versioned instances for rollback and traceability.

Canonical Path Access

The SFC shall access documents via deterministic, canonical paths derived from the structure:

```
/[repo-root]/[bbld]/[bbcld]/[bbcVersion]/configuration.json
/[repo-root]/[bbld]/[bbcld]/[bbcVersion]/configurationSafe.json
/[repo-root]/[bbld]/[bbcld]/[bbcVersion]/.* - for configurationFile payload
/[repo-root]/[bbld]/[bbcld]/[bbcVersion]/distribution/{distributionVersion}/distribution-job.json
```

These paths are used to locate artifacts for dependency resolution, activation orchestration, and trust validation.

Dependency Management

The SFC shall support recursive dependency resolution for all dependencies listed in the configuration.json of any BBC. The SFC must resolve dependencies until no further nested dependencies are referenced.

The SFC shall support HTTPS and enforce URI and certificate validation on repository access.

Recursive Retrieval and Caching

The SFC shall recursively retrieve all referenced BBC configuration.json documents and store them in the local configuration repository to ensure readiness for processing and distribution.

Creation in the Local Configuration Repository

Once dependencies are identified and retrieved (READ operation), the SFC shall store them in the local configuration repository (CREATE operation).

When retrieving a configuration and its resolved dependencies recursively, the SFC shall store these as read-only copies in the local configuration repository. The local repository serves as an available staging

area for dependency resolution before initiating distribution

If a dependency is declared with "configurationSafetyRelevance": "SAFE", the SFC shall check that the corresponding configurationSafe.json exists and that its configurationSafeHash is valid and linked in the parent configuration. Missing or invalid safe dependencies shall block the inclusion of such a BBC into the runtime deployment plan.

5.3 Dependency Tree validation requirements

This section defines how the Service Function Configuration (SFC) validates configuration artifacts retrieved from the Configuration Repository, ensuring the integrity and authenticity of the documents (BuildingBlockConfigurations) involved in a distribution job. These validations are essential to guarantee secure and trusted configuration deployment processes.

Dependency Availability Check

Before initiating a distribution job, the SFC shall verify that **all transitive dependencies** of the target BBC are available in the local configuration repository.

Signature Verification of configurationHash

For each BBC used in the dependency tree of the current distribution job, the SFC shall verify the authenticity and integrity of the configurationHashEncryptedSigned:

- Decode the JSON Web Token (JWT).
- Validate the digital signature using the public key of the Maintenance Operator Config Signer Certificate (OCSC) from the trusted Certificate Authority (CA)
- Confirm that the configurationHash and (if present) configurationFileTransferHostHash match the values in the referenced configuration.json.

Document integrity check

For each BBC, the SFC shall verify the integrity of the referenced document parts:

- Recalculate the canonicalised hash of the configuration.json
- Compare it to the value of configurationHash
- If present, recalculate the canonicalised hash of the transfer host section and verify it against configurationFileTransferHostHash

Payload Integrity and Authenticity

If a BBC references a payload via the configurationFile field, the SFC shall:

- Verify that the configurationFileHash matches the actual hash of the file
- Validate the configurationFileSignature against the configurationFileSignatureAlgorithm using the public key of the Manufacturer Config Signer Certificate (MCSC)

If the payload or signature is missing or invalid, the BBC must be marked as untrusted.

Validation of the Safe Dependency Tree

If a BBC or one of its dependencies is marked as SAFE, the SFC shall verify that the configurationSafeHash values provided in the parent configurationSafe.json are:

- Present
- Valid and match the recalculated hash of the dependent configurationSafe.json
- Signed using the JWT-based envelope with the MCSC

The complete dependency tree shall remain unchanged and consistent with the signed configurationSafeHash to preserve trust and traceability across integrator levels.

Validity of the configuration in regards to the BlackList

Before initiating a distribution job, for each BBC used in the Dependency Tree, the SFC shall check its presence in a Blacklist. If a BBC corresponds to one referred in said Blacklist, the SFC shall mark this BBC as [Blacklisted] and not authorise its preloading. All sub-dependencies of this BBC shall be removed

from the distribution job.

Note - The following is assumed with regard to blacklist: Issues occurred during operation of the railways system have to be reported to the European Railways Agency (ERA). It is supposed that ERA will then record the reported issues in a database. The blacklist could be generated out of the ERA database for reported issues (the blacklist is generated to prevent installation of configuration artifacts having major known issues). Basically, the ERA is maintaining the blacklist by having the ownership of the reported issues database.

ToDo next remit phase: The details of the blacklist including its functionalities (e.g. forbidden individual BBC versions, forbidden combinations of BBC versions) need to be defined.

5.4 Distribution requirements

5.4.1 Stages and Actors

It shall be possible to choose distribution-jobs that each are referencing a top-level BuildingBlockConfiguration version (Release) for distribution.

The configuration distribution process is characterised by the distribution of the BuildingBlockConfigurations (DistributionJob) and shall ensure the safety attestation, applicable for the safe BuildingBlockConfigurations.

The configuration distribution process is separated into the following stages:

Stage	Description
Preloading	<p>New BuildingBlockConfigurations are loaded to the BuildingBlock.</p> <p><i>Note: Background preloading without any restrictions is supported. The BB continues to operate normally during preloading.</i></p>
Release Activation	<p>Activation of the release - this is the Top Level BBC including its dependencies.</p>
Deactivation	<p>Incompatible safe BuildingBlockConfigurations are deactivated.</p> <p><i>Note: Incompatible are BuildingBlockConfigurations that will have a version switch considering the dependency tree of the new Top-Level BBC release version. In case of safe BuildingBlockConfigurations the SafeConfigurationAuthority ensures that all incompatible safe BuildingBlockConfigurations are deactivated before a version switch can be processed.</i></p>
Activation	<p>Preloaded BuildingBlockConfigurations are installed at the BuildingBlock. At this stage the activation of the Safe Top-Level BBC version occurs.</p> <p><i>Note: Background activation without any restriction is supported.</i></p>
Confirmation	<p>BuildingBlockConfigurations get the confirmation to return into operation.</p> <p><i>Note: In case of safe BuildingBlockConfigurations the SafeConfigurationAuthority ensures the safety attestation (post installation correctness) before the BuildingBlockConfiguration gets the confirmation (post installation confirmation) to go into operation.</i></p>

The configuration distribution process relies essentially on the following actors:

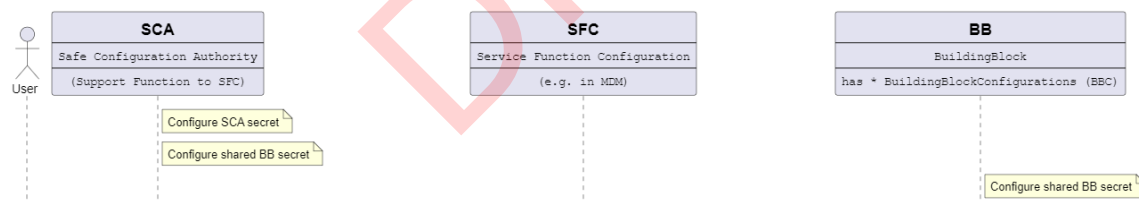
Actor	Description
Maintenance Operator	Maintenance Operator Responsible to express his "will" by selecting a distribution-job that links the top-level BuildingBlockConfiguration to be distributed.
SFC	Service Function Configuration Service function that realises the non-safety configuration functions; applicable for safe and non-safe BuildingBlockConfigurations. The SFC could be located in the MDM or a vehicle.
SCA	Safe Configuration Authority Safety function that realises the safety critical configuration functions; only applicable for safe BuildingBlockConfigurations. The SCA is a dedicated safe component within the SFC.
BB	Building Block A BuildingBlock can host safe and non-safe BuildingBlockConfigurations (BBC). For the further information refer to chapter Terms and definitions.



5.4.2 Sequences

5.4.2.1 Configuration distribution process stages

The configuration distribution process is separated into the following stages:

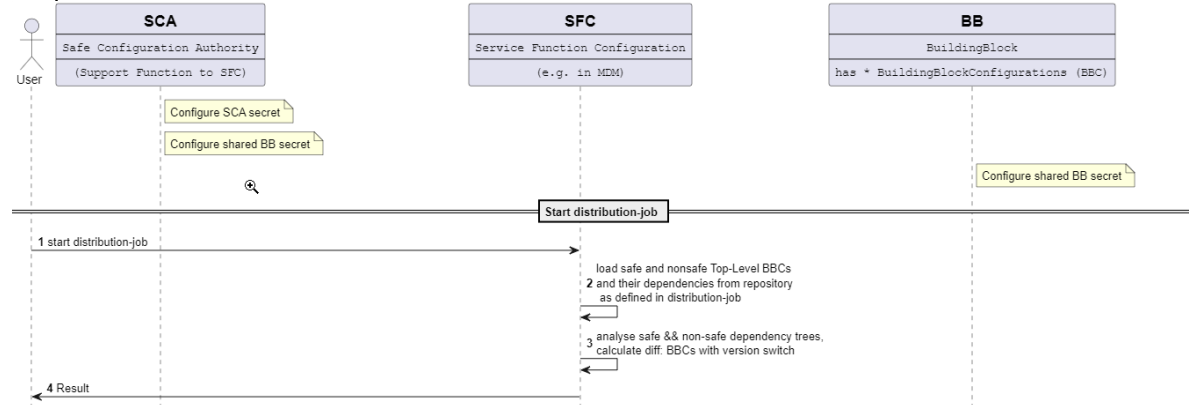
5.4.2.1.1 Preconditions



- BBC artifacts have been prepared and are accessible in the BBC repository via system interface REPO.
- Distribution-Job is prepared and accessible in the repository via system interface REPO.
- SCA: Shared BB secret for safe BuildingBlock (BB) provisioned in SCA. The shared secret that is known only to the SCA and the Building Block. It is used as part of the safety attestation process as shown in  SPT2TS-129876: for further details refer to stage activation (step 43) and confirmation (step 53).
- SCA: SCA secret for safe top level BBC activation provisioned in SCA and data preparation; for further details refer  SPT2TS-131282 (step 17 & 18).
- BuildingBlock (BB): Shared BB secret and unique bbId for safety attestation of safe BuildingBlockConfigurations provisioned in BuildingBlock (BB).

5.4.2.1.2 Start Distribution Job

Sequence

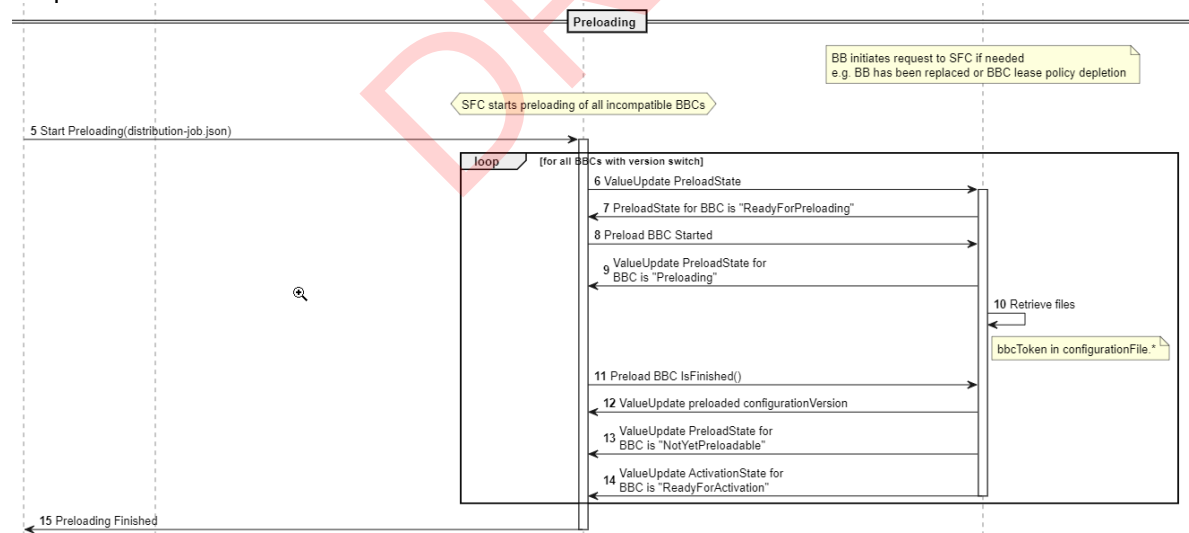


Start distribution-job

1. Start distribution-job. The distribution-job document contains a Top-Level BBC as dependency.
2. Load Top-Level BBC and its dependencies from repository as defined in distribution-job.
3. The SFC analyses the dependency tree and calculates which BBCs have a version switch (difference).
It also takes into account the Blacklist, to exclude any non-reliable configuration and its dependancies (to be discussed).
4. the Result is shown to the user: which safe and non-safe BBCs need an update.

5.4.2.1.3 Preloading

Sequence



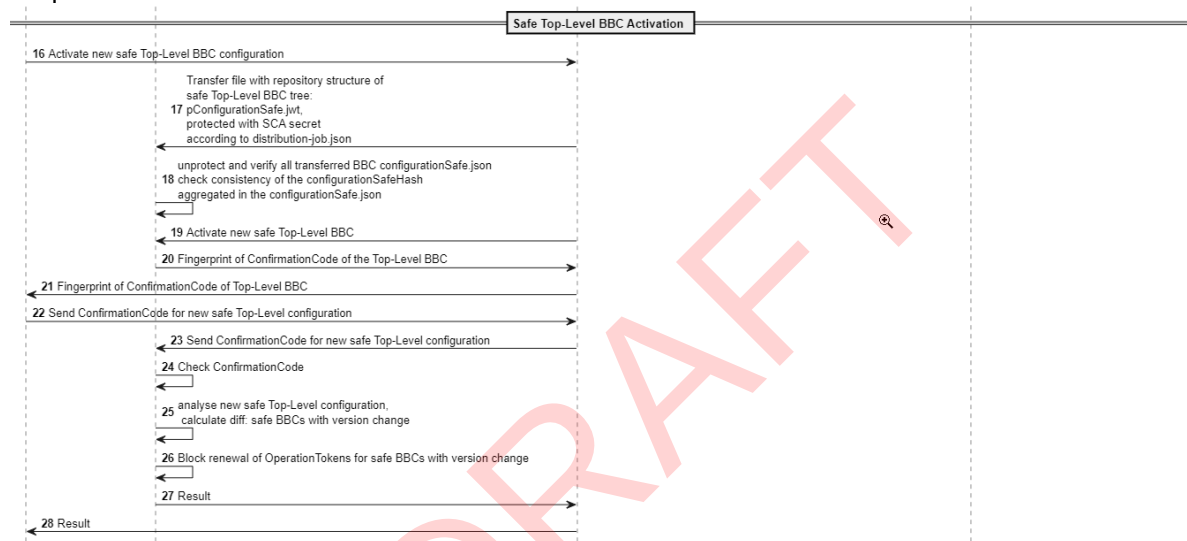
Preloading

5. Start preloading.
6. Get ValueUpdate PreloadState.
7. PreloadState for BBC is "ReadyForPreloading" expected
-- start the preloading of BBCs according to the diffs calculated in step 3.
8. Start BBC preloading: configuration.jwt, configurationFile.*, configurationFile.sig, public key, for safe

- BBC add pConfigurationSafe.jwt.
9. PreloadState is "Preloading" now.
 10. The files configuration.jwt, configurationFile.*, configurationFile.sig, public key and for safe BBC pConfigurationSafe.jwt are transferred.
 11. PreloadState is finished now.
 12. Value Update preloaded configurationVersion.
 13. PreloadState is "NotYetPreloadable".
 14. ActivationState is "ReadyForActivation".
 15. Finish Preloading.

5.4.2.1.4 Safe Top Level BBC Activation

Sequence



Safe Top-Level BBC Activation

16. The User chooses a safe Top-Level BBC corresponding to a preloaded distribution-job.
 17. The SFC transfers the complete safe Top-Level BBC dependency tree with all pConfigurationSafe.jwt to the SCA.
 18. The SCA checks the configurationSafe hashes, that are aggregated bottom up in the dependency tree.
 19. The SFC requests the activation of the new safe Top-Level BBC from the SCA.
 20. The SCA creates a Fingerprint of the ConfirmationCode (can only be decoded by the User).
 21. The SFC shows the Fingerprint of the ConfirmationCode to the User.
 22. If the User decides to proceed the User sends the ConfirmationCode for the new Top-Level configuration to the SFC.
 23. The ConfirmationCode is sent to the SCA.
 24. The SCA checks if the ConfirmationCode is valid.
 25. SCA analyses the dependency tree and calculates, which BBCs need to update the configurationVersion.
 26. The SCA knows which safe Top-Level-BBC is currently activated. Once the new safe Top-Level-BBC is under activation, the SCA stops giving OperationTokens to BBCs, until all the dependencies of the new safe Top-Level-BBC are completely activated.
- Knowing the currently installed safe Top-Level-BBC and the new safe Top-Level-BBC, the SCA is able to

calculate which BBC needs to be updated, and thus the list of BBCs for which new bbcHashPostInstallation have to be provided.

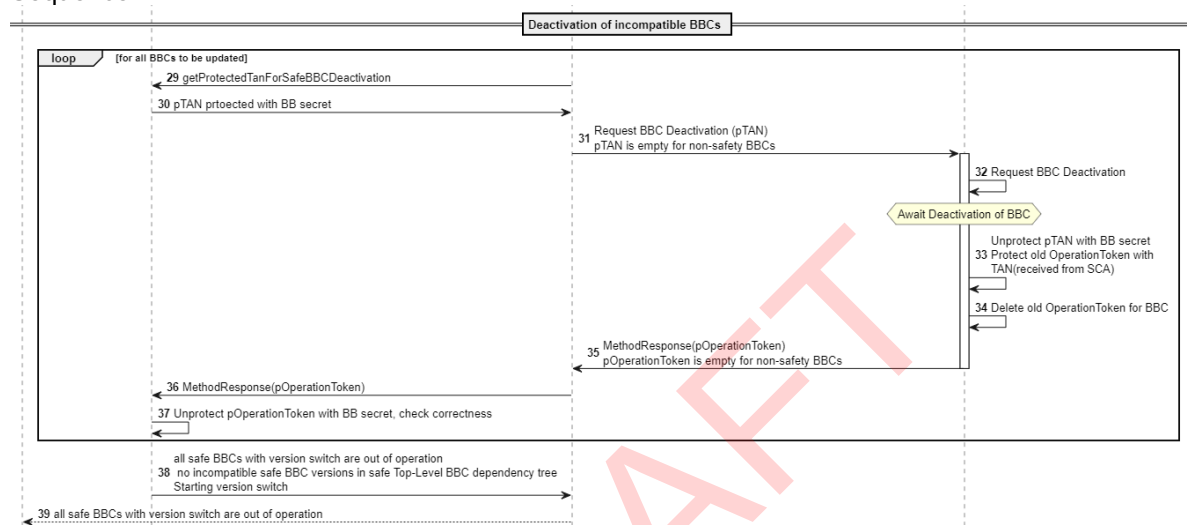
27. The SCA reports its status to the SFC.

28. The SFC reports the SCA status to the User.

ToDo Detailed specification of the ConfirmationCode and Fingerprint mechanism at the User-SCF/SCA interface.

5.4.2.1.5 Deactivation

Sequence



Deactivation of incompatible BBCs (only applicable for safety relevant updates)

29. The SFC requests a pTAN for a safe BBC (identified with bbld) at a specific Building Block (identified with bbld) to request BBC deactivation.

30. The SCA generates a TAN that is protected with the shared BB secret and responds with a protected TAN (pTAN).

31. Request to deactivate the BBC that needs to be updated, pTAN is a parameter (SFC is used as a "grey channel").

32. Request deactivation of the BBC that needs to be updated.

- BBC deactivation considering operational aspects.

33. BBC internal request processed: unprotect pTAN using shared BB secret, protect old OperationToken with TAN (received from SCA) and shared BB secret -> pOperationToken.

34. BBC delete OperationToken for safe BBCs.

35. MethodResponse (grey channel).

36. Deliver pOperationToken for safe BBC to the SCA.

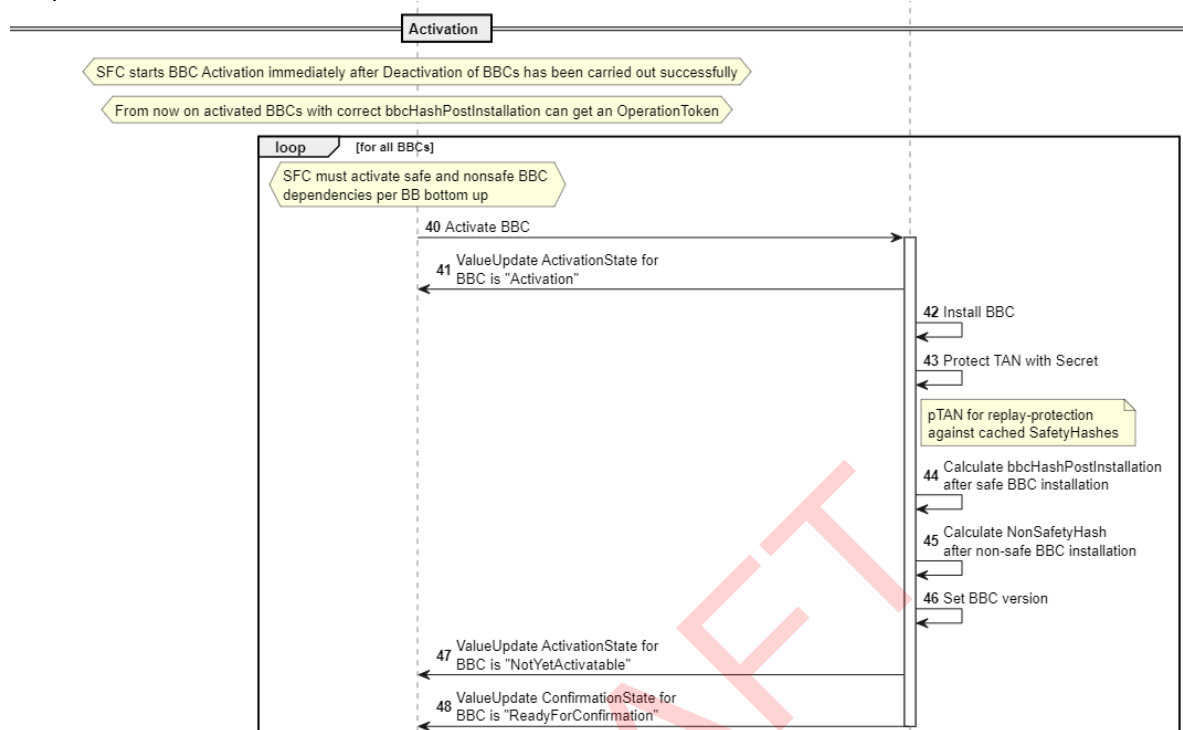
37. Unprotect pOperationToken using shared BB secret and TAN, check correctness.

38. All safe BBCs with version switch are deactivated: no incompatible safe BBC versions in safe Top-Level BBC dependency tree: now Starting version switch.

39. SFC reports to user.

5.4.2.1.6 Activation

Sequence



Activation

40. Activate BBC starts the installation of the corresponding BBC.

41. The ActivateState for the BBC is set to "Activation".

42. The BBC is installed. How the **installation** is done exactly is product specific.

43. A TAN is chosen randomly. The TAN is used to protect the safety hash against reply attacks, e.g. due to erroneous caching. The TAN **needs to be** sufficiently long to be unique (no collisions with previous TANs). The TAN is protected with a **preconfigured** secret (shared BB secret). In case of a safe BBC the preconfigured secret is only known by the SCA.

44. In case of a safe BBC the bbcHashPostInstallation of the BBC is calculated by the BB.

45. In case of a non-safe BBC the NonSafetyHash after installation of the BBC is calculated by the BB.

46. The new BBC configurationVersion is set.

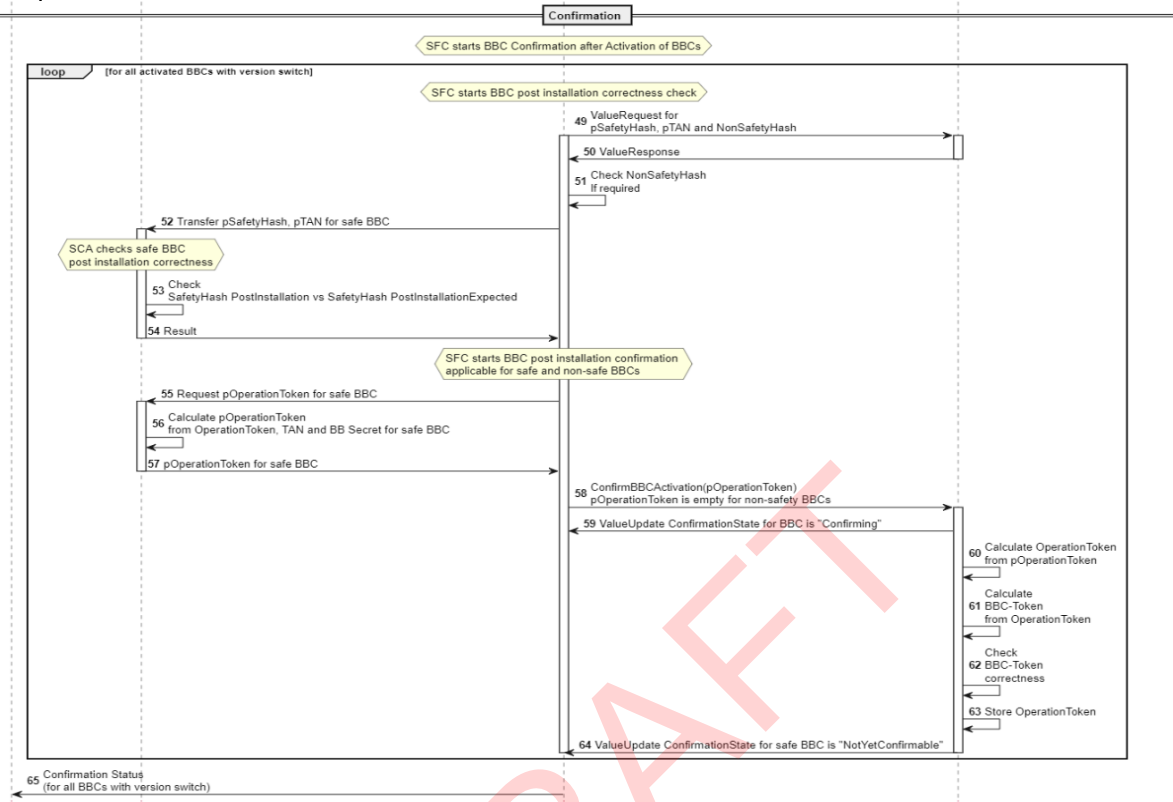
The hashes have to be unique for this BBC and they need to be reproducible. The hashes are used to prove that the installation of the BBC has been successfully done. The exact calculation is product specific, as it depends on the installation process. It must be pre-calculated.

47. Set ActivationState for this item to "NotYetActivatable", the activation is done.

48. Set ConfirmationState for this item to "ReadyForConfirmation", the confirmation starts.

5.4.2.1.7 Confirmation

Sequence



Confirmation

49. SFC requests values for BBC: pSafetyHash and pTAN, NonSafetyHash.
50. Read the response (grey channel in case of safe BBCs).
51. If non-safe BBC: SFC checks the NonSafetyHash (if required).
52. If safe BBC: Provide pSafetyHash and pTAN to the SCA.
53. SCA unprotects and checks the received SafetyHash (correlates with bbcHashPostInstallation) vs. the pre-calculated SafetyHash (correlates with bbcHashPostInstallationExpected) (that has been calculated before in data preparation for the BBC and can be found into the dependency tree rooting in the Top-Level-BBC).
54. Result of SafetyHash check
55. Request the new OperationTokens for all BBC that have been activated and successful post installation correctness check.
56. For each safe BBC: Calculate the protected operationToken (pOperationToken) by using the TAN (received during post installation correctness check; step 52), the operationToken and the SafetyHash.
57. Reply with the pOperationToken for safe BBC.
58. Provide the pOperationToken to the safe BBC via ConfirmBBCActivation. In case of non-safe BBC the pOperationToken is empty.
59. The BBC starts the confirmation process by setting BBC ConfirmationState to "Confirming".
60. The OperationToken is calculated from the pOperationToken by using the TAN that has been generated in step 43. The OperationToken is only applicable for this BBC located at this specific BB.
61. Calculate the bbcToken from the operationToken by using the bbld and the bbcHashPostInstallation.

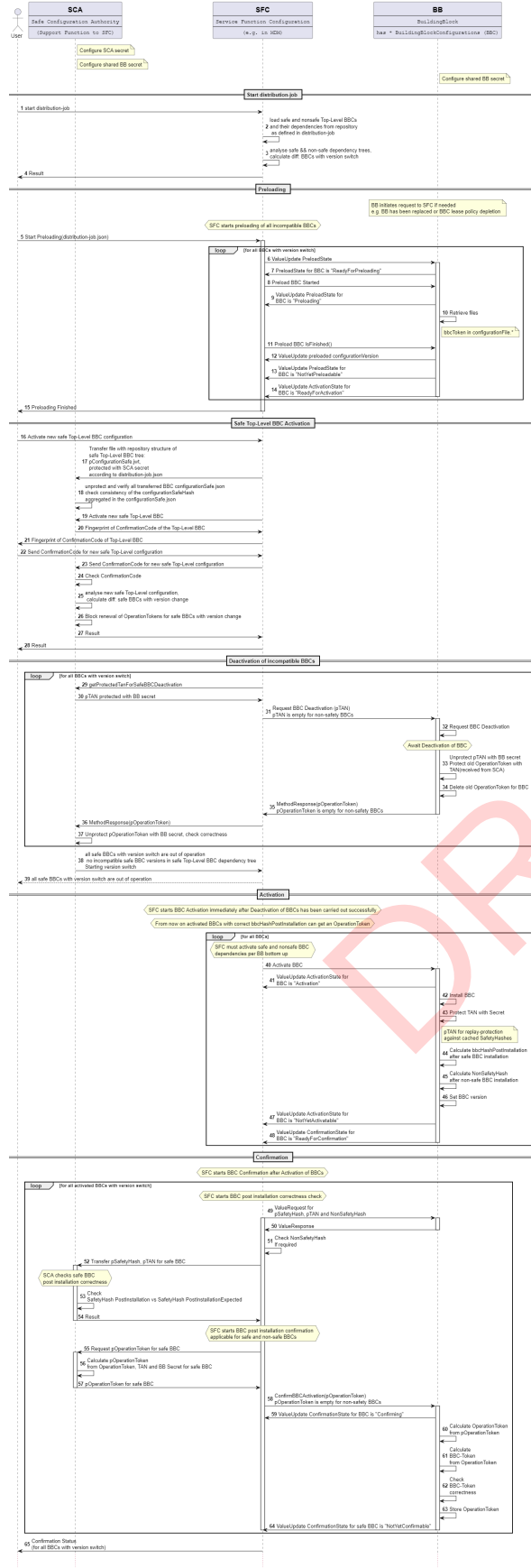
62. Check the bbcToken. The bbcToken have been transferred before in the configurationFile.* and is known to the entity responsible for the BBC only. If the bbcToken check fails, the operationToken (used for calculating the bbcToken) was meant for a different BB or BB is correct, but the BBC does not match.
63. If bbcToken check was successful, the OperationToken for this BBC is stored on the BB, so it can be checked whenever the BBC operation is enabled (e.g. during startup).
- If during startup no BBC OperationToken is known at the BB, it can request it from the SCA (through the SFC). If the installed BBC is correct, according to the current Top-Level-BBC, the SCA will reply (after successful post installation correctness check) with the OperationToken (through the SFC); if the installed BBC is incorrect, the BBC needs to be updated first before going into operation.
64. The ConfirmationState for the item is set to "NotYetConfirmable", the confirmation is done.
65. SFC reports to user.

DRAFT

5.4.2.2 Example: Top-Level BuildingBlockConfiguration version switch

DRAFT

Sequence



[SPT2TS-130481]

Pre-Conditions:

- BBC artifacts have been prepared and are accessible in the BBC repository via interface DI 1.
- Distribution-Job is prepared and accessible in the repository via interface DI 1.
- SCA: Shared BB secret for safe BuildingBlock (BB) provisioned in SCA.
- BuildingBlock (BB): Shared BB secret and unique bbld for safety attestation of safe BuildingBlockConfigurations provisioned in BuildingBlock (BB).

Start distribution-job

1. Start distribution-job. The distribution-job document contains a Top-Level BBC as dependency.
2. Load Top-Level BBC and its dependencies from repository as defined in distribution-job.
3. The SFC analyses the dependency tree and calculates which BBCs have a version switch (difference).
4. The Result is shown to the user: which safe and non-safe BBCs need an update.

Preloading

5. Start preloading.
6. Get ValueUpdate PreloadState.
7. PreloadState for BBC is "ReadyForPreloading" expected
- start the preloading of BBCs according to the diffs calculated in step 3.
8. Start BBC preloading: configuration.jwt, configurationFile.*, configurationFile.sig, public key, for safe BBC add pConfigurationSafe.jwt.
9. PreloadState is "Preloading" now.
10. The files configuration.jwt, configurationFile.*, configurationFile.sig, public key and for safe BBC pConfigurationSafe.jwt are transferred.
11. PreloadState is finished now.
12. Value Update preloaded configurationVersion.
13. PreloadState is "NotYetPreloadable".
14. ActivationState is "ReadyForActivation".
15. Finish Preloading.

Safe Top-Level BBC Activation

16. The User chooses a safe Top-Level BBC corresponding to a preloaded distribution-job.
17. The SFC transfers the complete safe Top-Level BBC dependency tree with all pConfigurationSafe.jwt to the SCA.
18. The SCA checks the configurationSafe hashes, that are aggregated bottom up in the dependency tree.
19. The SFC requests the activation of the new safe Top-Level BBC from the SCA.
20. The SCA creates a Fingerprint of the ConfirmationCode (can only be decoded by the User).
21. The SFC shows the Fingerprint of the ConfirmationCode to the User.
22. If the User decides to proceed, the User sends the ConfirmationCode for the new Top-Level configuration to the SFC.
23. The ConfirmationCode is sent to the SCA.
24. The SCA checks if the ConfirmationCode is valid.
25. SCA analyses the dependency tree and calculates, which BBCs need to update the

configurationVersion.

26. The SCA knows which safe Top-Level-BBC is currently activated. Once the new safe Top-Level-BBC is under activation, the SCA stops giving OperationTokens to BBCs, until all the dependencies of the new safe Top-Level-BBC are completely activated.

Knowing the currently installed safe Top-Level-BBC and the new safe Top-Level-BBC, the SCA is able to calculate which BBC needs to be updated, and thus the list of BBCs for which new bbcHashPostInstallation have to be provided.

27. The SCA reports its status to the SFC.

28. The SFC reports the SCA status to the User.

Deactivation of incompatible BBCs (only applicable for safety relevant updates)

29. The SFC requests a pTAN for a safe BBC (identified with bbclD) at a specific Building Block (identified with bblD) to request BBC deactivation.

30. The SCA generates a TAN that is protected with the shared BB secret and responds with a protected TAN (pTAN).

31. Request to deactivate the BBC that needs to be updated, pTAN is a parameter (SFC is used as a "grey channel").

32. Request deactivation of the BBC that needs to be updated.

- BBC deactivation considering operational aspects.

33. BBC internal request processed: unprotect pTAN using shared BB secret, protect old OperationToken with TAN (received from SCA) and shared BB secret -> pOperationToken.

34. BBC delete OperationToken for safe BBCs.

35. MethodResponse (grey channel).

36. Deliver pOperationToken for safe BBC to the SCA.

37. Unprotect pOperationToken using shared BB secret and TAN, check correctness.

38. All safe BBCs with version switch are deactivated: no incompatible safe BBC versions in safe Top-Level BBC dependency tree: now Starting version switch.

39. SFC reports to user.

Activation

40. Activate BBC starts the installation of the corresponding BBC.

41. The ActivateState for the BBC is set to "Activation".

42. The BBC is installed. How the installation is done exactly is product specific.

43. A TAN is chosen randomly. The TAN is used to protect the safety hash against reply attacks, e.g. due to erroneous caching. The TAN needs to be sufficiently long to be unique (no collisions with previous TANs). The TAN is protected with a preconfigured secret (shared BB secret). In case of a safe BBC the preconfigured secret is only known by the SCA.

44. In case of a safe BBC the bbcHashPostInstallation of the BBC is calculated by the BB.

45. In case of a non-safe BBC the NonSafetyHash after installation of the BBC is calculated by the BB.

46. The new BBC configurationVersion is set.

The hashes have to be unique for this BBC and they need to be reproducible. The hashes are used to prove that the installation of the BBC has been successfully done. The exact calculation is product specific, as it depends on the installation process. It must be pre-calculated.

47. Set ActivationState for this item to "NotYetActivatable", the activation is done.

48. Set ConfirmationState for this item to "ReadyForConfirmation", the confirmation starts.

Confirmation

49. SFC requests values for BBC: pSafetyHash and pTAN, NonSafetyHash.
 50. Read the response (grey channel in case of safe BBCs).
 51. If non-safe BBC: SFC checks the NonSafetyHash (if required).
 52. If safe BBC: Provide pSafetyHash and pTAN to the SCA.
 53. SCA unprotects and checks the received SafetyHash (correlates with bbcHashPostInstallation) vs. the pre-calculated SafetyHash (correlates with bbcHashPostInstallationExpected) (that has been calculated before in data preparation for the BBC and can be found into the dependency tree rooting in the Top-Level-BBC).
 54. Result of SafetyHash check
 55. Request the new OperationTokens for all BBC that have been activated and successful post installation correctness check.
 56. For each safe BBC: Calculate the protected operationToken (pOperationToken) by using the TAN (received during post installation correctness check; step 52), the operationToken and the SafetyHash.
 57. Reply with the pOperationToken for safe BBC.
 58. Provide the pOperationToken to the safe BBC via ConfirmBBCActivation. In case of non-safe BBC the pOperationToken is empty.
 59. The BBC starts the confirmation process by setting BBC ConfirmationState to "Confirming".
 60. The OperationToken is calculated from the pOperationToken by using the TAN that has been generated in step 43. The OperationToken is only applicable for this BBC located at this specific BB.
 61. Calculate the bbcToken from the operationToken by using the bbld and the bbcHashPostInstallation.
 62. Check the bbcToken. The bbcToken have been transferred before in the configurationFile.* and is known to the entity responsible for the BBC only. If the bbcToken check fails, the operationToken (used for calculating the bbcToken) was meant for a different BB or BB is correct, but the BBC does not match.
 63. If bbcToken check was successful, the OperationToken for this BBC is stored on the BB, so it can be checked whenever the BBC operation is enabled (e.g. during startup).
-- If during startup no BBC OperationToken is known at the BB, it can request it from the SCA (through the SFC). If the installed BBC is correct, according to the current Top-Level-BBC, the SCA will reply (after successful post installation correctness check) with the OperationToken (through the SFC); if the installed BBC is incorrect, the BBC needs to be updated first before going into operation.
 64. The ConfirmationState for the item is set to "NotYetConfigurable", the confirmation is done.
 65. SFC reports to user.
- [SPT2TS-130482]

5.4.3 Overall orchestration

Dependency Tree concept

The update process shall consider dependencies during configuration update. This means that there might be a sequence of installation whereby some components must be installed first before others can be installed.

Background information: The sequence of configuration is defined by the recursive dependency tree. If a BuildingBlockConfiguration is dependent on another BuildingBlockConfiguration, this BuildingBlockConfiguration must be installed first. The updates must start in the lowest levels (leaves) of the dependency tree. If all dependencies comply the next level up is being updated. This allows to cascade and aggregate the update process with entities each being responsible for a branch of the dependency tree.

Note: The higher levels are not actually updating anything. Their role is to define dependencies. If these dependencies are complying it is ensured that dependency trees of arbitrary size and depth have versions that comply to each other.

Example: A BuildingBlock might have two BuildingBlockConfiguration including (binary) configurationFiles as payloads. The firmware configuration has to be applied first, the parametrisation after that. So the firmware must be a dependency in the parametrisation configuration.json document.

Examples:

1. The BuildingBlockConfiguration of a field element may depend on BuildingBlockConfigurations of type firmware and parametrisation.
2. The BuildingBlockConfiguration of an interlocking may depend on BuildingBlockConfigurations of type field element and TPS.
3. The BuildingBlockConfiguration of an area of control may depend on BuildingBlockConfigurations of type TPS, EAL and ATO-TS

The concept of the BuildingBlockConfiguration configuration.json document is applicable to arbitrary number of dependency levels. [SPT2TS-129752]

Deactivation Top Down

When deactivating safe BuildingBlockConfigurations, the process shall begin from the top of the dependency tree and works its way down. The top-most safe BuildingBlockConfiguration, which depends on other safe BuildingBlockConfigurations, is taken out of operation first. Then, as you move down the tree, dependent BuildingBlockConfiguration are progressively deactivated. This ensures that no dependent safe BuildingBlockConfiguration is left running when its parent BuildingBlockConfiguration has already been taken out of operation, which could otherwise lead to errors or instability.

Activation Bottom-Up

The SFC shall enforce that updates are **activated bottom-up**, following the dependency order defined in configuration.json files. A higher-level BBC must only be activated when all its dependencies have been successfully activated.

When activating a configuration version, the process starts from the bottom of the dependency tree and works its way up. The lowest-level Activation, which are depended upon by other BuildingBlockConfigurations, are activated first. As moving up the tree, the BuildingBlockConfigurations that rely on these lower BuildingBlockConfigurations are then activated. This ensures that when a higher-level BuildingBlockConfiguration is activated, all its dependencies are already activated, providing a stable foundation for the system.



It shall be ensured, that a BuildingBlock is only in operation if its version complies as a dependency to the High Level BuildingBlockConfiguration. This implies as well, that all BuildingBlockConfigurations it depends on shall have a specified version.


The rollback shall use the identical process: a new distribution-job has to be created that references an unchanged older version.

Note: it could be useful to prepare such a distribution-job before the update to reduce the time needed in case the update fails.

Open question: how many versions should be stored in the BuildingBlocks - should at least the last version still be available to reduce the time needed for preloading etc.?

In case of a TopLevel BuildingBlockConfiguration version switch all incompatible BuildingBlockConfigurations shall be deactivated.

A BuildingBlock can only go into safe operation if all its safe BuildingBlockConfigurations get a confirmation from the SafeConfigurationAuthority. in accordance with  SPT2TS-131285 and  SPT2TS-129876.

A BuildingBlock can only stay in safe operation if all its safe BuildingBlockConfigurations have been confirmed by the SafeConfigurationAuthority and the BuildingBlock meets the lease policy ( SPT2TS-129869) valid for each of the safe BuildingBlockConfigurations.

Non-safe BuildingBlockConfigurations can go into operation once a confirmation from the ServiceFunctionConfiguration has been received.

The confirmation from the SafeConfigurationAuthority to the BuildingBlock can hold a lease policy for each of the safe BuildingBlockConfigurations.

The lease policy might hold conditions based on operational status like: onReboot, onReconnect, onMaintenance, inDepot (geofencing).

We may consider to add a temporal lease condition. If the temporal lease condition expires the BuildingBlockConfiguration needs a confirmation from the SCA to stay in operation. This could be useful if we need to take a BuildingBlockConfiguration out of operation, but this cannot be ensured in the process, e.g. we cannot connect to the BuildingBlock. In this case we could wait until the temporal lease condition has expired to be sure that the BuildingBlockConfiguration is out of operation. Then other BuildingBlockConfigurations with a version switch could be confirmed.

Note: The lease policy could be part of the operational token and needs to be prepared by the integrator. [SPT2TS-129869]

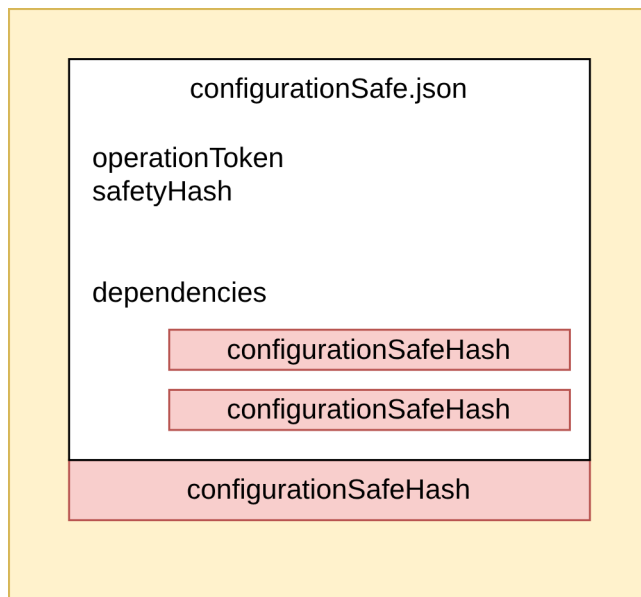
5.4.4 Safety Attestation Requirements

Safety attestation refers to the formal process of verifying and declaring that a safe BuildingBlockConfiguration installed on a BuildingBlock complies with the intended configuration version.

In order to reach this goal, the safety attestation is used to prove safely, that the correct BBC is installed at the intended BuildingBlock and the conditions for activation are fulfilled. During the update of a safe BuildingBlockConfiguration (BBC) on a BB different actors like SFC, SCA and the User are involved in the safety attestation process.

The principles shown here are applicable for safe BuildingBlockConfigurations only. [SPT2TS-129873]

For safe BuildingBlockConfiguration the attributes “operationToken” and “safetyHash” are composed during data preparation and added to the “configurationSafe.json” document. The document has been put to the repository corresponding to the ID of the BuildingBlockConfiguration (see above for definitions). The content of these attributes have been prepared by the responsible entity (Supplier or Integrator).



During data preparation the Supplier delivers the `bbcHashPostInstallationExpected`, the `bbcToken` and the `bbcid` of the `configurationFile` artifact for its generic model.

The Supplier also adds the `bbcToken` to the binary `configurationFile` that will be transferred to the BB later.

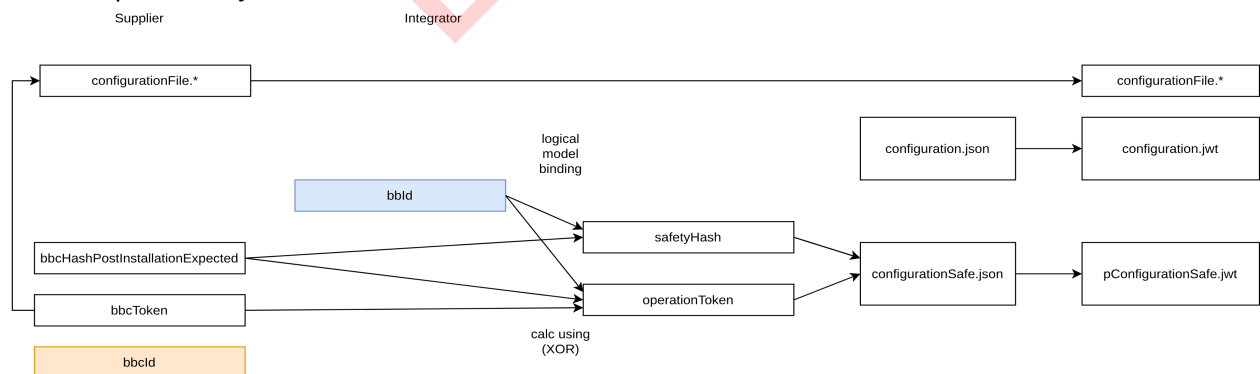
The Integrator creates the logical `bbcid` for the logical component he intends to use the generic model.

The binding of the logical component to the generic model is done with calculating:

- the `safetyHash` from the `bbcid` of the logical component and the `bbcHashPostInstallationExpected`.
- the `operationToken` from the `bbcid` of the logical component, the `bbcHashPostInstallationExpected` and the `bbcToken`

The `safetyHash` and the `operationToken` are added to the `configurationSafe.json` document that will be protected with the SCA secret.

The next picture only shows the flow of the attributes:



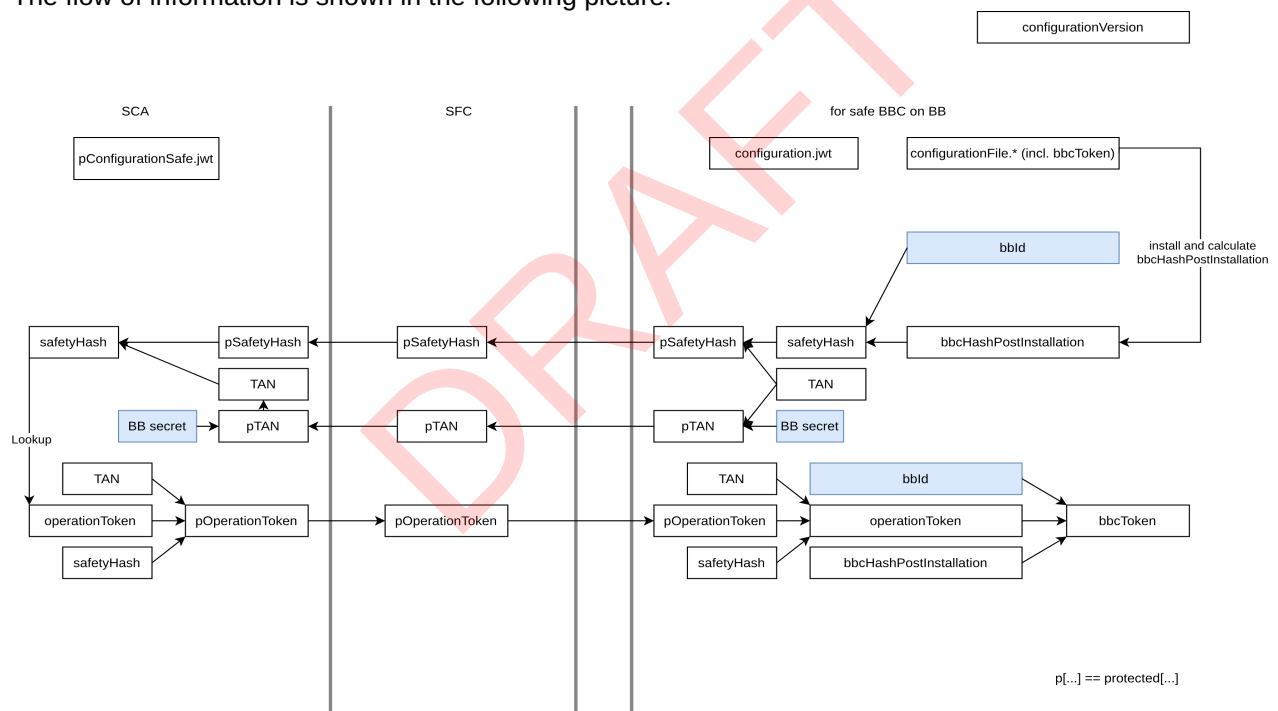
Basically the BB repeats the same calculations done in data preparation:

- The `safetyHash` is calculated from the `bbcid` assigned to the logical component and the `bbcHashPostInstallation` is calculated on the state of the filesystem after the installation. The `safetyHash` calculated on the safe computer of the BB will only be identical to the one calculated during data preparation if the `bbcHashPostInstallation` matches the `bbcHashPostInstallationExpected`.

Note: For example in a EULYNX object controller the bbld assigned to the logical component is stored on the basic data identifier which is plugged in the object controller. On a different building-block (component) the bbld (together with the basic data) might not be stored on a physical plug. For the configuration process it is not relevant where it is stored. The configuration process relies on the basic principle that a bbld is assigned to the logical component and can be used. The bbld is already needed for regular operation purposes.

- The randomly chosen transaction number (TAN), protected with the BB secret, ensures that the BB request to commit is uniquely connected to the SCA commit. The TAN is used to protect the safety hash against reply attacks, e.g. due to erroneous caching. The TAN needs to be sufficiently long to be unique (no collisions with previous TANs).
- The SCA checks if the safety hash calculated on and transferred from the BB matches the safety Hash calculated during data preparation. This safetyHash can be found in the protected pConfigurationSafe that can be read by the SCA only.
- If the safety Hash matches the SCA looks up the operationToken from the pConfigurationSafe.
- The SCA bundles the operationToken, the TAN, and the safetyHash into the protected pOperationToken to start the commit.
- Excluding the different values from the pOperationToken the bbcToken is calculated. The bbcToken is contained in the binary configurationFile of the supplier. It will only match if TAN, logical component and bbcHashPostInstallation is correct.

The flow of information is shown in the following picture:



5.4.5 Safe Configuration Authority Functions

The SCA shall check if BuildingBlockConfiguration versions comply to the **safe** High Level BuildingBlockConfiguration dependency tree. It shall commit safe BuildingBlockConfigurations only if all safe BuildingBlockConfigurations that are incompatible to the new Top-Level BuildingBlockConfiguration are out of operation.

The SCA shall check if the BuildingBlockConfiguration versions of a specific BuildingBlock comply to the **safe** High Level BuildingBlockConfiguration dependency tree on demand. In the following situations a

BuildingBlock might trigger an request to the SCA:

- The BuildingBlock is rebooting (e.g. after power loss, BuildingBlock exchanged, etc.)
- The lease time has expired
- At least one BuildingBlockConfiguration of the BuildingBlock has updated to a different version

The SafeConfigurationAuthority shall parse the safe dependency tree that it gets from data preparation in repositories and must calculate the intended BuildingBlockConfiguration versions according to the intended High Level BuildingBlockConfiguration ("will").

Note:

It could be useful to partition this function to the SCA so that the SCA is capable of switch from any version to any other version. If DataPreparation has this function then the possible version deltas have to be calculated in advance.

The SafeConfigurationAuthority must store persistently which BuildingBlockConfiguration versions are in operation on which BuildingBlock.

The SafeConfigurationAuthority must be able to safely recursively calculate the difference of the currently existing version of the dependency tree vs. intended version for safe configurations. As a result it will know which BuildingBlockConfigurations need to switch to which intended version.

The SafeConfigurationAuthority shall safely confirm that all BuildingBlocks with incompatible BuildingBlockConfiguration versions are out of operation.

Note: This is a necessary condition for the Top Level BuildingBlockConfiguration version switch that any new version can be confirmed to go into operation.

The SafeConfigurationAuthority must be able to report that the switch to the new version is possible now.

During the update process the SafeConfigurationAuthority shall monitor and store persistently, which BuildingBlock has activated which BuildingBlockConfigurations, or any exceptions during the activation attempt.

Note: On this basis a decision can be made to go into operation, although some BuildingBlocks are not operating because they could not activate the intended version.

The SafeConfigurationAuthority (SCA) shall lookup and provide the operational token to a BuildingBlock in case the activation of the intended BuildingBlockConfiguration version was successful.

The SafeConfigurationAuthority (SCA) shall provide the operational token to a BuildingBlock in case the BuildingBlock has the intended BuildingBlockConfiguration version.

Note: This function is needed during version changes AND in case a failed BuildingBlock is being replaced.

5.5 OPC UA Security Requirements

The OPC UA endpoint in the BuildingBlock shall support the following permissions:

Permission Name	Description	Corresponding OPC UA Node Permissions(*)
eu.rail.smi.configuration-read	Permission to read OPC UA nodes containing configuration-related information.	Browse Read
eu.rail.smi.configuration-distribute	Permission to read OPC UA nodes and execute OPC UA methods (e.g. OPC UA File Type) to distribute configuration data to the component.	Browse Read Execute
eu.rail.smi.configuration-activate	Permission to read OPC UA nodes and to execute OPC UA methods to process changes on the component.	Browse Read Execute
eu.rail.smi.component-reset	Permission to execute OPC UA method to process a component reset.	Browse Execute
		(*) If required, the list of OPC UA node permissions might be extended with additional OPC UA node permissions.

5.6 Non-functional requirements

5.6.1 SFC

ToDo

5.6.2 BuildingBlock

ToDo

5.7 Functional requirements

5.7.1 SFC

The SFC shall take a valid Blacklist into consideration to validate all BBC in the dependency tree. The blacklist contains all BBC with safety relevant issue to be rejected. This requirement for a Blacklist has been identified as a result of the risk analysis for the SFC. See [SPT2TS-130480]. It serves a mitigation for the cases where :

- A BuildingBlockConfiguration with known safety relevant issues is preloaded.
- A BuildingBlock preloaded with a BuildingBlockConfiguration that has known safety relevant issues is activated/set in operation.

The validity of the blacklist is determined by its lease time.

Notice : An additional constraint can be added to the previous phase (configuration integration) of the configuration management process. To be discussed.

ToDo next remit phase: The details of the blacklist including its functionalities and related impact on the

sequence need to be defined.

5.7.2 BuildingBlock

The supplier shall ensure that the configuration of its BuildingBlock is safe according to the SIL levels assigned to the functions within the system border of the BuildingBlock, if the generic product is configured within the limits defined in the safety case of the supplier according to EN 50716:2023.

ToDo At a proper reference to the standard (i.e. link to the PRAMS project)

The <dependencies> are made explicit in the configuration.json of the BuildingBlockConfiguration. The concept allows to assign responsibilities to different levels of the dependency tree. This allows that the in many cases the competences on the different levels will be in line with the responsibilities.
[SPT2TS-129754]

A BuildingBlock shall request the SafeConfigurationAuthority for approval to go into operation again in the following cases:

- The BuildingBlock is rebooting (e.g. after power loss, BuildingBlock exchanged, etc.)
- The lease time has expired
- At least one BuildingBlockConfiguration of the BuildingBlock has updated to a different version

6 Risk analysis

The risk analysis for the configuration process described in this document can be found attached to this work item. This risk analysis is performed through FMEA method, in accordance with PRAMS **[version 2, version 1 corresponds to an older 4.1 schema]**. This version 2 takes the sequence diagram into account as it is currently the lowest possible level of function description.

The current state of the risk analysis identifies the following possible failure modes as **safety relevant issues**. The severity category of these issues is considered **[catastrophic]**.

Step number in the sequence diagram	Function	Failure mode	Output	Linked function	Effect on linked function	Effect on the SUC	Effect on the railway system	Estimated THR*	Possible mitigation
2, 3, 4	Service Function Configuration (SFC) Generate Distribution Job (See 5.4.2.1.2)	Incorrect	REPO	Self, human user	Wrong version switch is calculated from the distribution job.	Worst case : Wrong top level version is deployed.	A safety relevant command during operation can be executed based on unreliable configuration data.	9.71E-07	The development of this function shall follow the requirements for tools, defined in EN50716 standard.
8	Service Function	Incorrect	SMI (V3+)	Preload, activate	No error occurs	A config	A safety	9.71E-07	Blacklist of all

Step number in the sequence diagram	Function	Failure mode	Output	Linked function	Effect on linked function	Effect on the SUC	Effect on the railway system	Estimated THR*	Possible mitigation
	Configuration (SFC) Preloading (See 5.4.2.1.3) File transfert			and commit Building Block Configuration Preloading (See 5.4.2.1.3) File transfert	but a configuration with safety issues identified might be preloaded in the BB.	uration with safety issues is deployed.	relevant command can be executed based on unreliable configuration.		identified BBC with safety relevant issues. This blacklist is an input of the SFC.
35	Preload, activate and commit Building Block Configuration Deactivation (See 5.4.2.1.5) MethodResponse	Commission	SMI (V3+)	Service Function Configuration (SFC) Deactivation (See 5.4.2.1.5) MethodResponse	The SFC receives the confirmation of the empty pOperationToken too early or at a wrong time.	The BB sends the confirmation before being deactivated but after the request for the SFC. An incompatible safe BBC might stay in operation.	A safety relevant command can be executed based on unreliable configuration.	9.71E-07	<i>The development of this function shall follow the requirements for tools, defined in EN50716 standard.</i>
38	Safe Configuration Authority (SCA) Deactivation (See 5.4.2.1.5) Starting	Commission	Internal	Service Function Configuration (SFC) Deactivation (See 5.4.2.1.5)	The SFC receives the version switch confirmation too early or at a	The version switch process might start although the	A safety relevant command can be executed based on	9.71E-07	<i>The development of this function shall follow the requirements for tools, defined in</i>

Step number in the sequence diagram	Function	Failure mode	Output	Linked function	Effect on linked function	Effect on the SUC	Effect on the railway system	Estimated THR*	Possible mitigation
	version switch			Starting version switch	wrong time.	SCA hasn't fully checked the correctness. An incompatible safe BBC might stay in operation.	unreliable configuration.		EN50716 standard.
40	Service Function Configuration (SFC) Activation (See 5.4.2.1.6)	Incorrect	SMI (V3+)	Preload, activate and commit Building Block Configuration Activation (See 5.4.2.1.6)	No error occurs but a configuration with safety issues identified might be preloaded in the BB.	A configuration with safety issues is deployed.	A safety relevant command can be executed based on unreliable configuration.	9.71E-07	Blacklist of all identified BBC with safety relevant issues. This blacklist is an input of the SFC.
54	Safe Configuration Authority Confirmation (See 5.4.2.1.7) Send hash check result for Safe BBC	Incorrect	SMI (V3+)	Service Function Configuration (SFC) Confirmation (See 5.4.2.1.7) Send hash check result for Safe BBC	The SFC receives wrong result from SCA check.	The SFC might consider a Safe BBC with wrong hash to be activated. The SFC might exclude a Safe	A safety relevant command can be executed based on unreliable configuration. Availability issue if a	9.71E-07	The development of this function shall follow the requirements for tools, defined in EN50716 standard.

Step number in the sequence diagram	Function	Failure mode	Output	Linked function	Effect on linked function	Effect on the SUC	Effect on the railway system	Estimated THR*	Possible mitigation
						BBC with correct hash.	correct Safe BBC is excluded.		
57	Safe Configuration Authority Confirmation (See 5.4.2.1.7) Send pOperationToken result for Safe BBC	Incorrect	SMI (V3+)	Service Function Configuration (SFC) Confirmation (See 5.4.2.1.7) Send pOperationToken for Safe BBC	The SFC receives wrong token from SCA check.	The SFC might consider a wrong Safe BBC to be activated.	A safety relevant command can be executed based on unreliable configuration.	9.71E-07	<i>The development of this function shall follow the requirements for tools, defined in EN50716 standard.</i>

*For a rough estimation of the Tolerable Hazard Rate, we considered the following number :

- 50 millions BuildingBlocks for field elements ;
- 30000 trains with a mean of 50 BuildingBlocks per train : 1,5 millions BuildingBlocks for rolling stock ;
- Yearly updates for each buildingblock for both fields elements and rolling stock ;
- 1 hour of functioning of the SFC per update ;
- Up to 100 dangerous failures per year.

These numbers can be reconsidered with more accurate data.

The formula used is the following :

$THR = [Number\ of\ dangerous\ failures\ per\ year] / ([time\ of\ operation\ per\ year] * [number\ of\ buildingblocks\ updated])$

[SPT2TS-130480]

7 Application conditions

The DataPreparation must create safe consistent dependency trees including all BuildingBlockConfiguration versions starting from a safe High Level BuildingBlockConfiguration and put this in repositories. [SPT2TS-129866]



*Note to author: Specify functional system requirements derived from dedicated specifications / inputs from architectural design that are **applicable to each system function**. If it seems more appropriate not to include these requirements in a separate chapter, but to include them in other chapters in order to maintain the contextual coherence, a corresponding note can be inserted here instead.*

Add requirements that are valid for multiple functions here instead of keeping them multiple times in the subchapters. Consider to create a new chapter for these requirements if necessary.

7.1.1 Functional overview

Note to author: Provide a high-level functional overview diagram.

7.1.2 <System function X>

*Note to author: Define the full set of function-specific requirements for necessary to perform system function X, including inputs and outputs of the function. Specify  SPPR-4173 - Functional requirements and  SPPR-4174 - Non-functional requirements **applicable to the considered system function.***

If needed or appropriate, create separate chapters for each non-functional requirements type. If it seems more appropriate not to include these requirements in a separate chapter, but to add them to the functional requirements in order to maintain the contextual coherence, a corresponding note can be inserted here instead.

DRAFT