




ERJU SYSTEM PILLAR

System Interface Description_TCCS- System Interface REPO (SERA Version)



System Interface Description_TCCS-System Interface REPO (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 describes the Configuration Interface required by EN 50126-1:2017 Phase 5 (Architecture and Apportionment of System Requirements) between the Data Preparation Toolchain and the Service Function Configuration (SFC) system. It enables secure, traceable configuration deployment through: Structured files (configuration.json, configurationSafe.json, distribution.json, distribution-job.json) Canonical, machine-readable representations with hierarchical versioning Cryptographically verifiable artifact integrity and authorship Deterministic, read-only repository storage for reproducibility and auditability The interface is designed for modular responsibility: suppliers author low-level configurations, integrators assemble higher levels, and operators manage controlled deployment across physical assets and operational scenarios. |
| Config Item | System Interface Description |
| Document ID | TCCS Service Function Configuration _SFC_ L5/TCCS System Interface REPO#723726  System Interface Description_TCCS-System Interface REPO (SERA Version) |
| Classification | Public |
| Status | In Review by System Pillar |
| Version | 1.0 |
| Revision | 723726 |
| 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

DRAFT

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 |
| Approvals | LOSTUN Virgil : Waiting , Boryana Tezgetarska : Waiting , Kilian Jens : 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 |
| 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 | 5 |
| 1.1 Scope and intended audience | 5 |
| 1.2 Purpose | 5 |
| 1.3 Glossary | 6 |
| 2 Overview | 6 |
| 2.1 Overall description | 6 |
| 2.2 Non-functional characteristics / non-functional requirements | 6 |
| 3 REPO REST API | 7 |
| 3.1 Data description | 9 |
| 3.1.1 Configuration Metadata: configuration.json | 10 |
| 3.1.1.1 Core Concept Requirements | 10 |
| 3.1.1.2 Signing Specification Requirements | 11 |
| 3.1.1.2.1 Scope | 12 |
| 3.1.1.3 Summary | 13 |
| 3.1.1.4 Schema document | 14 |
| 3.1.2 Configuration Metadata: configurationSafe.json | 19 |
| 3.1.2.1 Core Concept Requirements | 19 |
| 3.1.2.2 Signing Process Requirements | 21 |
| 3.1.2.3 Schema Document | 23 |

| | |
|---|----|
| 3.1.3 distribution-Job | 27 |
| 3.1.3.1 Core Concept requirements | 27 |
| 3.1.3.2 Signing Process requirements | 29 |
| 3.1.3.3 Schema Document | 30 |
| 3.2 Interace description | 33 |
| 3.3 Behaviour description | 33 |
| 3.4 Interdependencies to other interface layers | 34 |
| 4 Appendix | 34 |
| 4.1 Input documents | 34 |
| 4.2 Standards and References | 34 |
| 5 Workspace for discussions, actions and issues | 34 |
| 6 Scope of interface constraints | 35 |
| 6.1 Role of interface | 35 |

1 Preamble

1.1 Scope and intended audience

This document defines the interface for structured configuration artifact exchange between configuration-authoring entities (suppliers, integrators, operators) and Service Function Configuration (SFC) consumers. It applies to the full lifecycle of configuration data, including safety-related payloads, safe dependency validation, and distribution execution.

Intended audience includes:

- Suppliers and Integrators – preparing configuration content and metadata (configuration.json, configurationSafe.json) and signing it during data preparation
- Operators – authoring distribution.json and distribution-job.json to control the application of configurations
- SFCs – interpreting and verifying configurations prior to controlled activation
- PKI administrators – managing trust anchors (public keys) for verification

1.2 Purpose

This document describes the Configuration Interface required by [EN 50126-1:2017] Phase 5 (Architecture and Apportionment of System Requirements) between the Data Preparation Toolchain and the Service Function Configuration (SFC) system. It enables secure, traceable configuration deployment through:

- Structured files (configuration.json, configurationSafe.json, distribution.json, distribution-job.json)
- Canonical, machine-readable representations with hierarchical versioning
- Cryptographically verifiable artifact integrity and authorship
- Deterministic, read-only repository storage for reproducibility and auditability

The interface is designed for modular responsibility: suppliers author low-level configurations, integrators assemble higher levels, and operators manage controlled deployment across physical assets and operational scenarios.

1.3 Glossary

| Term | Description |
|------------------|--|
| Data Preparation | Authoring phase where suppliers/integrators create and sign configuration artifacts. Also referred to as configuration-authoring. |
| JWT | JSON Web Token used to sign hashes of configuration files. Each artifact has a signature generated using the private key from an MCSC or OCSC certificate. |
| MCSC / OCSC | Manufacturer Config Signer Certificate / Operator Configuration Signer Certificate used to sign and verify artifacts via Shared Security Services PKI. |

2 Overview

2.1 Overall description

The configuration interface is read-only from the point of view of the SFC, and write-only during the data preparation phase.

Data preparation is performed by authorized entities:

- Suppliers (for firmware or other low-level payloads)
- Integrators (for assembling parameterizations and higher-level BBCs)
- Operators (for distribution orchestration)

All artifacts are written into a deterministic, immutable repository structure organized by:

```
/<bbld>/<bbcld>/<bbcVersion>/
```

- |— configuration.json
- |— configurationSafe.json (if payload is safety related)
- |— [*.*] payload

All signature objects (JWTs) follow RFC 7515 and are created using private keys managed under the Shared Security Services PKI. The public key must be available in the same folder.

2.2 Non-functional characteristics / non-functional requirements

Safety and RAMS

- Signature chains guarantee that each configuration and safe hash was reviewed and explicitly released
- Safety-related dependencies use recursive signed hashes (configurationSafeHash) to ensure upstream visibility and traceability

Cybersecurity

- All critical artifacts are signed with certificates (MCSC, OCSC)
- configurationSafe.json is encrypted and inaccessible to non-safe systems

Performance and Integrity

- Use of deterministic JSON (RFC 8785) ensures signature reproducibility
- Immutable repository layout prevents accidental overwrites or deletions

- Read access is lightweight (GET-only)

Deployment Flexibility

- distribution-job.json defines preload and activation windows
- Activation can be triggered by operator or automated decision logic (e.g. GOA4)
- Offline-capable via local repository sync

3 REPO REST API

The interface layer defines the standardized REST-based access to configuration artifacts within a distributed, version-controlled configuration repository. It provides a clear contract for the creation and retrieval of configuration packages associated with hierarchical BuildingBlockConfigurations.

[SPT2TS-130393]

This interface must ensure immutability: Once published, configuration versions cannot be modified or deleted. If a change is needed, a new version must be created.

[SPT2TS-130395]

This interface must ensure traceability: All configurations are uniquely identified via a combination of bbld, bbclid, and bbcVersion. These identifiers follow a dot-separated convention (e.g., de.db.signaling) and are internally mapped to slash-separated folder structures. [SPT2TS-130391]

This interface must ensure separation of concerns: Each level in the configuration hierarchy encapsulates:

- The main configuration.json, describing dependencies and metadata
- Optional raw payload files (e.g., firmware binaries or parameter sets)
- Optionally, an encrypted and signed configurationSafe.json, if the configuration is safety-relevant

[SPT2TS-130392]

This interface must support the following operations in exposing two primary endpoints per configuration version via HTTPS only:

- GET: Retrieve metadata and file locations for a given configuration version.
- PUT: Upload a new configuration version (including metadata and payloads), implemented by using a quote identifier.

[SPT2TS-130389]

The API is designed must support integration with CI/CD workflows, release automation, and version-controlled publishing of configuration artifacts by suppliers and integrators. It ensures secure access via bearer token authentication and enables consistent verification, audit, and deployment across distributed systems.

[SPT2TS-130390]

Below is the normative OpenAPI specification for the Configuration Repository API.

openapi: 3.1.0

info:

title: Configuration Repository API

version: 1.0.0

description: >

REST API for managing immutable configuration repository contents. Supports creation (upload) and retrieval (read) of configuration payloads and metadata at various hierarchy levels.

- No update or delete operations are allowed.
- Content is versioned and immutable: new versions must be created instead of updating existing ones.
- Each configuration level may contain:
 - `configuration.json` (mandatory)
 - `configurationSafe.json` (encrypted, mandatory if safety-relevant)
 - Raw payload file (e.g., .zip, .bin, .xml) (optional)

paths:

/repository/{bbld}/{bbclD}/{bbcVersion}/:

parameters:

- name: bbld

in: path

required: true

description: >

Unique BuildingBlock identifier. Dots (.) represent hierarchical structure and will be translated into slashes (/) in the actual repository path.

Example: `de.db.signaling` → `/de/db/signaling`

schema:

type: string

- name: bbclD

in: path

required: true

description: >

Unique BuildingBlockConfiguration identifier. Like bbld, dots (.) represent hierarchy and are mapped to slashes in the resource path.

Example: `interlocking.firmware` → `/interlocking/firmware`

schema:

type: string

- name: bbcVersion

in: path

required: true

description: Semantic version (e.g., 1.2.3) of the configuration.

schema:

type: string

get:

summary: Get full metadata and file listing for the configuration version

responses:

'200':

description: Metadata and listing retrieved successfully

content:

application/json:

schema:

type: object

properties:

configuration:

type: string

description: URI to configuration.json

configurationSafe:

type: string

description: URI to encrypted configurationSafe.json (if applicable)

payloadFiles:

type: string

description: URI to payload file (e.g., firmware, params)

'404':

description: Configuration not found

/repository/{bbld}/{bbclD}/{bbcVersion}/upload:

post:

summary: Upload configuration package for a new version

requestBody:

required: true

content:

multipart/form-data:

schema:

type: object

properties:

configuration:

type: string

format: binary

description: configuration.json file (required)
 configurationSafe:
 type: string
 format: binary
 description: Encrypted configurationSafe.json file (optional)
 payload:
 type: string
 format: binary
 description: Payload file (.zip, .bin, etc.)
 required:
 - configuration
 responses:
 '201':
 description: Configuration version uploaded successfully
 '409':
 description: Version already exists (immutable storage)

components:
 securitySchemes:
 bearerAuth:
 type: http
 scheme: bearer
 bearerFormat: JWT
 security:
 - bearerAuth: []
 [SPT2TS-130394]

3.1 Data description

The configuration process involves the creation and structured documentation of versioned configuration artifacts. These artifacts are organized along a hierarchical dependency tree that reflects the functional decomposition of systems into BuildingBlocks. The following key document types are defined and handled at distinct stages of the configuration lifecycle. [SPT2TS-129687]

Each BuildingBlockConfiguration must describe the intended configuration absolutely. No relative updates, always complete; dependencies must be explicit. [SPT2TS-129750]

During data preparation, both suppliers and integrators contribute to the authoring of configuration artifacts across different levels of the dependency tree. This includes:

- The core payload files (e.g., firmware, parameter sets) that represent the actual configuration data
- The corresponding configuration.json documents, which define metadata, semantic versioning, and dependencies in a standardized format

These configuration packages are immutable and uniquely identified using a combination of bbld, bbcld, and bbcVersion. [SPT2TS-130398]

Some of the BBC configuration payloads are safety-related. The authoring entity must make sure that the preparation process of these artifacts meets the safety requirements according to the SIL level of the functions configured by the BuildingBlockConfiguration. To protect these safety-related configuration payloads, a second, cryptographically protected document is required: configurationSafe.json: an encrypted and signed companion to the configuration.json that ensures integrity, authenticity, and confidentiality of safety-relevant content. This file is signed using a Manufacturer Config Signer Certificate (MCSC) and, where necessary, encrypted for access control. [SPT2TS-130399]

The controlled distribution of approved configuration packages is managed through the distribution-job.json document:

This file is authored by the operator and describes when and to which targets a specific top-level BuildingBlockConfiguration (including all its dependencies) is to be deployed. It includes:

- Time windows for preload and activation
- Required activation conditions

- Target systems (e.g., vehicles, interlockings)
- Repository locations for synchronized access

The distribution job links to the top-level BuildingBlockConfiguration in the dependency tree and serves as the authoritative instruction for orchestrated rollout of configurations in operational environments.

The top-level BuildingBlock Configuration and its dependency tree form a Release.
[SPT2TS-130396]

The following chapters provide detailed descriptions and data schemas for all configuration, safety, and distribution-related documents defined in this specification.
[SPT2TS-130400]

3.1.1 Configuration Metadata: configuration.json

3.1.1.1 Core Concept Requirements

The configuration.json file is the core artifact used to describe a versioned BuildingBlockConfiguration (BBC). It contains an exhaustive, unambiguous, human- and machine-readable definition of a configuration item that has been approved by a responsible entity (either a supplier or integrator). It is written in a standardized, lightweight JSON format for interoperability across tooling and systems.
[SPT2TS-130401]

Each configuration.json document describes the WHAT of a BuildingBlockConfiguration:

- What type of logical or physical component is being configured
- What version of the configuration is applied
- What dependencies must be fulfilled for the item to be valid and operable
- What associated binary payload (e.g. firmware, parameter set) is part of the configuration

The file includes metadata, safety classification, dependency links, and optionally a reference to a payload file (configurationFile). It also contains repository metadata used for resolution during composition and distribution.

[SPT2TS-129691]

A configuration.json must not reference hardware-specific attributes such as serial numbers. Instead, it must describe reusable configurations bound to:

- A logical identifier (e.g., subsystem_identification)
- A hardware model or product line

This abstraction allows for:

- Reusability across fleet elements or identical hardware units
- Rapid replacement of failed components without re-signing entire configuration trees
- Simplified validation and shorter integration cycles

Reason: if configurations would be hardware specific (linked to the specific serial number) integrators have to prepare an individual config for each vehicle or create a new one in case of a component failure where a component needs to be replaced. This would cause long downtimes, because all upper integrator levels have to check and sign.

[SPT2TS-130402]

The configuration.json supports a recursive dependency mechanism. Each configuration may reference other configurations through a dependencies array. These references are identified by:

- bbId: Identifies the BuildingBlock (e.g., country.area.type.tag.number)
- bbcId: Identifies the configuration aspect (e.g., software, firmware, parameters)
- bbcVersion: Semantic version (e.g., 1.0.3)

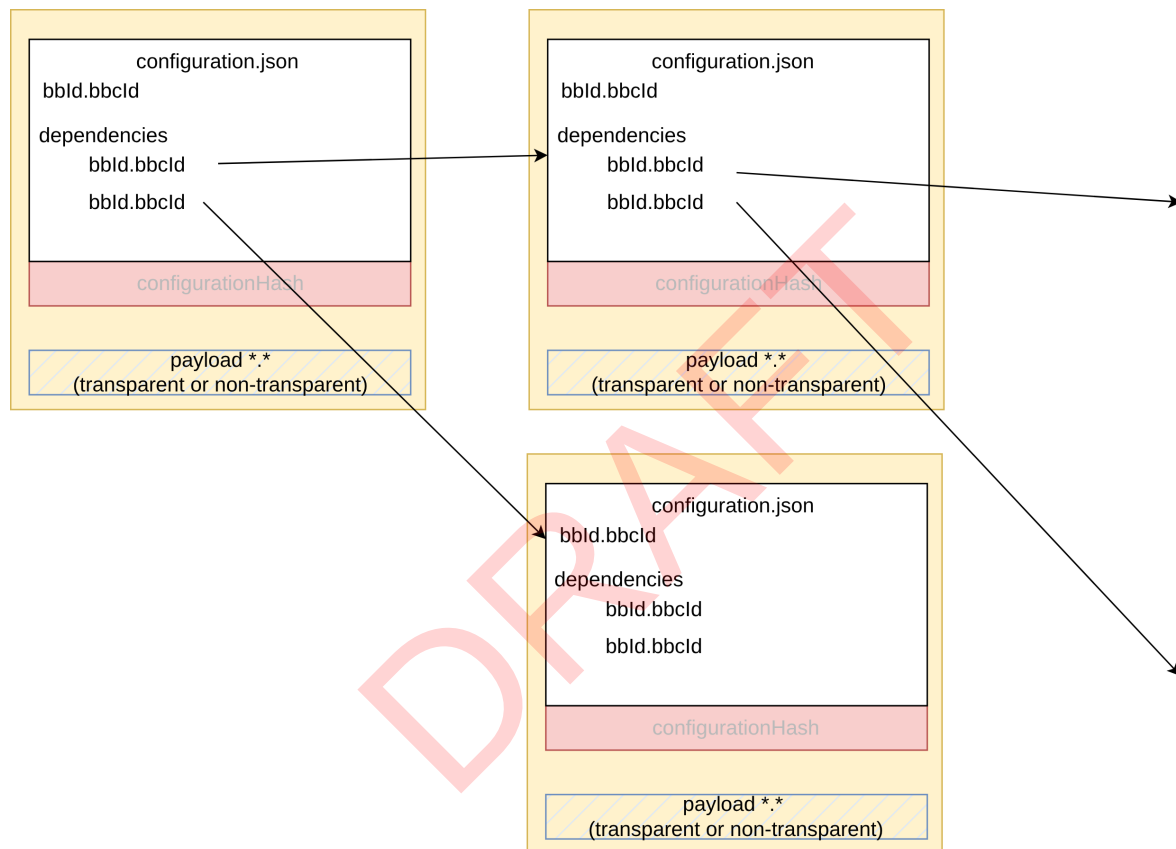
This builds a tree of dependencies, starting from a Top-Level BBC, down to the most atomic configurations.

- lower level configuration.json often reference core software/firmware payloads.
- Integrators build upon these by referencing dependencies and adding their own configuration layers, e.g. parameters.
- Multiple levels of integrators may exist, forming a structured chain of responsibility.

[SPT2TS-129693]

The bbld and bbcld fields within the configuration.json must exactly match the identifiers of the BuildingBlockConfiguration as referenced by the Configurable BuildingBlock. [SPT2TS-130470]

The number of dependency levels is not limited.



This dependency tree structure allows to build greater entities. [SPT2TS-130403]

The Top-Level BuildingBlockConfiguration (Release)

The top-level BuildingBlockConfiguration (BBC) represents the entry point of the configuration management process and is explicitly referenced by a distribution job. Each top-level BBC is uniquely identified through three attributes bbld, bbcld and bbcVersion (see above).

The top-level BBC explicitly references a dependency tree and forms a "release," clearly and unambiguously specifying the exact configuration state intended for deployment.

[SPT2TS-130469]

3.1.1.2 Signing Specification Requirements

This document defines the requirements and process for digitally signing configuration documents (configuration.json) according to the SMI v3 schema and integrity envelope structure. It ensures the authenticity, integrity, and traceability of configuration data across suppliers and integrators involved in the configuration and deployment of railway service functions. [SPT2TS-130406]

3.1.1.2.1 Scope

This specification applies to all configuration data packages that use the standardized BuildingBlockConfiguration (configuration.json) document structure. The signing process covers two parties:

- Suppliers authoring configuration data for lower-level components (e.g. firmware, parameters)
- Integrators composing and assembling higher-level configuration.

[SPT2TS-130407]

Document Structure and Signing Responsibilities

Item 1: Configuration Document (configuration.json)

Author:

- Supplier (for lower levels) or Integrator (for higher levels)

Purpose:

- Defines the configuration metadata, version, dependencies, and optionally links to a configuration payload (e.g., binary firmware or parameter zip).

Requirements:

- Lower levels typically include a configurationFile with integrity hash and digital signature (MCSC).
- Higher levels authored by integrators may consist only of dependencies and repository references.
- bbld, bbcl, and bbcVersion must uniquely identify the configuration.
- Format: Canonical JSON (RFC 8785 compliant)

[SPT2TS-130411]

Item 2: Item 1 Signature Envelope

Author:

- Supplier (for lower levels) or Integrator (for higher levels)

Purpose:

- Attests to the authenticity and integrity of the configuration.json by creating a cryptographic hash and signing it using a JWT (JWS compact format).

Required Fields:

- @schemaLocation: <http://schemas.rail-research.europa.eu/smi/v3/configuration-supplier-hash/v0.1>
- configurationHash: Canonical JSON hash of Item 1
- configurationHashEncryptedSigned: JWT signed using the supplier's Manufacturer Config Signer Certificate (MCSC), algorithm ES512
- signatureMeta (contains kid, alg, x5u) and issuedBy

Validation:

- The endpoint shall verify the signature of the configuration document using the corresponding installed roots of trust before installation.
- Signature must match the canonical hash of Item 1

[SPT2TS-130409]

Item 3: Integrator Host Extension (Optional)

Author:

- Integrator (only if a configurationFileTransferHost needs to be added)

Purpose:

- In cases where the supplier cannot define the file transfer host, the integrator extends the signed configuration by referencing the same configurationHash from Item 2 and adding a transfer host block. The new combination is signed and sealed in a separate JWT.

Required Fields:

- @schemaLocation: <http://schemas.rail-research.europa.eu/smi/v3/configuration-integrator-hash-extension/v0.1>
- configurationHash: Must match Item 2
- configurationFileTransferHost: (if applicable)
- configurationFileTransferHostHash: Canonical hash of the transfer host block
- configurationHashEncryptedSigned: JWT signed using the Operator Config Signer Certificate (OCSC), algorithm ES512
- signatureMeta, issuedBy

Validation:

- JWT payload must match hashes of Item 1 and host block
- Signature must verify against the integrator's Configuration Signer Certificate, also published via the Shared Security Services PKI following this process:
 - start with the trusted root CA certificate (to establish trust).
 - then use it to validate the integrator's signing certificate.
 - finally, we take the public key from the integrators's Configuration Signer Certificate to verify the signature.

[SPT2TS-130412]**Signing Algorithm Requirements**

- All JWTs must use: alg = ES512 (as defined in RFC 7518 section 3.1)
- Canonicalization must follow RFC 8785
- JWT header must include kid, and may include x5

[SPT2TS-130413]**3.1.1.3 Summary**

The configuration model supports a multi-tiered responsibility structure across suppliers and integrators.

- A supplier is end-to-end responsible for the correctness of the configuration.json document they author. These documents are typically located at the lower levels of the dependency tree and often include a reference to a binary configurationFile (e.g., firmware or safety-critical logic). The supplier also generates the cryptographic hash and digitally signs this configuration using their Manufacturer Config Signer Certificate (MCSC).
- An integrator is responsible for authoring configuration.json documents at higher levels, which typically include dependencies referencing BuildingBlockConfigurations from suppliers or other integrators. These documents may or may not include a configurationFile, depending on the configuration context (e.g., logical parameter sets). An integrator's document is easily identified by the presence of a dependencies array.
- Integrators may exist at multiple levels. Each integrator is independently and fully responsible for the correctness of the configuration composition they define—including the validation of all referenced dependencies. In cases where a payload exists but the supplier cannot provide the deployment context, the integrator may extend the configuration with a transfer host, and sign this information.

This layered structure ensures traceable, role-specific responsibility throughout the configuration lifecycle in data preparation allows each role to independently sign and validate the parts of the configuration under

their control.
[SPT2TS-130414]

3.1.1.4 Schema document

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Configuration Schema for Service Function Configuration according to SMI v3",
  "description": "Defines the structure of a BuildingBlockConfiguration in the context of a dependency tree.",
  "x-canonical-json": true,
  "version": "1.0.0",
  "author": "European Rail Research",
  "maintainer": "karl.klinge@praedicta.de",
  "created": "2025-03-23",
  "contributors": [
    "Prof. Dr. Karl-Albrecht Klinge"
  ],
  "type": "array",
  "minItems": 2,
  "maxItems": 3,
  "items": [
    {
      "description": "Configuration document.",
      "type": "object",
      "properties": {
        "@schemaLocation": {
          "type": "string",
          "const": "http://schemas.rail-research.europa.eu/smi/v3/configuration/v0.1",
          "description": "Fixed URI identifying the schema version of the configuration document."
        },
        "modelVersion": {
          "type": "string",
          "description": "Version of the underlying configuration model or schema."
        },
        "bbId": {
          "type": "string",
          "pattern": "^[a-zA-Z0-9]+(\\.[a-zA-Z0-9]+)*$",
          "description": "Unique identifier of the BuildingBlock. Assigned by the integrator. Must follow a dot-separated hierarchy (e.g., 'INT001.IO.BB123') and be valid for use in REST paths and repository directory structures."
        },
        "bbcId": {
          "type": "string",
          "pattern": "^[a-zA-Z0-9]+(\\.[a-zA-Z0-9]+)*$",
          "description": "Identifier of a specific BuildingBlockConfiguration (e.g., firmware, logical parameters). Assigned by the supplier on lower levels of the dependency tree. Must follow a dot-separated hierarchy and be unique within its associated bbId."
        },
        "bbcVersion": {
          "type": "string",
          "pattern": "^[\\d+\\.\\d+\\.\\d+]+$",
          "description": "Semantic version of the BuildingBlockConfiguration instance identified by the combination of bbId and bbcId. Must follow Semantic Versioning 1.0.0 format: major.minor.patch (e.g., '1.2.3')."
        },
        "configurationSafetyRelevance": {
          "type": "string",
          "enum": [
            "SAFE"
          ]
        }
      }
    }
  ]
}
```

```

"NONSAFE"
],
"description": "Indicates whether the configuration has safety relevance in operational deployment."
},
"configurationFile": {
"type": "object",
"description": "Contains metadata and integrity information for an optional configuration file associated with this configuration item.",
"properties": {
"configurationFileName": {
"type": "string",
"description": "The name of the optional configuration file located in the same directory or URL path as the 'configuration.json'. Example: 'parametrization.zip'."
},
"configurationFileHash": {
"type": "string",
"description": "Fingerprint (hash) of the optional configuration file. It must include the logical ID (e.g., subsystemId) and the hardware model, but must not include identifiers of specific hardware units (e.g., serial numbers). This ensures support for model-level reuse and interchangeability."
},
"configurationFileSignatureAlgorithm": {
"type": "string",
"enum": [
"ecdsa-with-sha512"
],
"description": "The cryptographic signature algorithm used to sign the configuration file. Must be 'ecdsa-with-sha512', complying with MCSC specification (see SPPRAMSS-7356)."
},
"configurationFileSignature": {
"type": "string",
"description": "Base64-encoded digital signature of the configuration file, created using the MCSC-compliant private key corresponding to the declared signature algorithm."
},
"configurationFileTransferHostPath": {
"type": "string",
"pattern": "^\\([a-zA-Z0-9_\\.]+\\W\\)*[a-zA-Z0-9_\\.]+$",
"description": "Path on the transfer host to the configuration directory or file. Must start with a slash and use UNIX-style path segments. Example: '/configs/vehicle-model-a/param-v1.0.0.zip'"
},
"required": [
"configurationFileName",
"configurationFileHash",
"configurationFileSignatureAlgorithm",
"configurationFileSignature",
"configurationFileTransferHost",
"configurationFileTransferHostPath"
]
},
"name": {
"type": "string",
"description": "Provider-specific name for the configuration item. This name is typically used for human-readable identification within the provider's tools or repositories."
},
"url": {
"type": "string",
"format": "uri",
"description": "Provider-specific URL linking to additional content or documentation related to this configuration item. May point to a repository, API, or resource index."
},
"dependencies": {

```

```

"type": "array",
"description": "List of direct dependencies required for this BuildingBlockConfiguration to operate correctly. Dependencies must be resolved recursively from bottom to top.",
"items": {
  "type": "object",
  "description": "A single dependency pointing to another BuildingBlockConfiguration, identified by its bbld, bbcld, and bbcVersion.",
  "properties": {
    "bbld": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9]+)*$",
      "description": "Unique identifier of the dependent BuildingBlock. Assigned by the integrator. Follows a dot-separated hierarchy (e.g., 'INT001.IO.BB123')."
    },
    "bbcld": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9]+)*$",
      "description": "Identifier of the dependent BuildingBlockConfiguration. Assigned by the supplier. Describes a specific configuration part such as firmware or logical parametrization."
    },
    "bbcVersion": {
      "type": "string",
      "pattern": "^[\\d+\\.\\d+\\.\\d+]+$",
      "description": "Semantic version of the dependent BuildingBlockConfiguration (e.g., '1.2.3'). Follows Semantic Versioning 1.0.0 (major.minor.patch)."
    },
    "configurationSafetyRelevance": {
      "type": "string",
      "enum": [
        "SAFE",
        "NONSAFE"
      ],
      "description": "Indicates whether the dependent configuration is safety-relevant. Affects handling and validation during deployment."
    }
  },
  "required": [
    "bbld",
    "bbcld",
    "bbcVersion",
    "configurationSafetyRelevance"
  ]
},
"repositories": {
  "type": "array",
  "description": "List of repository definitions used to locate and resolve BuildingBlockConfiguration artifacts. Repositories follow a hierarchical resolution strategy similar to Maven.",
  "items": {
    "type": "object",
    "description": "A repository entry defining the source location and order for resolving configuration components.",
    "properties": {
      "name": {
        "type": "string",
        "pattern": "^[a-z0-9]+([-_\\.][a-z0-9]+)*$",
        "description": "Unique identifier for the repository (e.g., 'main-repo', 'vendor.repo-1'). Must consist of lowercase letters, numbers, dots, hyphens, or underscores. Used in resolution logs and tooling."
      },
      "url": {
        "type": "string",

```



```

"format": "uri",
"pattern": "^https://.*",
"description": "Base URI of the repository. Must begin with 'https://' for secure resolution. Used to locate
configuration artifacts using bbld/bbcld/bbcVersion path structure."
},
"sequenceNr": {
"type": "integer",
"description": "Optional numeric priority indicator. Lower values are checked first during dependency
resolution (similar to Maven's repository order)."
},
"required": [
"name",
"url"
]
},
"required": [
"@schemaLocation",
"modelVersion",
"bbld",
"bbcld",
"bbcVersion",
"configurationSafetyRelevance",
"name",
"url",
"repositories"
]
},
{
"description": "This object contains the final signed integrity envelope created by the author of the first item
(on lowest levels of the dependency tree the supplier, higher levels: integrators). It confirms the
authenticity and integrity of the configuration document.",
"type": "object",
"properties": {
"@schemaLocation": {
"type": "string",
"const": "http://schemas.rail-research.europa.eu/smi/v3/configuration-supplier-hash/v0.1",
"description": "URI identifying the envelope containing the supplier's signed hash of the configuration
document."
},
"configurationHash": {
"type": "string",
"description": "Hash value of the configuration document (item 1), computed using canonical JSON
serialization (e.g., RFC 8785)."
},
"configurationHashEncryptedSigned": {
"type": "string",
"description": "A compact JWS (JWT) object with a signed payload containing the configurationHash.
Signed using the Manufacturer Config Signer Certificate (MCSC) with ES512."
},
"signatureMeta": {
"type": "object",
"description": "Metadata extracted from the JWT header to assist in validation and traceability.",
"properties": {
"alg": {
"type": "string",
"enum": [
"ES512"
]
}
}
}
}

```

```

"description": "The signing algorithm used. Must be ES512."
},
"kid": {
  "type": "string",
  "description": "Key ID to identify the Configuration Signer Certificate used for signature verification."
},
"x5u": {
  "type": "string",
  "format": "uri",
  "description": "Optional URI pointing to the Configuration Signer Certificate or certificate used for verification."
},
},
},
"issuedBy": {
  "type": "string",
  "description": "Logical name or ID of the entity (e.g., supplier) that issued the signature."
},
},
"required": [
  "@schemaLocation",
  "configurationHash",
  "configurationHashEncryptedSigned",
  "signatureMeta",
  "issuedBy"
],
},
{
  "description": "This object is optionally added by the integrator to specify a host for payload delivery and confirm integrity of both the configuration and the host location.",
  "type": "object",
  "properties": {
    "@schemaLocation": {
      "type": "string",
      "const": "http://schemas.rail-research.europa.eu/smi/v3/configuration-integrator-hash-extension/v0.1",
      "description": "URI identifying the integrator's signed envelope extending the supplier configuration with host delivery information."
    },
    "configurationHash": {
      "type": "string",
      "description": "Hash of the configuration document (identical to the hash in the previous supplier-signed envelope)."
    },
    "configurationFileTransferHost": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9.-]+$",
      "description": "Host or domain name for file transfer. Optional and only required when a payload file exists."
    },
    "configurationFileTransferHostHash": {
      "type": "string",
      "description": "Canonical hash of the integrator-defined transfer host block. Used in the combined JWT."
    },
    "configurationHashEncryptedSigned": {
      "type": "string",
      "description": "A compact JWS (JWT format) signed by the integrators Config Signer Certificate, covering the configurationHash and configurationFileTransferHostHash (if present). Uses ES512."
    },
    "signatureMeta": {
      "type": "object",
      "description": "Metadata extracted from the JWT header to assist in validation and traceability.",

```

```

"properties": {
  "alg": {
    "type": "string",
    "enum": [
      "ES512"
    ],
    "description": "The signing algorithm used. Must be ES512."
  },
  "kid": {
    "type": "string",
    "description": "Key ID to identify the Configuration Signer Certificate used for signature verification."
  },
  "x5u": {
    "type": "string",
    "format": "uri",
    "description": "Optional URI pointing to the Configuration Signer Certificate or certificate used for verification."
  }
},
"issuedBy": {
  "type": "string",
  "description": "Logical identifier of the integrator signing this envelope."
},
"required": [
  "@schemaLocation",
  "configurationHash",
  "configurationHashEncryptedSigned",
  "signatureMeta",
  "issuedBy"
]

```

[SPT2TS-130417]

3.1.2 Configuration Metadata: configurationSafe.json

3.1.2.1 Core Concept Requirements

The configurationSafe.json document is a mandatory companion to the standard configuration.json for any safety-relevant BuildingBlockConfiguration (BBC). It provides integrity- and operation-relevant metadata in a protected and attestable format. This ensures that a configuration deployed in a railway system meets both the safety requirements and the responsibility separation required across different levels of integration.

[SPT2TS-129704]

Safety-Scoped Configuration Management

- A BBC marked with "configurationSafetyRelevance": "SAFE" must have a corresponding configurationSafe.json.
- The configurationSafe.json is used only during validation and attestation by the Safe Configuration Authority (SCA).
- It is never deployed to the BuildingBlock, only its hash is used to verify dependency integrity in the Service Function Configuration.

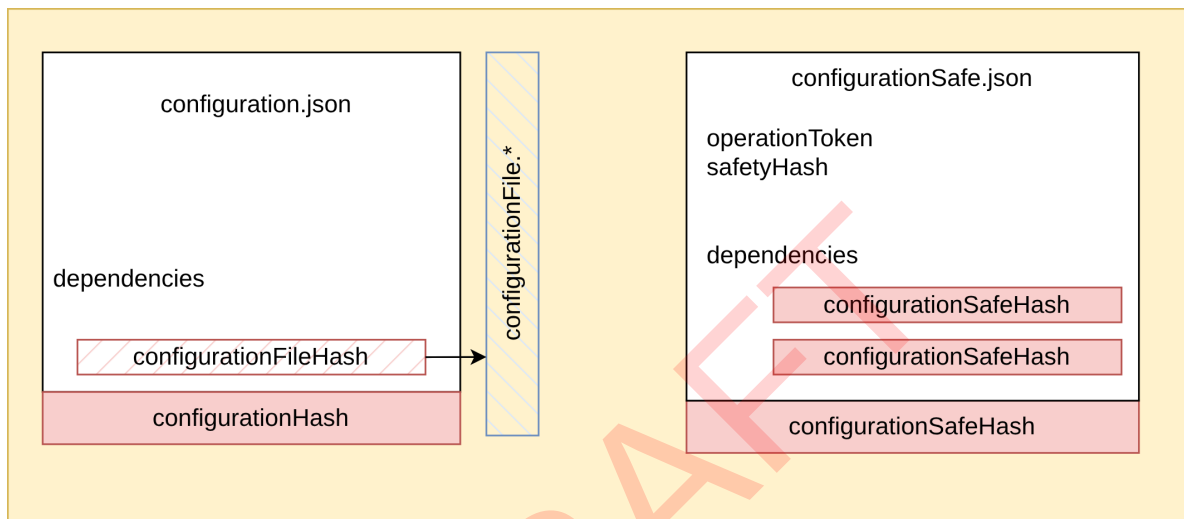
[SPT2TS-130426]

Integrity Enforcement via Hash Chaining

- The document includes a configurationSafeHash, which is a deterministic hash over the full configurationSafe content (canonical JSON, e.g., RFC 8785).
- This hash is signed using the Manufacturer Config Signer Certificate (MCSC) in the form of a JWT (JWS Compact Serialization).
- On higher integration levels, this configurationSafeHash is added to the dependencies[] list of the integrator's own configurationSafe.json.
- This chaining ensures that no change can be made to lower-level configurations without triggering re-signing and approval on all higher levels.

[SPT2TS-130423]

The "configurationSafe.json" is placed next to the "configuration.json" document.

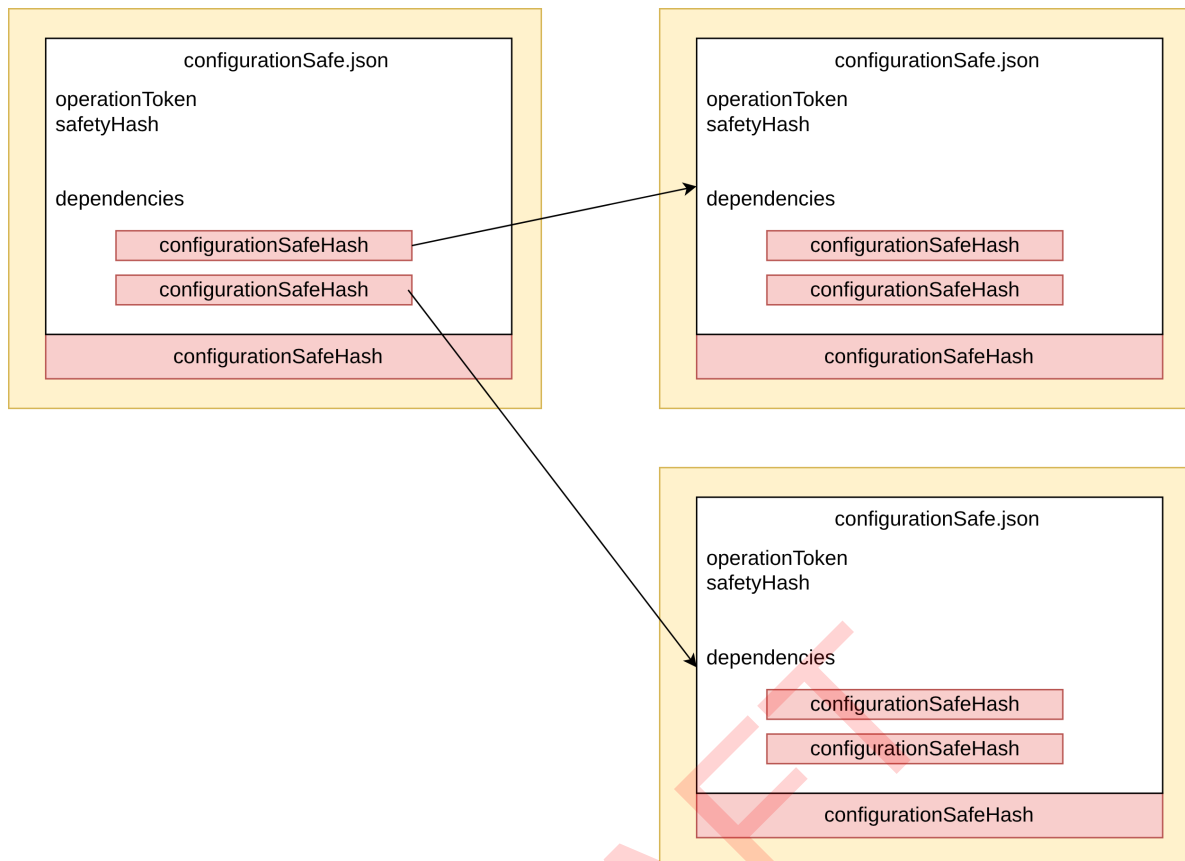


[SPT2TS-130425]

The hash of the "configurationSafe.json" of the dependency is contained within the configurationSafeHash.

For the "configurationSafe.json" this attribute must be added to the dependency on a higher level of the dependency tree to ensure the integrity of the safe dependency tree structure, because it is not possible to change any value in the dependency tree below without also updating the configurationSafeHash and signing on all upper integration levels.

The arrows in the following picture show which items must be included in the calculation of the configurationSafeHash fingerprints to ensure the integrity of these entities.



This recursive (re-)calculation of the configurationSafeHash is a burden to the process that is needed for the safe configuration, because it ensures that all integrators on higher levels check and sign the integration.

As a result the integrators on each of the higher levels of the "safe" dependency tree have to update and approve their "configurationSafe.json" files and the corresponding configurationSafeHash values. This is needed for the safe contents and imposes a burden on the data preparation process. But it is the only way to express the responsibility of each integrator for the correctness of the composition of the included dependencies.

The "configurationSafe.json" documents also include the operationToken and the safetyHash that are needed during safety attestation (see below).
[SPT2TS-130421]

3.1.2.2 Signing Process Requirements

Item 1: Safety-Relevant Configuration Metadata

Author:

- Supplier (for lower levels) or Integrator (for composite configurations on higher levels)

Purpose:

- Provide a complete, safety-scoped configuration that includes operational tokens, integrity data, and dependency hashes of all safety-relevant BuildingBlockConfigurations for the next lower dependency level.

Requirements:

- Must follow the schema <http://schemas.rail-research.europa.eu/smi/v3/configurationSafe/V0.1>
- Must include all mandatory fields: bbld, bbcld, bbcVersion, configurationSafetyRelevance, safetyHash, operationToken, and repository/dependency references
- Dependency tree must be fully expanded with all configurationSafeHash entries included at this level
- Must use consistent canonical JSON serialization (RFC 8785) to ensure hash determinism.

[SPT2TS-130418]

Item 2: Signed Integrity Envelope

Author:

- Supplier or Integrator depending on who authored Item 1

Purpose:

- Bind the configuration metadata (Item 1) to a secure, verifiable hash and ensure its authenticity using cryptographic signing

Requirements:

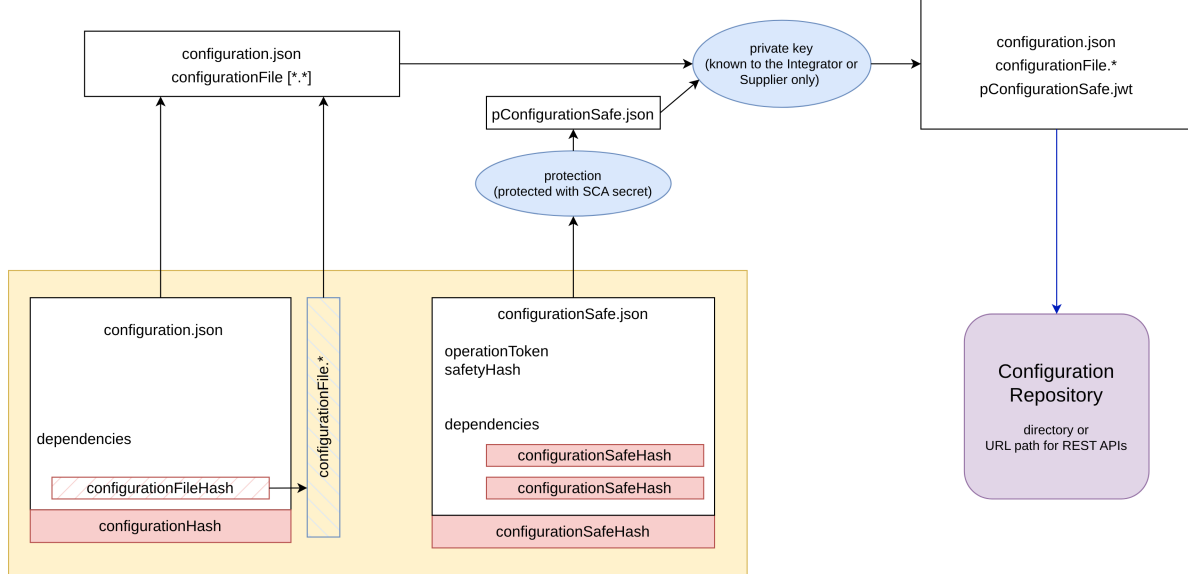
- Hash Item 1 using SHA-512 over a canonicalized JSON structure (RFC 8785)
- Sign the resulting hash using the Manufacturer Config Signer Certificate (MCSC)
- Encode the signature as a JWT (JWS Compact Serialization)
 - JWT Header must include: alg: ES512, typ: JWT, kid: identifier of signing key
 - JWT Payload must include: configurationSafeHash and optional metadata like issuedBy, iat
 - JWT Signature must cover the Header and Payload
- Store the signed JWT in configurationSafeHashEncryptedSigned
- Include the raw hash value in configurationSafeHash to allow quick verification before signature decoding
- Ensure that the certificate containing the public key used for validation is included in the signature package and chains up to a trusted root CA from the Shared Security Services PKI. The root CA certificate must be known and trusted by every Building Block that validates the signature.

Validation requirements:

- Verify signature using the public key identified by kid
- Ensure hash inside the JWT payload matches recomputed value from Item 1
- Optionally validate standard JWT claims (iss, iat, exp) if present

[SPT2TS-130430]

The following picture represents this for a repository of one BuildingBlockConfiguration.



[SPT2TS-129700]

3.1.2.3 Schema Document

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://schemas.rail-research.europa.eu/smi/v3/configurationSafeSchema-v0.1.json",
  "title": "Configuration Safe Document",
  "description": "Defines the structure of the safety-related configuration extensions to the configuration that are protected and available to the Safe Configuration Authority (SCA) only.",
  "x-canonical-json": true,
  "author": "European Rail Research",
  "maintainer": "karl.klinge@praedicta.de",
  "created": "2025-03-23",
  "contributors": [
    "Prof. Dr. Karl-Albrecht Klinge"
  ],
  "type": "array",
  "minItems": 2,
  "maxItems": 2,
  "items": [
    {
      "description": "This object represents the primary configuration details including security and operational parameters for a specific firmware of a product.",
      "type": "object",
      "properties": {
        "@schemaLocation": {
          "type": "string",
          "const": "http://schemas.rail-research.europa.eu/smi/v3/configurationSafe/V0.1",
          "description": "Fixed URI identifying the schema version of the configuration document."
        },
        "modelVersion": {
          "type": "string",
          "description": "Version of the underlying configuration model or schema."
        },
        "bbid": {
          "type": "string",

```

```

"pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9])*$",
"description": "Unique identifier of the BuildingBlock, assigned by the integrator. Must follow a dot-separated hierarchy and be globally unique."
},
"bbcid": {
"type": "string",
"pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9])*$",
"description": "Identifier for a specific BuildingBlockConfiguration, such as firmware or parameter set. Assigned by the supplier."
},
"bbcVersion": {
"type": "string",
"pattern": "^\\d+\\.\\d+\\.\\d+$",
"description": "Version of the BuildingBlockConfiguration. Must follow Semantic Versioning (major.minor.patch)."
},
"configurationSafetyRelevance": {
"type": "string",
"enum": [
"SAFE",
"NONSAFE"
],
"description": "Indicates whether the configuration has safety relevance in operational deployment."
},
"safetyHash": {
"type": "string",
"description": "Cryptographic hash for integrity validation of the safety-relevant configuration content. The safetyHash is calculated during data preparation and is used for safety attestation."
},
"operationToken": {
"type": "string",
"description": "Token required for activating or operating this configuration. Provided by the supplier or integrator to allow the operation of the BBC."
},
"name": {
"type": "string",
"description": "Provider-specific item name used for identification and repository indexing."
},
"url": {
"type": "string",
"format": "uri",
"description": "Provider-specific URL to linked content, documentation, or download endpoint."
},
"dependencies": {
"type": "array",
"description": "List of required BuildingBlockConfigurations that must be resolved and installed before this one. Recursively resolved.",
"items": {
"type": "object",
"description": "A dependency referencing another BuildingBlockConfiguration document.",
"properties": {
"bbid": {
"type": "string",
"pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9])*$",
"description": "Identifier of the dependent BuildingBlock."
},
"bbcid": {
"type": "string",
"pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9])*$",
"description": "Identifier of the configuration element of the dependent BuildingBlock (e.g., firmware or parameter set)."
}
}
}
}

```



```

},
"bbcVersion": {
  "type": "string",
  "pattern": "^\\d+\\.\\d+\\.\\d+$",
  "description": "Version of the dependent BuildingBlockConfiguration (Semantic Versioning format)."
},
"configurationSafeHash": {
  "type": "string",
  "description": "Hash of the dependent BuildingBlockConfiguration used for integrity validation and trust anchoring."
},
"configurationSafetyRelevance": {
  "type": "string",
  "enum": [
    "SAFE",
    "NONSAFE"
  ],
  "description": "Whether the dependency is safety-relevant and thus must be handled with stricter guarantees."
},
"required": [
  "bbld",
  "bbcld",
  "bbcVersion",
  "configurationSafeHash",
  "configurationSafetyRelevance"
]
},
"repositories": {
  "type": "array",
  "description": "List of repositories where BuildingBlockConfiguration artifacts are located, resolved in order of sequenceNr.",
  "items": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string",
        "pattern": "^[a-z0-9]+([-_\\.][a-z0-9]+)*$",
        "description": "Logical name for the repository (e.g., 'main-repo', 'vendor.repo-1')."
      },
      "url": {
        "type": "string",
        "format": "uri",
        "pattern": "^https://.*",
        "description": "HTTPS base URL of the repository root."
      },
      "sequenceNr": {
        "type": "integer",
        "description": "Optional priority for repository resolution. Lower numbers have higher precedence."
      }
    }
  },
  "required": [
    "name",
    "url"
  ]
},
"required": [

```

```

"@schemaLocation",
"modelVersion",
"bbld",
"bbcld",
"bbcVersion",
"configurationSafetyRelevance",
"safetyHash",
"operationToken",
"name",
"url",
"repositories"
],
},
{
  "description": "This object contains the supplier-signed hash of the primary configuration object (first object in the configurationSafe document). It provides verifiable integrity and authenticity using a cryptographically signed JWT, created with the Manufacturer Config Signer Certificate (MCSC).",
  "type": "object",
  "properties": {
    "@schemaLocation": {
      "type": "string",
      "const": "http://schemas.rail-research.europa.eu/smi/v3/configurationSafeHash/V0.1",
      "description": "Fixed schema URI for configuration hash validation objects. Identifies this section as the hash and signature envelope for the supplier-signed configuration."
    },
    "configurationSafeHash": {
      "type": "string",
      "description": "Top-level integrity hash of the supplier's configuration document (first object in the array). This hash must be calculated using a deterministic, canonical JSON serialization (e.g., RFC 8785 / JSON Canonicalization Scheme). The result is typically encoded in hexadecimal or base64 format."
    },
    "configurationSafeHashEncryptedSigned": {
      "type": "string",
      "description": "A compact JSON Web Token (JWT) signed using the Manufacturer Config Signer Certificate (MCSC). It includes the configurationSafeHash in its payload and is digitally signed using the ES512 algorithm (ECDSA with SHA-512).\n\nThe JWT consists of three base64url-encoded parts:\n1. **Header**: includes algorithm (ES512), token type (JWT), and key ID (kid)\n2. **Payload**: contains the configurationSafeHash and optional metadata (e.g., issuedBy, iat)\n3. **Signature**: digital signature over the header and payload\n\nThis field enables cryptographic validation of the integrity and authenticity of the configuration block, without requiring access to the private key.\n\nTools verifying this section should:\n- Recalculate the hash over the first configuration object\n- Extract and decode the JWT\n- Verify the payload includes the expected hash\n- Verify the signature using the MCSC public key"
    },
    "signatureMeta": {
      "type": "object",
      "description": "Metadata extracted from the JWT header to assist in validation and traceability.",
      "properties": {
        "alg": {
          "type": "string",
          "enum": [
            "ES512"
          ],
          "description": "The signing algorithm used. Must be ES512."
        },
        "kid": {
          "type": "string",
          "description": "Key ID to identify the public key used for signature verification."
        },
        "x5u": {
          "type": "string",
          "format": "uri",

```

```
{
  "description": "Optional URI pointing to the public key or certificate used for verification."
},
{
  "issuedBy": {
    "type": "string",
    "description": "Logical identifier of the integrator signing this envelope."
  }
},
{
  "required": [
    "@schemaLocation",
    "configurationSafeHash",
    "configurationSafeHashEncryptedSigned",
    "signatureMeta",
    "issuedBy"
  ]
}
]
}
```

[SPT2TS-130432]

3.1.3 distribution-Job

3.1.3.1 Core Concept requirements

The distribution-job.json file describes the conditions of **WHEN** to distribute BuildingBlockConfiguration items in json-syntax.

[SPT2TS-129706]

The distribution.json document is authored by integrators or operators responsible for a homologated fleet (e.g., vehicles with the same configuration baseline) or a specific infrastructure element (e.g., interlocking system). While integrators ensure the technical correctness and compatibility of the included configurations, it is the operator who holds the critical knowledge about when and where the BuildingBlocks to be configured are in use. In most cases, a BuildingBlock cannot be actively used during a configuration update. The operator must therefore ensure that the update process aligns with the operational plan, taking into account availability windows, maintenance schedules, and system states. This ensures that the update does not interfere with ongoing operations. As such, the operator is responsible for verifying that the timing and targets of the distribution are compatible with current operational constraints.

[SPT2TS-130435]

The distribution-job references a top-level BuildingBlockConfiguration to be distributed (Release). That includes the dependencies recursively.

[SPT2TS-130434]

The `distribution.json` file must be placed inside a `/distribution` subdirectory under the repository path of the top-level BBC it distributes. This ensures traceability and a clear relationship between the distribution and the configuration it applies to.

[SPT2TS-130433]

The operator is responsible, that the distribution-job is in line with the operational requirements (see above).

The activation may need an additional condition to be met within the activation time window to cover different operational scenarios, e.g.:

- a vehicle is running late and cannot start the activation before arriving in the target station
- A vehicle provided for activation might be needed on short notice
- A train is running late and the interlocking cannot update

The DistributionJob can contain an additional condition that must be met to start the activation: a time window, driver confirmation, GOA4 confirmation. This could be allocated to the DMI and could be specified

further in TrainCS.
[SPT2TS-129709]

The following table provides an explanation of the attributes of the distribution-job.json documents:

| Element | Description |
|---|---|
| distributionGroupId | <p>The distributionGroupId must be generally unique amongst different organizations and models in case of more than one configuration is distributed in the DistributionJob.</p> <p>Examples: com.integrator-a.vehicle-model-c com.operator-b.area-1</p> <p>All configurations distributed within one distribution-job are the same. That means that all configurations must be for the same overall model. So the dependency tree contains the same composition of models that need the identical firmware and parametrization.</p> <p>That would be true for ONE interlocking configuration, because interlockings are most often unique. It would also be true for all vehicles complying to one overall MODEL within a fleet of vehicles. In case of a retrofit a new homologation must be created and a new model must be created.</p> |
| distributionIds | <p>This array contains distributionId that each must be unique together with the distributionGroupId and represent the target for the distribution.</p> <p>Each distributionId item additionally contains the information when (start and end for preloadTimeWindow and activationTimeWindow) to distribute. All times are given as UNIX Timestamp.</p> <p>A distributionId can contain a repository, if the configuration preloading and activation should be processed locally e.g. on a vehicle. This might be the case for vehicles that might be parked in an area of poor cell phone reception. In this case it could be useful to partition a repository on the train. A time window is specified when the synchronisation of that repository is made.</p> |
| @id | <p>A version might be transferred to multiple items that have the same homologation. The @id will make these distribution-Jobs unique. The expected start of the distribution-job can be used, e.g. "@id": "2024-03-31-01-00"</p> |
| name | Human readable name of the distribution. |
| bbId bbcId bbcVersion configurationSafetyRelevance | <p>The distribution.json links to the Top-Level BuildingBlockConfiguration identified by bbId.bbcId.bbcVersion.</p> <p>For more information refer to the description in the configuration document.</p> |

Table 1 BuildingBlockConfiguration distribution-job.json description

[SPT2TS-129713]

3.1.3.2 Signing Process requirements

Item 1 — Main Content

Author:

- Integrator or operator responsible for releasing a complete and tested set of configurations for a vehicle or interlocking model.

Purpose:

- To define the exact set of BuildingBlockConfigurations to be distributed and activated together, ensuring compatibility and compliance with operational and safety constraints.

Requirements:

- Must be serialized deterministically (canonical JSON, e.g., [RFC 8785]).
- Must include all required fields:
 - distributionGroupId, distributionId, distributionVersion
 - A reference to the top level dependency tree
- One or more repositories
- The object must be stored in a distribution-specific subfolder of the top-level BBC it distributes: /distribution/<@id>/distribution-job.json.
- The file must not be updated. If changes are required, a new version must be created.

[SPT2TS-130438]

Item 2 — Hash & JWT Envelope

Author:

- The same integrator or operator who authored the first item.

Purpose:

- To provide a verifiable proof of authenticity and integrity of the distribution definition, using cryptographic signatures compliant with public-key infrastructure (PKI) standards.

Requirements:

- The distributionHash must be calculated using canonical JSON serialization.
- A JWT (JWS compact form) must be created containing:
 - The computed hash as payload (distributionHash)
 - A header including:
 - alg = ES512
 - typ = JWT
 - kid (Key ID identifying the public key)
- If Operator signs: A digital signature created using the Operator Config Signer Certificate (OCSC) private key
- The JWT is stored in the field distributionHashEncryptedSigned.
- The signed metadata (header info) must be included as signatureMeta for traceability.
- The issuedBy field must clearly identify the signer (organization or tool).
- The JWT and its public key (or chain) must be verifiable by the Shared Security Services Certificate Authority (CA).

[SPT2TS-130439]

3.1.3.3 Schema Document

The distribution-job.json documents use the json syntax. [SPT2TS-130442]

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://schemas.rail-research.europa.eu/distribution/schema-v0.1.json",
  "title": "BuildingBlockConfiguration Distribution Definition",
  "description": "Defines a distribution.json document that lists validated, released, and integrator-ready combinations of BuildingBlockConfigurations for a specific model or deployment target.",
  "x-canonical-json": true,
  "author": "European Rail Research",
  "maintainer": "karl.klinge@praedicta.de",
  "created": "2025-03-23",
  "contributors": [
    "Prof. Dr. Karl-Albrecht Klinge"
  ],
  "type": "array",
  "minItems": 2,
  "maxItems": 2,
  "items": [
    {
      "description": "This object defines a set of tested and released BuildingBlockConfigurations that work together for a given vehicle model or infrastructure element. It is used by integrators to build valid configuration trees.",
      "type": "object",
      "properties": {
        "@schemaLocation": {
          "type": "string",
          "const": "http://schemas.rail-research.europa.eu/distribution/V0.1",
          "description": "Fixed schema URI identifying this as a distribution.json file."
        },
        "modelVersion": {
          "type": "string",
          "description": "Version of the distribution model used to describe this file structure."
        },
        "distributionGroupId": {
          "type": "string",
          "description": "Globally unique identifier for the group of configurations, usually scoped by integrator and model (e.g., 'com.integrator.vehicle-model-c')."
        },
        "distributionId": {
          "type": "string",
          "description": "Unique identifier of this specific distribution instance within the group. Represents one application context, such as a single vehicle or interlocking."
        },
        "distributionVersion": {
          "type": "string",
          "pattern": "^\\d+\\.\\d+\\.\\d+\\$",
          "description": "Version of the distribution. Must follow Semantic Versioning (major.minor.patch)."
        },
        "name": {
          "type": "string",
          "description": "Human-readable name of the distribution. Used for documentation and tooling."
        },
        "url": {
          "type": "string",
          "format": "uri",

```

```

"description": "Provider-specific URL pointing to further documentation or index of this distribution."
},
"dependency": {
  "type": "object",
  "description": "The single top-level BuildingBlockConfiguration to be distributed. This configuration has
been tested, released, and validated for the target deployment context.",
  "properties": {
    "bbld": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9]+)*$",
      "description": "BuildingBlock identifier, assigned by the integrator. Dot-separated for hierarchical structure."
    },
    "bbcld": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9](\\.[a-zA-Z0-9]+)*$",
      "description": "BuildingBlockConfiguration identifier, assigned by the supplier, describing firmware or
parametrization."
    },
    "bbcVersion": {
      "type": "string",
      "pattern": "^[\\d+\\.\\d+\\.\\d+]+$",
      "description": "Semantic version of the configuration (e.g., firmware or parameter set)."
    },
    "configurationSafetyRelevance": {
      "type": "string",
      "enum": [
        "SAFE",
        "NONSAFE"
      ],
      "description": "Indicates if this configuration item is safety-relevant and must be validated accordingly."
    }
  },
  "required": [
    "bbld",
    "bbcld",
    "bbcVersion",
    "configurationSafetyRelevance"
  ],
  "repositories": {
    "type": "array",
    "description": "List of repositories where the BuildingBlockConfigurations included in this distribution can
be resolved from.",
    "items": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "pattern": "^[a-z0-9]+([-\\.][a-z0-9]+)*$",
          "description": "Logical name for the repository (e.g., 'main-repo')."
        },
        "url": {
          "type": "string",
          "format": "uri",
          "pattern": "^https://.*",
          "description": "Secure URL (HTTPS) pointing to the root of the repository."
        },
        "sequenceNr": {
          "type": "integer",
          "description": "Optional priority for repository resolution. Lower values are preferred."
        }
      }
    }
  }
}

```

```

},
"required": [
  "name",
  "url"
]
}
},
"required": [
  "@schemaLocation",
  "modelVersion",
  "distributionGroupId",
  "distributionId",
  "distributionVersion",
  "name",
  "url",
  "dependencies",
  "repositories"
]
},
{
  "description": "This object contains the integrity and authenticity envelope for the distribution.json file. It includes a canonicalized hash over the main distribution document (first object in the array) and a digitally signed JWT using the Operator Config Signer Certificate (OCSC).",
  "type": "object",
  "properties": {
    "@schemaLocation": {
      "type": "string",
      "const": "http://schemas.rail-research.europa.eu/smi/v3/distributionHash/V0.1",
      "description": "Fixed schema URI identifying this object as the signed hash envelope for a distribution.json file."
    },
    "distributionHash": {
      "type": "string",
      "description": "Cryptographic hash of the distribution (the first object in the array). Must be calculated using canonical JSON serialization (e.g., RFC 8785 / JSON Canonicalization Scheme). This hash represents the authoritative fingerprint of the distribution document."
    },
    "distributionHashEncryptedSigned": {
      "type": "string",
      "description": "A signed JSON Web Token (JWT, compact JWS format) that includes the `distributionHash` in its payload. This JWT is produced by the operator or integrator using the Operator Config Signer Certificate (OCSC), and signed with the ES512 algorithm (ECDSA with SHA-512). The JWT has three parts, separated by periods: 1. **Header** – Declares algorithm (`ES512`), token type (`JWT`), and an optional key identifier (`kid`) 2. **Payload** – Contains the `distributionHash` and additional metadata such as `issuedBy` and `iat` (issued-at timestamp) 3. **Signature** – A digital signature over the Base64URL-encoded header and payload This envelope allows offline cryptographic verification of the distribution.json file's integrity and trustworthiness using the OCSC public key. Verification process: - Canonicalize and hash the distribution.json object - Decode the JWT and extract the payload - Confirm that the `distributionHash` matches the computed hash - Validate the signature using the trusted public key of the OCSC"
    },
    "signatureMeta": {
      "type": "object",
      "description": "Metadata extracted from the JWT header to assist in validation and traceability.",
      "properties": {
        "alg": {
          "type": "string",
          "enum": [
            "ES512"
          ],

```



```

    "description": "The signing algorithm used. Must be ES512."
  },
  "kid": {
    "type": "string",
    "description": "Key ID to identify the public key used for signature verification."
  },
  "x5u": {
    "type": "string",
    "format": "uri",
    "description": "Optional URI pointing to the public key or certificate used for verification."
  },
  },
  },
  "issuedBy": {
    "type": "string",
    "description": "Logical identifier of the integrator signing this envelope."
  },
  },
  "required": [
    "@schemaLocation",
    "distributionHash",
    "distributionHashEncryptedSigned",
    "signatureMeta",
    "issuedBy"
  ]
}

```

[SPT2TS-130443]

3.2 Interace description

Exchange Scenarios between Configuration Repository and IM, RU, and supplier.

3.3 Behaviour description

Note to author: This section is optional and shall

a description of the behavioural elements / dynamic description of subsystems related to an interface, how the interface reacts to stimulus, or what logical sequencing happens, etc.

Example sor network communications:

- Initialization
- Information flow
- errors (means of detection and any backups)
- low control (volume, frequency, limits)
- Sequencing of exchanges and timing (exchange protocol)

For physical interfaces

- it/signal conversion for an electronic interface (NRZ, Manchester...)
- pressure variations for a brake peak and transient behaviour

For logical interfaces (Capella logical component exchange) and functional interfaces (set of functional exchanges):

- sequences / scenarios

- exchanged data (messages) and their structure (technology-independent)
- ...

3.4 Interdependencies to other interface layers

Analysis of dependencies between levels like time-out values among OSI layers, disconnection detection and reconnection, ...

DK: Consideration of Service access points (vertical between OSI layers), APIs? This might be too detailed in some cases.

4 Appendix

4.1 Input documents

4.2 Standards and References

5 Workspace for discussions, actions and issues

Define and review template for the System Interface Definition deliverable [SPT2TS-128993]

Discussion topic: interface definition vs. interface specification

Visualisation:

<Image: diagram_20240710-1712.51421.mxd>

Approach 1: interface definition (without requirements)

only static + dynamic definitions and descriptions for System A and B (example: protocol description, overview, protocol stack definition (example TACS SCI) --> statement of facts, no "shall" -> no requirements

Approach 2: interface specification (with requirements)

same as approach 1 but with shall statements for each System -> with requirements for System A and System B regarding the interface A <-> B)

General

- There should be no redundancy, means for example that behaviours (e.g. data exchanges, handshakes etc.) are not described here and in the requirements specification again.

Futher evaluation (in relation with Polarion work item content and outputs in Capella model explorer):

- recommendation use approach 1 (better because requirements are then placed at once and distributed across multiple documents which are referring)
- TACS / EULYNX uses already approach 1 in principle, easier to integrate them at later stage
- show exchange items + all the detailed of their exchange item elements (unit, data type, multiplicity, ranges, ...)
- one Polarion document per component exchange (or physical link?)
- considering the use PVMT for attributing OSI layers
- focus on application payload, maybe not lower layers in all cases, just refer then to lower layer specs

Meeting 2024-08-01 (Dennis, Sayfeddine , Gilles)

Let's try first **approach 2**.

Rationale:

- one single point of truth (even if it is splitted into two documents, definition and specification to hold the requirements) At least all subsystems should refer to the same (set of) documents for a given interface.
- Still called it interface definition for now and see how many requirements to be included and where to stop. Important: avoid redundancy of information between interface def. and req. spec. document
Requirements for the interface def. would be: general performance values, master and slave roles, sequencing between the two interfacing systems (e.g. handshakes). The req. specs. would be then refer to this, but only for each corresponding side.
- Some requirements may be included in this documents to have a middleway to establish traceability of system requirements and (or) with the system design satisfies these requirements (to be elaborated in detail).

Glossary definitions to be considered:

 SPT2OD-6831 - FFFIS - Form Fit Functional Interface Specification

 SPT2ARC-1015 - FORM FIT FUNCTIONAL INTERFACE SPECIFICATION

 SPLI-1157 - FUNCTIONAL INTERFACES SPECIFICATION

6 Scope of interface constraints

6.1 Role of interface

DK: If there is the case that there will be a new protocol developed and defined by a domain for e.g. the application layer, should we give the possibility to move it to a separate document (e.g. protocol definition) which is referenced here like the standards? In case of new and detailed content, maybe one single interface definition gets to big.