Copernicus Data Space Ecosystem
Traceability Service Interface Control Document

Doc-ID	CDSE-TRC-TSY
Issue	1.3
Date	11.10.2024
Security classification	ESA unclassified
Data protection level	For official use

VERSION HISTORY

Version	Date	Editor	Changes / Remarks
0.1	21.04.2023	Michael Aspetsberger	Draft
1.0	26.04.2023	Anna Irska, Jolyon Martin, Razvan Cosac	Review
1.1	31.10.2023	Michael Aspetsberger, Aleksander Cesarz	Added staging environment, Trace Event Content Guideline, Contents for L0/AUX
1.2	20.09.2024	Michael Aspetsberger	Clarify token source, clarify content guideline for re-packaged products, update content rule for S2 and S5P.
1.3	11.10.2024	Michael Aspetsberger	Update content rule for S2 and S3

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 General

This document defines the interface of the Traceability Service as part of the Copernicus Data Space Ecosystem. As such, it will outline the general design principles behind the service, document the interaction through APIs and command line tools, give recommendations for its usage, and ultimately provide guidance for additional integrations with the service.

This document is structured as follows:

- **1 Introduction**, this chapter;
- **2 Service Overview**, a general design and context of the service;
- **3 Traceability Service Interfaces**, the API and command line interfaces (CLI);
- **4 Usage Guidelines**, the recommendations for usage;
- **5 Integrating the Traceability Service**, the recommendations for integrations; and
- **ANNEX OpenAPI Specification**, the detailed specification for the API endpoints.

1.2 Applicable Documents

Table 1: Applicable Documents

ID	Document Title
-	-

1.3 Reference Documents

Table 2: Reference documents

ID	Document Title
RD-01	Copernicus Data Space Ecosystem Token Generation[1]
RD-02	Copernicus Data Space Ecosystem Traceability Service CLI Repository[2]

[1] https://documentation.dataspace.copernicus.eu/APIs/Token.html

[2] https://github.com/eu-cdse/trace-cli

2 SERVICE OVERVIEW

This section outlines the purpose and general design principles of the Copernicus Data Space Ecosystem Traceability Service and the context it operates in.

2.1 Purpose of the Service

The purpose of the Traceability Service is to provide the necessary means to track the lifecycle of a data product. The lifecycle begins when the product is created, it includes the copying to a number of archives and possible the deletion thereof, and it would ultimately end when a product is being marked as obsolete. This is visualized in Figure 1 below.

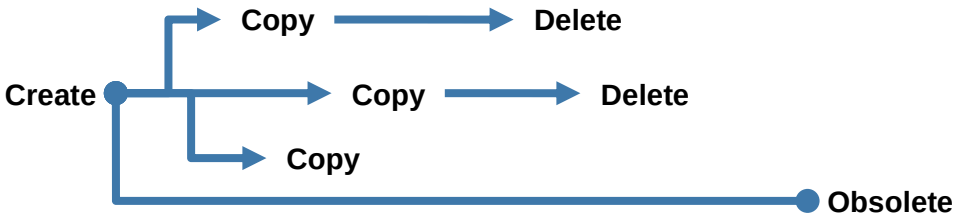

Figure 1: The lifecycle of traces start with Creation, can have several Copies which can be Deleted, and ends with Obsolescence.

The Traceability Service acts as a historian of the product's lifecycle, collecting the traces of all related events. These traces can then be used to check the integrity of a product, its current whereabouts, its impact on other products, or ultimately its inadequacy for continued use in case of obsolescence. Digital signatures on the traces provide the end-users with the ability to verify the integrity and authenticity of the traces themselves, and by extension to detect any alterations of the product along the lifecycle.

It shall be noted explicitly that the Traceability Service does not track the end-usage of the products.

2.2 General Design Principles

The Traceability Service at its core is realized as a RESTful webservice API implementing the OpenAPI specification[3]. The Traceability API can be used from arbitrary clients. While a command-line client (CLI) is supplied, using this CLI should be considered a comfort rather than a requirement.

The algorithms and configurations used for generating product checksums are to be based on open and standard hash functions so that any user can reproduce traces using means other than those supplied by the Traceability Service itself – e.g. by using checksum implementations supplied by the operating system.

The Traceability Service will provide hash functions which clients use to generate traces. The selected hash function is BLAKE3, which is the default supported in the CLI tool.

The Traceability Service makes no assumption on the product format or contents. It is not limited to Copernicus products alone. The traces will include checksums for the full product, and they can include checksums for the product's content when available.

[3] https://www.openapis.org/

In order to understand a products heritage, the traces may contain information for which products the traces were created from. This allows to connect one product to another, e.g. to indicated downstream processing (L0->L1->L2), or when a product has been reformatted. At the same time, this also allows to understand which products are affected by any product marked as being deleted.

The Traceability Service is not implemented as a federated solution, e.g. where each archive or each data producer has their own instance, but instead it is implemented as a centralized solution deployed in the Copernicus Data Space Ecosystem which receives information from various sources.

While there are additional requirements and functionality, this set of principles represents the core idea behind the Copernicus Data Space Ecosystem Traceability Service.

2.3 Service Context

Copernicus Data Space Ecosystem provides free and open access to wide range of data and services from the Copernicus Sentinel missions and more on our plant's land, oceans and atmosphere. Earth Observation data is collected, stored and processed to be made available for end users. The traceability service is an important part of the overall EO data architecture providing insight into the EO product lifecycle. As per the design principles provided above, the Traceability Service is relevant to data producers and archives, who create new traces, and to data consumers, who check a product's integrity. The embedding of the Traceability within the Copernicus Data Space Ecosystem is shown in Figure 2. The Traceability Service relies on the Copernicus Data Space Ecosystem Identity Management to control trace registration. The trace validation and the product integrity checking is possible without an authentication.

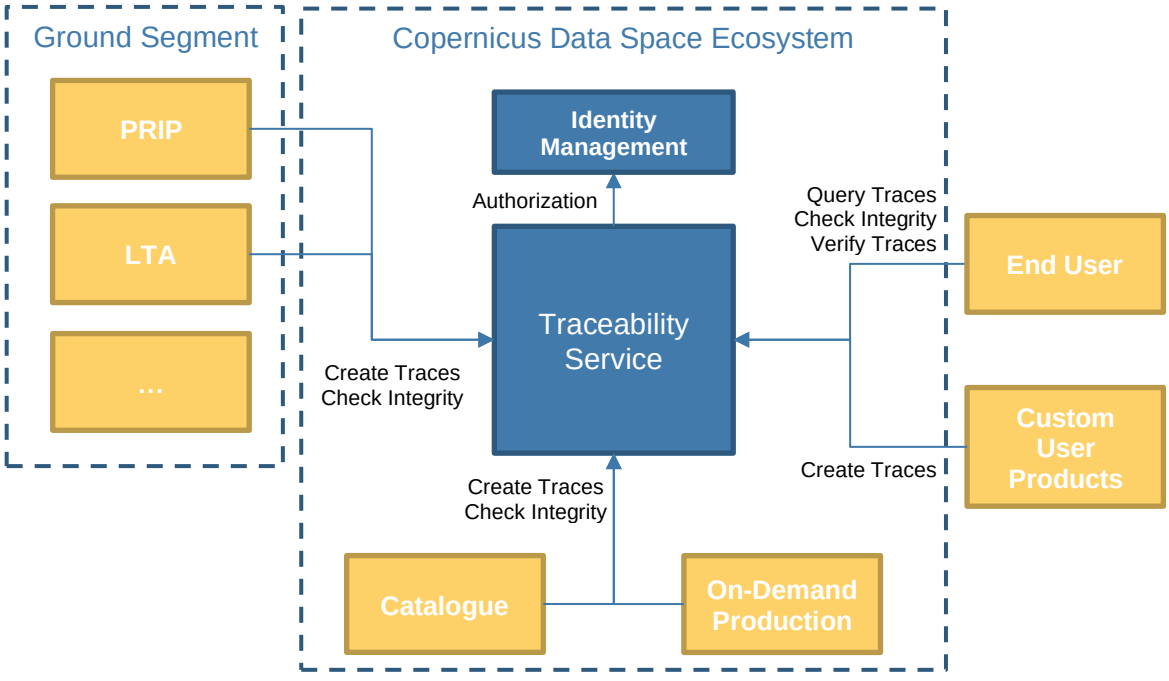

Figure 2: The Traceability Service within the Copernicus Data Space Ecosystem System Context.

The primary interfaces for the Traceability Service are described further in section 3. The Traceability Service is deployed to the endpoints defined in Table 3. The staging deployment is provided for testing the integration. The API is provided with a built-in documentation to simplify integration interactions.

Table 3: The Traceability Service URLs.

Traceability Production Deployment	
API Endpoint URL	https://trace.dataspace.copernicus.eu/api
API Endpoint Documentation	https://trace.dataspace.copernicus.eu/api/docs
API OpenAPI Specification	https://trace.dataspace.copernicus.eu/api/openapi.json
Traceability Staging Deployment	
API Endpoint URL	https://trace-val.dataspace.copernicus.eu/api
API Endpoint Documentation	https://trace-val.dataspace.copernicus.eu/api/docs
API OpenAPI Specification	https://trace-val.dataspace.copernicus.eu/api/openapi.json

3 TRACEABILITY SERVICE INTERFACES

The primary interfaces to the Traceability Service are either directly through the API, or by means of the CLI tool. This section describes the interfaces and the prerequisites to interact with them.

3.1 Prerequisites

The Traceability Service can be used without any credentials when querying trace history or to validate data products for which trace information is available. In the event that new traces are registered, users will require an API key for the Copernicus Data Space Ecosystem and they will need a digital certificate to digitally sign the traces.

3.1.1 API Key for Trace Registration

The API key for trace registration is, depending on the configuration, either provided by the Copernicus Data Space Ecosystem or to be queried from the Copernicus Data Space Ecosystem Identity provider as described in [RD-01].

3.1.2 Digital Certificate for Trace Generation

The Traceability Service uses X.509 digital certificates[4]. The certificates are being supplied by the Copernicus Data Space Ecosystem.

3.2 API Interaction

The Traceability Service is implemented through a RESTful web API following the OpenAPI specification. The OpenAPI specification provides a standardized way to describe and document a service' API including all its endpoints and its data models. The specification for the Traceability Service is included in Annex A to this document and it shall serve as the primary reference for the API interface.

As an overview, the API provides endpoints to:

- Put new Create/Copy/Delete/Obsolete traces
- Get all traces for a given product name
- Get all traces for a given checksum
- Validate whether a given checksum is valid for a given product

The endpoints arguments and response objects are encoded in JSON[5]. In the event of an error, the endpoints will return different return codes depending on the error cause. This is also documented in the specification.

Registering new traces to the service adds them to the internal queue, from which they will be persisted in the internal trace registry. Therefore, there will be a delay of several seconds between the moment when traces are registered and by which they become available.

[4] https://www.itu.int/rec/T-REC-X.509

[5] https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

3.2.1 Interface Versioning

The Traceability Service will undergo an evolution over time. This evolution will also include changes to the API interface. To mitigate the impact of this, the API endpoints include a protocol version identifier in the URL, e.g. “v1” for the v1.0.0 of the service. Whenever there are breaking changes that alter the interface or the functioning in such a way that a client would be incompatible, this protocol version identifier will be incremented, e.g. to “v2”. For other changes, that either only address internal behavior or add functionality the protocol version remains unchanged. As further discriminator, the version of the API interface is included in the OpenAPI specification.

In the event of a protocol change, the previous protocol version will remain available for a migration period, albeit being marked as deprecated and with limitations based on the changes. This gives service users time to adapt to the new version, e.g. by either re-generating the connectors or updating the CLI tool.

3.3 CLI Tool

As a convenience method to interact with the Traceability Service a command line tool is provided for Linux and Windows systems for the x86_64 architecture. The binaries are standalone and require no installation. The binaries are distributed through the CLI source code repository[6].

The command line tool has a documentation included in the binary, which is shown when invoking the tool without arguments, as shown in Table 4 in for Linux and Windows respectively.

Table 4: Trace-CLI invocation instructions.

Linux	Windows
./trace-cli	trace-cli.exe

The general structure for the CLI commands is along the following pattern:

./trace-cli [OPTION…] COMMAND FILE…

where **OPTION** define certain configuration parameters, such as endpoint URL or authentication token, **COMMAND** defines the operation to execute, e.g. trace registration or product verification, and the trailing **FILEs** define the products to operate on.

The CLI tool supports all major API operations, such as trace generation, trace registration, and client-side trace validation. The CLI tool can handle multiple files in one invocation to e.g. check an entire folder of products at once. The CLI tool can either generate and publish traces in one go, or split the process by generating traces locally and then publishing them at the Traceability Service at a later stage.

By default, the CLI tool will limit the logging output to simplify the piping of outputs. If additional output is needed the verbose option can be selected.

[6] https://github.com/eu-cdse/trace-cli

4 USAGE GUIDELINES

The Traceability Service provides a flexible and evolvable solution for data product tracing. This flexibility also opens the door for different usage patterns which could ultimately lead to an inhomogeneous experience for end users. This section presents the recommended usage guidelines for the service.

4.1 Checksum Algorithm

Purpose: The checksum algorithm provides a digital fingerprint of a product. This fingerprint can be used for an accelerated comparison of whether two products are identical, as opposed to comparing every single byte of them.

Guideline: The default checksum algorithm is BLAKE3 with 256 bits of information.

Rationale: BLAKE3[7] has been selected based on its substantially improved performance when compared to SHA-2 or SHA-3. In order to provide out-of-the-box support for product validation using the Traceability Service's CLI tool, traces generated outside of the CLI tool should also provide BLAKE3 checksums. The BLAKE3 team provides the "b3sum" command line utility to generate checksums.

4.2 Origin

Purpose: The origin attribute of a trace defines who created the trace. This is an essential information for end-users to assess the validity and trustworthiness of a trace.

Guideline: The origin shall be bound to the service who creates the trace, not the implementing entity. The origin shall be a compound value of service and domain.

Rationale: It should be easily possible for an end-user to assess a trace origin and whether to trust this or not. The entity providing a certain service may change over time, but the trace origin should not be impacted by this. A descriptive compound value gives end-users the possibility to solicit further information on the trace creator. The compound value also allows the grouping of traces into different spaces.

Example: The Copernicus Data Space Ecosystem uses cdse_csc@dataspace.copernicus.eu as origin for its traces.

Note, the origin attribute is not configurable when submitting traces in the Traceability Service, but it is derived from the submitter's identity. This guideline applies therefore to the definition of new origin labels and the rationale for how they are selected.

4.3 Product Name

Purpose: The Traceability service allows querying traces through two primary ways: by checksum and by name.

Guideline: The name specified in the trace shall correspond to the product's primary identifier. Product name may also have a ".zip" or ".tar" suffix.

[7] https://github.com/BLAKE3-team/BLAKE3

Rationale: When people look for a specific product, they should also be able to find the traces that correspond to it. When archive completeness checks are being made based on the trace information, it should be obvious which products are missing.

Example: S2A_MSIL1C_20230420T100021_N0509_R122_T33UVP_20230420T120027.SAFE.zip for a Sentinel-2 L1C product.

4.4 Product Contents

Purpose: When only partial product contents are available, e.g. individual bands downloaded from a product's zip archive, there needs to be a possibility to check these contents for trace information.

Guideline: The product contents shall be limited to the primary product content. Metadata and auxiliary content shall not be included.

Rationale: The more checksums that are included in a trace, the more data needs to be stored and the more resources are necessary to query for them. When a general-purpose file is included in the contents, attempting to validate it would return an abundance of traces where it is part of. The more checksums for a product need to be made, the longer it takes to generate a trace.

Example: Sentinel-2 MSI L1C products may only include the granule image data in the traces.

4.5 Relevant Product Contents per Trace Event

Purpose: When only partial product contents are available, e.g. individual bands downloaded from a product's zip archive, there needs to be a possibility to check these contents for trace information.

Guideline: Trace contents should be populated for CREATE and OBSOLETE events for original products. Traces contents should not be populated if the product is repackaged. Trace contents should be empty for COPY and DELETE events.

Rationale: The more checksums to be stored in the service the more resources are necessary to store and query for them. Having content checksums is only interesting for those who want to verify individual product parts and, therefore, only the CREATE and OBSOLETE traces are of relevance.

4.6 Product Inputs

Purpose: The inputs define from which products a given product has been derived from. Tracking this information enables the impact analysis when a certain product is e.g. marked as obsolete because of faulty calibration information.

Guideline: The inputs should be limited to the primary product inputs. This includes upstream level data (e.g. L1 for L2 products) and auxiliary datasets. Repackaged products should refer to the original product being repackaged.

Rationale: The limitation to primary inputs should be seen as a way to simplify the trace generation, as for all inputs the ID and checksum needs to be provided.

Example: The Copernicus Data Space Ecosystem processed product traces include references to the original product from which they were generated of.

4.7 Recommended patterns for Sentinel Products

The product name shall correspond to the product name including packaging suffix, e.g.:

S2A_MSIL1C_20230420T100021_N0509_R122_T33UVP_20230420T120027.SAFE.zip

Table 5 lists the recommended content and inputs for the Sentinel products. For L0 and AUX data it is recommended to not include contents (as the data is not commonly distributed or dis-assembled from the packaging), except for S1 RAW data which is distributed to users through the Copernicus Data Space Ecosystem, or as noted in the table.

Table 5: The recommended trace information for Sentinel products.

Mission	Contents	CLI Include Pattern
Sentinel-1	All measurement data.	{*.tiff, *.xml, manifest.safe}
		RAW: {*.dat, manifest.safe}
		OCN: {*.nc, manifest.safe}
Sentinel-2 EUP	All image data.	{IMG_DATA/*.jp2, manifest.safe, MTD_MSIL1C.xml, MTD_MSIL2A.xml, AUX_ECMWFT, AUX_CAMSFO}
Sentinel-2 PDI	TL, DS, and GR for L0, L1C, L2A.	GR: *no contents*
		DS: {manifest.safe}
		TL: {manifest.safe, MTD*.xml, AUX_ECMWFT, AUX_CAMSFO, IMG_DATA/*.jp2}
Sentinel-3	All observation bands.	{*.nc, xfdumanifest.xml}
		L0: {xfdumanifest.xml}
Sentinel-5p	All measurement data.	{*.nc}
Sentinel-6	All measurement data.	{*.nc, xfdumanifest.xml}

5 INTEGRATING THE TRACEABILITY SERVICE

The Traceability Service API is implemented following the OpenAPI specification, which provides a machine readable description of all endpoints. This specification can be used to automatically generate clients for a large variety of programming languages and frameworks, e.g. through the OpenAPI Generator[8]. The automatically generated code will handle all the boiler-plate code related to connection handling and serialization.

As an example, in order to generate Python with the generator referenced above, the following command can be used[9]:

```
java -jar openapi-generator-cli-6.5.0.jar generate
    -i https://trace.dataspace.copernicus.eu/api/openapi.json
    -g python-nextgen -o cdse_trace_client_python
```

The generator will not implement the actual logic, however, e.g. filling the trace contents or calculating checksums. This task remains with the API integrator. As a means to maintain compatibility, a test suite is provided with the CLI tool's sources which can be used by any integrator who seeks builds their own trace generator.

[8] https://github.com/OpenAPITools/openapi-generator

[9] This command needs to be executed in one line, or use '\' on Linux or '^' on Windows for line breaks.

A ANNEX OPENAPI SPECIFICATION

The OpenAPI specification for the Traceability Service is provided along with the API, cf. Table 3, and is included for reference in yaml format as follows:

```yaml
openapi: 3.0.2
info:
  title: Traceability Service - Data Access Service
  version: 1.1.0
servers:
- url: "/api"
paths:
  "/status":
    get:
      tags:
      - Montoring
      summary: Provides monitoring information.
      operationId: ping_status_get
      responses:
        '200':
          description: The server status was retrieved.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/ServerInfo"
  "/v1/traces":
    put:
      tags:
      - Authorized Users
      summary: Registers a new set of traces in the system. All the given traces must
        share the same event and be unique (each combination of product hash and event
        may only occur once).
      description: |-
        This endpoint registers new traces in the system.

        This endpoint accepts a maximum number of 50 traces.
      operationId: put_traces_v1
      requestBody:
        content:
          application/json:
            schema:
              title: Traces
              maxItems: 50
              minItems: 1
              type: array
              items:
                "$ref": "#/components/schemas/RegisterTrace"
        required: true
      responses:
        '201':
          description: Indicates that there were traces successfully queued to be
            registered in the system.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/TraceRegistrations"
        '403':
          description: Traces can only be registered by authorized users.
        '413':
          description: The sent traces data is exceeding the limit.
        '422':
          description: Invalid traces. This can occur if the traces are not valid
            JSON, if too many (more than 50) traces were sent, if traces with different
            events were sent, or if the given traces are not unique (e.g. two traces
            with the same product hash and event were sent).
```

```
    security:
    - HTTPBearer: []
"/v1/traces/{id}":
  get:
    tags:
    - Trace Retrieval
    summary: Returns the trace given its id.
    operationId: get_trace_by_id_v1
    parameters:
    - required: true
      schema:
        title: Id
        type: string
      name: id
      in: path
    responses:
      '200':
        description: The trace of the given id was found.
        content:
          application/json:
            schema:
              "$ref": "#/components/schemas/Trace"
      '404':
        description: No trace for the given id was found.
      '422':
        description: Validation Error
        content:
          application/json:
            schema:
              "$ref": "#/components/schemas/HTTPValidationError"
"/v1/traces/name/{productname}":
  get:
    tags:
    - Trace Retrieval
    summary: Returns all traces for a given product name.
    description: |-
      This endpoint returns the traces (up to 50) for a given product name.

      Multiple trace may exist for a single product to indicate the product's history
      (e.g. creation, copying, obsoletion) and origin.
    operationId: get_trace_by_product_name_v1
    parameters:
    - required: true
      schema:
        title: Productname
        type: string
      name: productname
      in: path
    responses:
      '200':
        description: Searching traces for the given product name was successful.
        content:
          application/json:
            schema:
              title: Response Get Trace By Product Name V1
              type: array
              items:
                "$ref": "#/components/schemas/Trace"
      '422':
        description: Validation Error
        content:
          application/json:
            schema:
              "$ref": "#/components/schemas/HTTPValidationError"
"/v1/traces/hash/{hash}":
  get:
    tags:
    - Trace Retrieval
```

```
      summary: Returns all traces for a given filehash.
      description: |-
        This endpoint returns the traces (up to 50) for a given filehash.

        The filehash may be the product's own hash, or it may be of a product's
contents.
      operationId: search_hash_v1
      parameters:
      - required: true
        schema:
          title: Hash
          type: string
        name: hash
        in: path
      responses:
        '200':
          description: Searching traces for the given filehash was successful.
          content:
            application/json:
              schema:
                title: Response Search Hash V1
                type: array
                items:
                  "$ref": "#/components/schemas/Trace"
        '422':
          description: Validation Error
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/HTTPValidationError"
  "/v1/traces/{productname}/validate":
    get:
      tags:
      - Trace Validation
      summary: Validates a given product against a given filehash.
      description: This endpoint validates a given product against a provided filehash.
      operationId: validate_product
      parameters:
      - required: true
        schema:
          title: Productname
          type: string
        name: productname
        in: path
      - required: true
        schema:
          title: Filehash
          type: string
        name: filehash
        in: query
      responses:
        '200':
          description: The validation has been performed.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/TraceValidation"
        '422':
          description: Validation Error
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/HTTPValidationError"
components:
  schemas:
    Content:
      title: Content
      required:
```

```
    - path
    - hash
    type: object
    properties:
      path:
        title: The path to the content of the product.
        type: string
        example: "/path/to/content"
      hash:
        title: The filehash of the content.
        type: string
        example: f3419f0f7ee5f0c578e7a4f58a7266bdaf08618d3353e72c0647346cd61aaf6f
    description: A product's content contains the path and the filehash
  HTTPValidationError:
    title: HTTPValidationError
    type: object
    properties:
      detail:
        title: Detail
        type: array
        items:
          "$ref": "#/components/schemas/ValidationError"
  Input:
    title: Input
    required:
    - name
    - hash
    type: object
    properties:
      name:
        title: The name of the product used to derive the product from.
        type: string
        example: product-name
      hash:
        title: The filehash of the product which is referred.
        type: string
        example: 77afbb502fc9016d8eacb49b43e45147a9dfdaa319f0ecdd00323872293549fe
    description: The input product used to derive a product from.
  Product:
    title: Product
    required:
    - name
    - size
    - hash
    type: object
    properties:
      name:
        title: The name of the product.
        type: string
        example: product-name
      size:
        title: The product's filesize in number of bytes.
        type: integer
        format: int64
        example: 524288
      hash:
        title: The product's filehash as calculated by the given hash algorithm.
        type: string
        example: 520cfa1ad26ed376ca2be697316d40bc74484ef239c84a4b460f57b90facea54
      contents:
        title: The path and filehashes of the product's contents
        type: array
        items:
          "$ref": "#/components/schemas/Content"
      inputs:
        title: The input products used to derive this product from.
        type: array
        items:
```

```
        "$ref": "#/components/schemas/Input"
    description: A product is either a file itself, or a collection of files.
  RegisterTrace:
    title: RegisterTrace
    required:
    - product
    - event
    - hash_algorithm
    - signature
    type: object
    properties:
      product:
        title: The product for which the trace is generated.
        allOf:
        - "$ref": "#/components/schemas/Product"
      event:
        title: The trace event.
        allOf:
        - "$ref": "#/components/schemas/TraceEvent"
        example: CREATE
      obsolescence:
        title: The message describing the reason why the product is obsolete. Only
          valid for OBSOLETE traces.
        type: string
      hash_algorithm:
        title: The hashing algorithm used to create all filehashes in this trace.
        type: string
        example: BLAKE3
      signature:
        title: The signature of this product as signed by the trace origin.
        allOf:
        - "$ref": "#/components/schemas/Signature"
    description: A trace describes a specific event for a product used for validate
      incoming traces.
  ServerInfo:
    title: ServerInfo
    required:
    - server_version
    - protocol_version
    - status
    type: object
    properties:
      server_version:
        title: Server Version
        type: string
      protocol_version:
        title: Protocol Version
        type: array
        items:
          type: string
      status:
        "$ref": "#/components/schemas/ServerStatus"
    description: The information describing the state of the server.
  ServerStatus:
    title: ServerStatus
    enum:
    - running
    - degraded
    - error
    type: string
    description: The status of the server.
  Signature:
    title: Signature
    required:
    - signature
    - algorithm
    - certificate
    - message
```

```
      type: object
      properties:
        signature:
          title: The digital signature of the message as base64-encoded bytes.
          type: string
          example:
MEUCIQDP4v4KKE1QMfRzKr2bQAkMtEYQ1mxyVLBwqSl985xFiQIgJWEjydCmBQMlkMuCu/NtETbDUUGfXgmv2vJM
Ajld644=
        algorithm:
          title: The signature algorithm used to create the signature.
          type: string
          example: RSA-SHA256
        certificate:
          title: The x509 certificate of the signing authority in DER format as base64-
encoded
            bytes.
          type: string
          example:
MIICBjCCAaugAwIBAgIUbBpJ9cGCZ5Dj0Q+DmYgRmcv/TVgwCgYIKoZIzj0EAwIwWDELMAkGA1UEBhMCQVQxDTAL
BgNVBAcMBExpbnoxITAfBgNVBAoMGENsb3VkZmxpZ2h0IEF1c3RyaWEgR21iSDEXMBUGA1UEAwwOY2xvdWRmbGln
aHQuaW8wHhcNMjMwMzI5MTY1MDQ3WhcNMjQwMzI4MTY1MDQ3WjBYMQswCQYDVQQGEwJBVDENMAsGA1UEBwwETGlu
ejEhMB8GA1UECgwYQ2xvdWRmbGlnaHQgQXVzdHJpYSBHbWJIMRcwFQYDVQQDDA5jbG91ZGZsaWdodC5pbzBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABNLInXtAStyBUW79nOwMtI0cSPJYP23N2I5hcwnog/PWlZS7cqq7IR0MGzSP
evfTtriaKSIP8zxjf2H4sabW4+ajUzBRMB0GA1UdDgQWBBSqtiqY7E/7mTr59Li05nMPMNlV2jAfBgNVHSMEGDAW
gBSqtiqY7E/7mTr59Li05nMPMNlV2jAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQDO+PyCD9ZD
8rUsJP6kDekizStBfqOjDCEeD5qNT/0yuAIhAPLhXSDBnCR926O9cue6R5L9vegs9bVh8saZFCXkeMhP
        message:
          title: The message on which the signature was applied to. This should be
            equal to the trace information in JSON format.
          type: string
          example: '{"hash":"b0d03ae....",...}'
      description: |-
        The trace signature can be used to verify a products integrity.

        The signature is created using an asymmetric, public-key encryption system.
        The bytes being signed correspond to the trace's product dictionary, in lower-
case,
        compact JSON format (i.e. without whitespaces or linebreaks) and encoded in utf-
8.
    Trace:
      title: Trace
      required:
      - product
      - event
      - hash_algorithm
      - signature
      - timestamp
      - origin
      - id
      type: object
      properties:
        product:
          title: The product for which the trace is generated.
          allOf:
          - "$ref": "#/components/schemas/Product"
        event:
          title: The trace event.
          allOf:
          - "$ref": "#/components/schemas/TraceEvent"
          example: CREATE
        obsolescence:
          title: The message describing the reason why the product is obsolete. Only
            valid for OBSOLETE traces.
          type: string
        hash_algorithm:
          title: The hashing algorithm used to create all filehashes in this trace.
          type: string
          example: BLAKE3
```

```
    signature:
      title: The signature of this product as signed by the trace origin.
      allOf:
      - "$ref": "#/components/schemas/Signature"
    timestamp:
      title: The timestamp when the trace has been registered.
      type: string
      format: date-time
    origin:
      title: The creator of this trace, e.g. lta@csc.esa.int
      type: string
    id:
      title: Primary key which identifies a trace.
      type: string
  description: A trace describes a specific event for a product at a specific
    origin with primary id.
TraceEvent:
  title: TraceEvent
  enum:
  - CREATE
  - COPY
  - DELETE
  - OBSOLETE
  type: string
  description: |-
    A trace event describes how the trace comes into life.

    CREATE: A new product is generated.
    COPY: A product is copied to a new location.
    DELETE: A product is removed from a location.
    OBSOLETE: A product is no longer recommended for use.
TraceRegistration:
  title: TraceRegistration
  required:
  - success
  - message
  type: object
  properties:
    success:
      title: Success
      type: boolean
    message:
      title: Message
      type: string
  description: The results of a trace registration.
TraceRegistrations:
  title: TraceRegistrations
  required:
  - success
  - error
  - traces
  type: object
  properties:
    success:
      title: Success
      type: integer
    error:
      title: Error
      type: integer
    traces:
      title: Traces
      type: array
      items:
        "$ref": "#/components/schemas/TraceRegistration"
TraceValidation:
  title: TraceValidation
  required:
  - success
```

```
    - message
    type: object
    properties:
      success:
        title: Success
        type: boolean
      message:
        title: Message
        type: string
    description: The results of a trace validation.
  ValidationError:
    title: ValidationError
    required:
    - loc
    - msg
    - type
    type: object
    properties:
      loc:
        title: Location
        type: array
        items:
          anyOf:
          - type: string
          - type: integer
      msg:
        title: Message
        type: string
      type:
        title: Error Type
        type: string
securitySchemes:
  HTTPBearer:
    type: http
    scheme: bearer
```