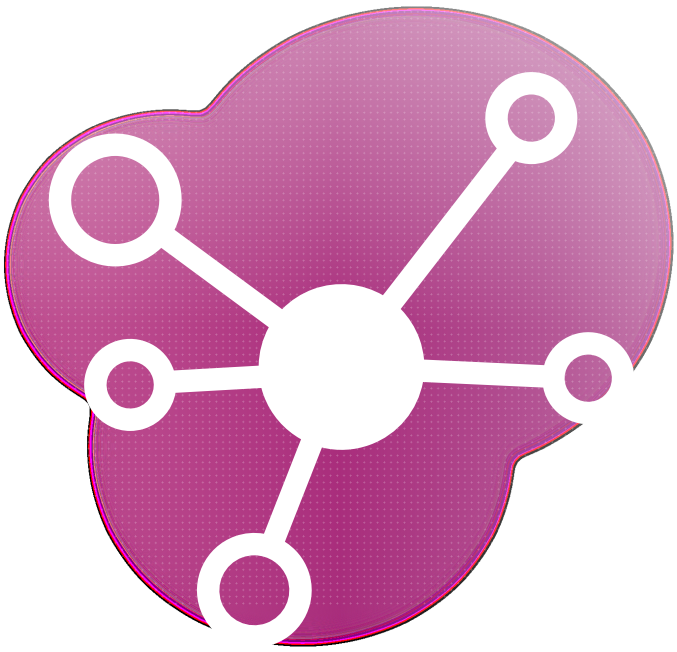
OSIA

Specifications version 2.0



Contents

CHAPTER 1

Introduction

1.1 Problem Statement: vendor lock-in

Target 16.9 of the UN Sustainable Development Goals is to "provide legal identity for all, including birth registration" by the year 2030. But there is a major barrier: the lack of vendor/provider and technology neutrality - commonly known as "vendor lock-in".

The lack of partner and technology neutrality and its consequences becomes apparent when a customer needs to replace one component of the identity management solution with one from another provider, or expand the scope of their solution by linking to new components. Technology barriers are the following:

1. *Solution architectures are not interoperable by design*. The lack of common definitions as to the overall scope of an identity ecosystem, as well as in the main functionalities of a system's components (civil registry, biometric identification system, population registry etc.), blurs the lines between components and leads to inconsistencies. This lack of so-called irreducibly modular architectures makes it difficult, if not impossible, to switch to a third-party component intended to provide the same function and leads to incompatibilities when adding a new component to an existing ecosystem.
2. *Standardized interfaces (APIs) do not exist*. Components are often unable to communicate with each other due to varying interfaces (APIs) and data formats, making it difficult to swap out components or add new ones to the system.

For government policy makers tasked with implementing national identification systems, vendor lock-in is now one of their biggest concerns.

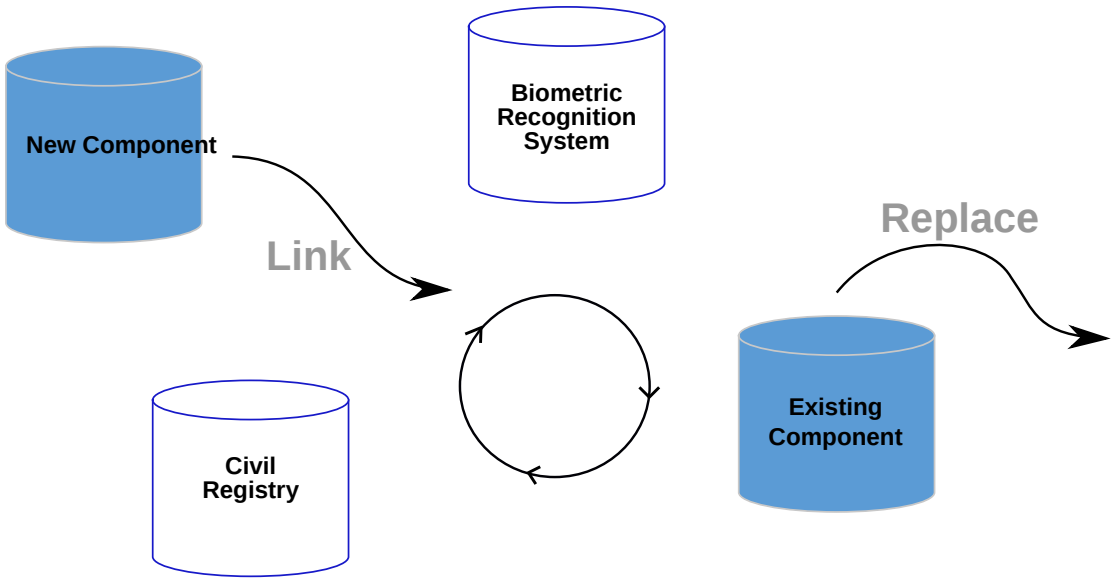


Fig. 1.1: Vendor Lock-In

1.2 The OSIA Initiative

Launched by the not-for-profit Secure Identity Alliance, *Open Standard Identity APIs* (OSIA) is an initiative created for the public good to address vendor lock-in problem.

OSIA addresses the vendor lock-in concern by providing a simple, open standards-based connectivity layer between all key components within the national identity ecosystem.

OSIA scope is as follows:

1. Address the lack of common definitions within the identity ecosystem – NON PRESCRIPTIVE

Components of the identity ecosystem (civil registry, population registry, biometric identification system etc.) from different vendors are functionally incompatible due to the absence of a common definition/understanding of broader functionalities and scope.

OSIA first step has been to formalize definitions, scope and main functionalities of each component within the identity ecosystem.

2. Create a set of standardized interfaces – PRESCRIPTIVE

This core piece of work develops the set of interfaces and standardized data formats to connect the multiple identity ecosystem components to ensure seamless interaction via pre-defined services.

Process of interaction among components (hence type of services each component implements) is down to each government to define and implement according to local laws and regulations.

With OSIA, governments are free to select the components they need, from the suppliers they choose – without fear of lock in.

And because OSIA operates at the interface layer, interoperability is assured without the need to rearchitect environments or rebuild solutions from the ground up. ID ecosystem components are simply swapped in and out as the use case demands – from best-of-breed options already available on the market.

This real-world approach dramatically reduces operational and financial risk, increases the effectiveness of existing identity ecosystems, and rapidly moves government initiatives from proof of concept to live environments.

1.3 Diffusion, Audience, and Access

This specification is hosted in GitHub and can be downloaded from ReadTheDocs.

Any country, technology partner or individual is free to download the functional and technical specifications to implement it in their customized foundational and sectoral ID systems or components. Governments can also reference OSIA as Open Standards in tenders. For more information on how to reference OSIA please see Section *OSIA Versions & Referencing*.

1.4 Document Overview

This document aims at:

- formalizing definitions, scope and main functionalities of each component within the identity ecosystem,
- defining standardized interface and data format to connect the multiple ecosystem components to ensure seamless interaction via pre-defined services.

This document is structured as follows:

- Chapter 1 Introduction
- Chapter 2 Functional View
- Chapter 3 Security and Privacy
- Chapter 4 OSIA Versions and Referencing
- Chapter 5 Interfaces
- Chapter 6 Components

1.5 Convention and Typographical Rules

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Code samples highlighted in blocks appear like that:

```
{
    "key": "value",
    "another_key": 23
}
```

Note: Indicates supplementary explanations and useful tips.

Warning: Indicates that the specific condition or procedure must be respected.

1.6 Revision History

Version	Date	Notes
1.0	2018-12	First release
2.0	2019-06	Second release

CHAPTER 2

Functional View

2.1 Components: Standardized Definition and Scope

OSIA provides seamless interconnection between multiple components part of the identity ecosystem.

The components are defined as follows:

- The *Enrolment* component. Enrollment is defined as a system to register biographic and biometric data of individuals.
- The *Population Registry* (PR) component.

  Population registry is defined as "an individualized data system, that is, a mechanism of continuous recording, or of coordinated linkage, of selected information pertaining to each member of the resident population of a country in such a way to provide the possibility of determining up-to-date information concerning the size and characteristics of that population at selected time intervals. The population register is the product of a continuous process, in which notifications of certain events, which may have been recorded originally in different administrative systems, are automatically linked on a current basis. A. method and sources of updating should cover all changes so that the characteristics of individuals in the register remain current. Because of the nature of a population register, its organization, and also its operation, must have a legal basis."[1]
- The *UIN Generator* component. UIN generator is defined as a system to generate and manage unique identifiers.
- The *Automated Biometric Identification System* (ABIS) component. An ABIS is defined as a *system to detect the identity of an individual when it is unknown, or to verify the individual's identity when it is provided, through biometrics*.
- The *Civil Registry* (CR) component.

  Civil registration is defined as "the continuous, permanent, compulsory and universal recording of the occurrence and characteristics of vital events pertaining to the population, as provided through decree or regulation is accordance with the legal requirement in each country. Civil registration is carried out primarily for the purpose of establishing the documents provided by the law."[2]

[1] *Handbook on Civil Registration and Vital Statistics Systems: Management, Operation and Maintenance, Revision 1, United Nations, New York, 2018, available at:* https://unstats.un.org/unsd/demographic-social/Standards-and-Methods/files/Handbooks/crvs/crvs-mgt-E.pdf , *para 65.*

[2] *Principles and Recommendations for a Vital Statistics System, United Nations publication Sales Number E.13.XVII.10, New York, 2014, paragraph 279*

- The *Document Management System* (DMS) component. DMS is defined as a system to manage the production and issuance of physical documents like ID Cards, passports, driving licenses, etc.
- The *Third Parties Services* component.

Table 2.1: Components

ID Ecosystem Component	Data	Functions
Enrollment	• Alpha* • UIN* • History* • Supporting documents*	• Recording application • Collecting personal data
PR	• Alpha • UIN • History • Supporting documents	• Identity attributes storage • Identity Life cycle management
UIN Gen	• Alpha • UIN	• UIN generation
ABIS	• UIN • Biometric data (images and templates)	• Authentication (1:1) • Identification (1:N) • Quality control and adjudication
CR	• Events • UIN • History • Supporting documents	• Events storage • Certificate production • Workflow
DMS	• Alpha • UIN • History • Supporting documents	• Document data storage • Document Life cycle management • Document Production • Workflow • SMS and email server
Third Parties Services	TBD	KYC/auth

The components are represented on the following diagram:

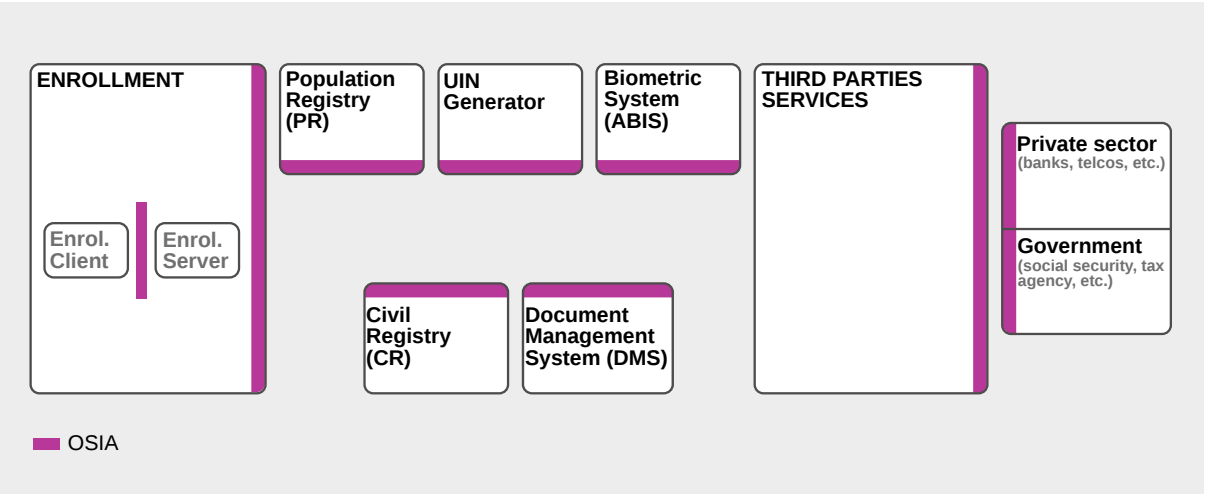


Fig. 2.1: Components

2.2 Interfaces

To do

This chapter describes the following interfaces.

- UIN management. This interface can be implemented by PR, by CR or by another system. We will consider it is provided by a system called *UIN Generator*.
- Notifications. When data is changed, a notification is sent and received by systems that registered for this type of events. For instance, PR can register for the events *birth* emitted by CR.
- Data access. A set of services to access data.

  The design is based on the following assumptions:

  1. All persons recorded in a registry have a *UIN*. The UIN can be used as a key to access person data for all records.
  2. The registries (civil, population, or other) are considered as centralized systems that are connected. If one registry is architectured in a decentralized way, one of its component must be centralized, connected to the network, and in charge of the exchanges with the other registries.
  3. Since the registries are customized for each business needs, dictionaries must be explicitly defined to describe the attributes, the event types, and the document types. See *Data Access* for samples of those dictionaries.
  4. The relationship parent/child is not mandatory in the population registry. A population registry implementation may manage this relationship or may ignore it and rely on the civil registry to manage it.
  5. All persons are stored in the population registry. There is no record in the civil registry that is not also in the population registry.
- Biometrics.
- Third party. Identity based services implemented on top of Identity system mainly *Identity Verification* and *Identity Attribute* sharing.

2.3 Components vs Interfaces Mapping

The interfaces described in this chapter are summarized in the following table:

Table 2.2: Components vs Interfaces Mapping

	Components							
Interfaces	Enroll	PR	UIN gen.	ABIS	CR	ID Card	Funct. Reg	Third Par-ties
Notifications								
Notify event		U			U			
Subscribe		U		U	U	U	U	
Unsubscribe		U		U	U	U	U	
Event callback		I		I	I	I	I	
UIN Management								
Generate UIN		U	I		U	U		
Data Access								
Get Person Attributes	U	IU		U	IU	U	U	U
Match Person Attributes		IU			IU	U	U	U
Verify Person Attributes		IU			IU	U	U	U
Get Person UIN	U	IU			IU	U	U	
Get document		IU			IU			
Biometrics								
Verify	U			I		U	U	U
Identify	U			I		U	U	U
Insert		U		I		U		
Read		U		I		U	U	U
Update		U		I		U		
Delete		U		I		U		
Get Gallery		U		I		U	U	
Get Gallery content		U		I		U	U	
Third Party Services								
Verify ID								I
Identify ID								I
Get Attributes								I
Get Attributes set								I

where:

- `I` is used when a service is implemented (provided) by a component
- `U` is used when a service is used (consumed) by a component

2.4 Use Cases - How to Use OSIA

Introduction to be done

2.4.1 Birth Use Case

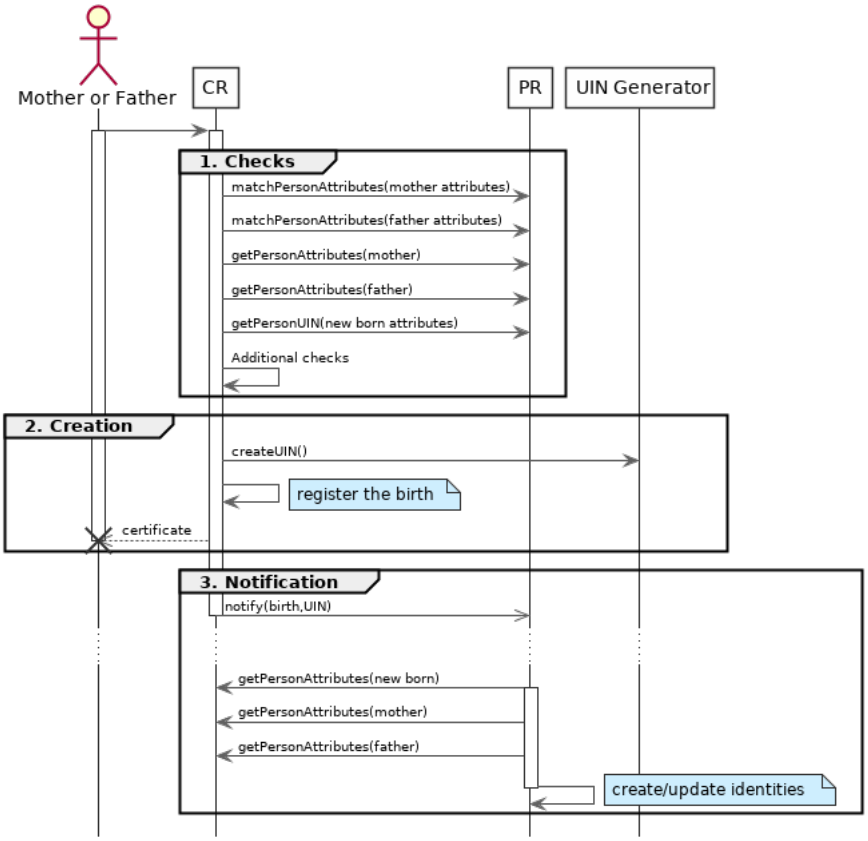


Fig. 2.2: Birth Use Case

1. Checks

   When a request is submitted, the CR may run checks against the data available in the PR using:

   - `matchPersonAttributes`: to check the exactitude of the parents' attributes as known in the PR
   - `getPersonAttributes`: to get missing data about the parents's identity
   - `getPersonUIN`: to check if the new born is already known to PR or not

   How the CR will process the request in case of data discrepancy is specific to each CR implementation and not in the scope of this document.

2. Creation

   The birth is registered in the CR. The first step after the checks is to generate a new UIN a call to `createUIN`.

3. Notification

   As part of the birth registration, it is the responsibility of the CR to notify other systems, including the PR, of this event using:

   - `notify`: to send a *birth* along with the new `UIN`.

   The PR, upon reception of the birth event, will update the identity registry with this new identity using:

   - `getPersonAttributes`: to get the attributes of interest to the PR for the parents and the new child.

2.4.2 Death Use Case

To be completed

2.4.3 Marriage Use Case

To be completed

2.4.4 Deduplication

During the lifetime of a registry, it is possible that duplicates are detected. This can happen for instance after the addition of biometrics in the system. When a registry considers that two records are actually the same and decides to merge them, a notification must be sent.

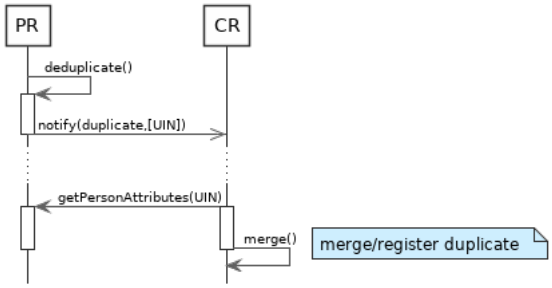


Fig. 2.3: Deduplication Use Case

How the target of the notification should react is specific to each subsystem.

2.4.5 ID Card Request

To be completed

2.4.6 Bank account opening Use Case

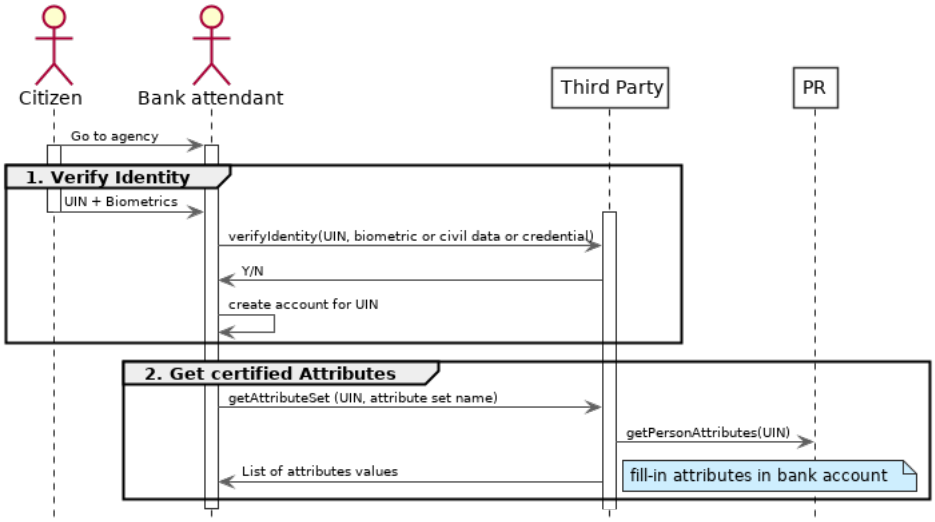


Fig. 2.4: Bank account opening Use Case

2.4.7 Police identity control Use Cases

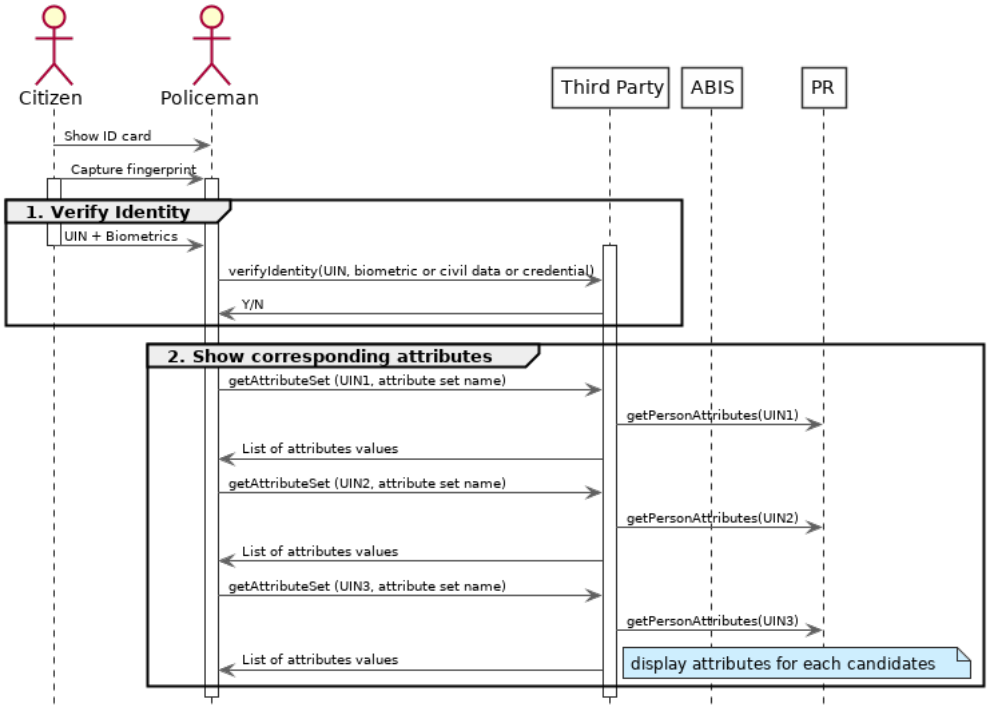


Fig. 2.5: Collaborative identity control

CHAPTER 3

Security & Privacy

3.1 Introduction

Insert diagram of security & privacy features

3.2 Virtual UIN

Explain: using a different UIN in each subsystem - no direct/easy links between the records in different subsystems

3.3 Authorization

To be completed

3.4 GDPR

To be completed

CHAPTER 4

OSIA Versions & Referencing

There will be a version for each interface. Each interface can be referenced in tenders as follows:

```
OSIA – [interface name] v. [version number]
```

For instance below is the string to reference the *Notification* interface:

```
OSIA – Notification v. 1.0
```

Below is the complete list of available interfaces with related version to date:

- OSIA - Notifications - v. 1.0
- OSIA - UIN Management - v. 1.0
- OSIA - Data Access - v. 1.0
- OSIA - Biometrics - v. 1.0
- OSIA - Third Party Services - v. 1.0

This document proposes as well a set of interfaces that could be used by each component (non-prescriptive).

As a consequence, it is possible to reference directly that set of interfaces bundled with a given component. It is possible to reference the bundle of these interfaces as follows:

```
OSIA – [component name] v. [version number]
```

For instance for Civil Registry (CR) OSIA proposes the following set of interfaces:

- OSIA - Notifications - v. 1.0
- OSIA - Data Access - v. 1.0

Below is the string to reference this set of interfaces linked to CR:

```
OSIA – CR v. 1.0
```

CHAPTER 5

Interfaces

5.1 Notification

See *Notification* for the technical details of this interface.

The subscription & notification process is managed by a middleware and is described in the following diagram:

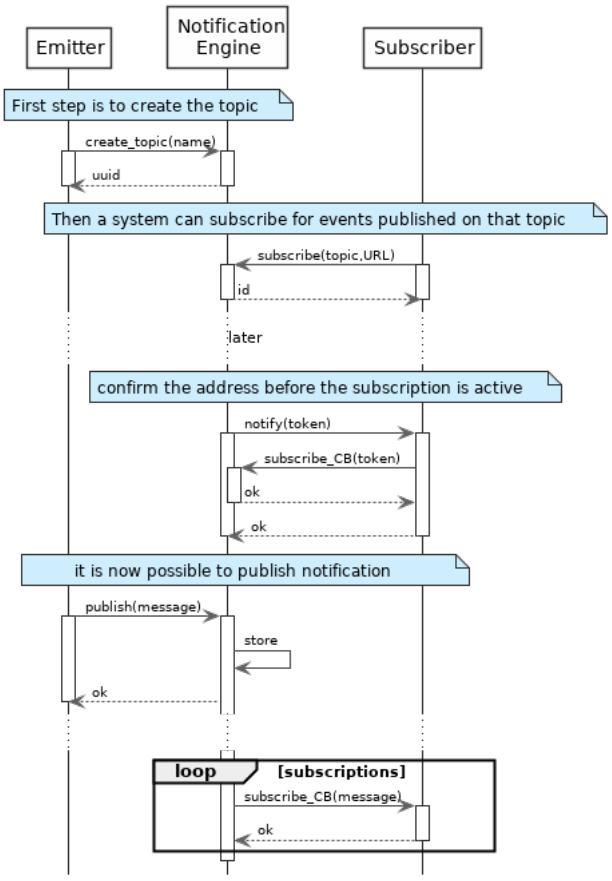


Fig. 5.1: Subscription & Notification Process

5.1.1 Services

subscribe(*topic*, *URL*)
Subscribe a URL to receive notifications sent to one topic

Parameters

- **topic** (*str*) – Topic
- **URL** (*str*) – URL to be called when a notification is available

Returns a subscription ID

This service is synchronous.

unsubscribe(*id*)
Unsubscribe a URL from the list of receiver for one topic

Parameters **id** (*str*) – Subscription ID

Returns bool

This service is synchronous.

confirm(*token*)
Confirm that the URL used during the subscription is valid

Parameters **token** (*str*) – A token send through the URL.

Returns bool

This service is synchronous.

publish(*topic*, *subject*, *message*)
Notify of a new event all systems that subscribed to this topic

Parameters

- **topic** (*str*) – Topic
- **subject** (*str*) – The subject of the message
- **message** (*str*) – The message itself (a string buffer)

Returns N/A

This service is asynchronous (systems that subscribed on this topic are notified asynchronously).

5.1.2 Dictionaries

As an example, below there is a list of events that each component might handle.

Table 5.1: Event Type

Event Type	Emitted by CR	Emitted by PR
Live birth	✓	
Death	✓	
Fœtal Death	✓	
Marriage	✓	
Divorce	✓	
Annulment	✓	
Separation, judicial	✓	
Adoption	✓	
Legitimation	✓	
Recognition	✓	
Change of name	✓	
Change of gender	✓	
New person		✓
Duplicate person	✓	✓

5.2 Data Access

See *Data Access* for the technical details of this interface.

5.2.1 Services

`getPersonAttributes` (*UIN*, *names*)

Retrieve person attributes.

Authorization: To be defined

Parameters

- **`UIN`** (*`str`*) – The person's UIN
- **`names`** (*`list[str]`*) – The names of the attributes requested

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

This service is synchronous. It can be used to retrieve attributes from CR or from PR.

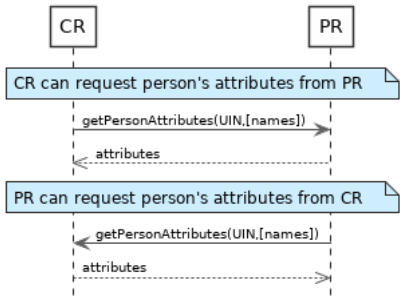


Fig. 5.2: `getPersonAttributes` Sequence Diagram

`matchPersonAttributes` (*UIN*, *attributes*)

Match person attributes. This service is used to check the value of attributes without exposing private data

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **attributes** (*list[(str,str)]*) – The attributes to match. Each attribute is described with its name and the expected value

Returns If all attributes match, a *Yes* is returned. If one attribute does not match, a *No* is returned along with a list of (name,reason) for each non-matching attribute.

This service is synchronous. It can be used to match attributes in CR or in PR.

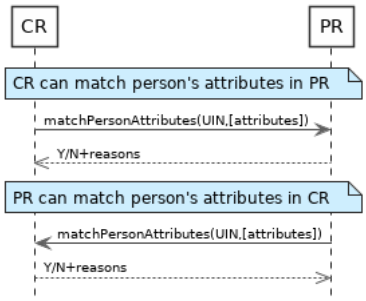


Fig. 5.3: `matchPersonAttributes` Sequence Diagram

---

verifyPersonAttributes (*UIN*, *expressions*)

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **expressions** (*list[(str,str,str)]*) – The expressions to evaluate. Each expression is described with the attribute's name, the operator (one of <, >, =, >=, <=) and the attribute value

Returns A *Yes* if all expressions are true, a *No* if one expression is false, a *Unknown* if To be defined

This service is synchronous. It can be used to verify attributes in CR or in PR.

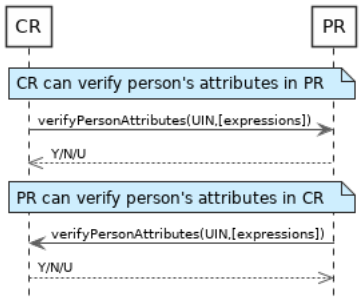


Fig. 5.4: `verifyPersonAttributes` Sequence Diagram

---

getPersonUIN (*attributes*)
Retrieve UIN based on a set of attributes. This service is used when the UIN is unknown.

Authorization: To be defined

Parameters **attributes** (*list[(str,str)]*) – The attributes to be used to find UIN. Each attribute is described with its name and its value

Returns a list of matching UIN

This service is synchronous. It can be used to get the UIN of a person.

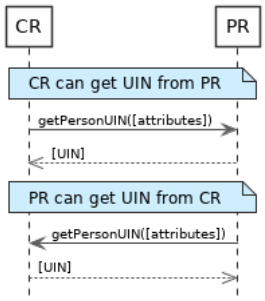


Fig. 5.5: `getPersonUIN` Sequence Diagram

---

getDocument (*UINs*, *documentType*, *format*)
Retrieve in a selected format (PDF, image, . . .) a document such as a marriage certificate.

Authorization: To be defined

Parameters

- **UIN** (*list[str]*) – The list of UINs for the persons concerned by the document
- **documentType** (*str*) – The type of document (birth certificate, etc.)
- **format** (*str*) – The format of the returned/requested document

Returns The list of the requested documents

This service is synchronous. It can be used to get the documents for a person.

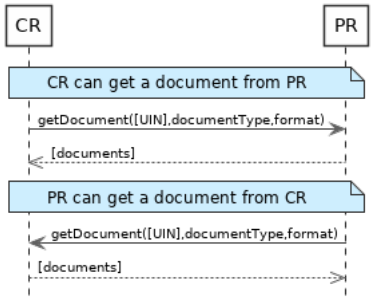


Fig. 5.6: `getDocument` Sequence Diagram

5.2.2 Dictionaries

As an example, below there is a list of attributes/documents that each component might handle.

Table 5.2: Person Attributes

Attribute Name	In CR	In PR	Description
UIN	✓	✓	
first name	✓	✓	
last name	✓	✓	
spouse name	✓	✓	
date of birth	✓	✓	
place of birth	✓	✓	
gender	✓	✓	
date of death	✓	✓	
place of death	✓		
reason of death	✓		
status		✓	Example: missing, wanted, dead, etc.

Table 5.3: Certificate Attributes

Attribute Name	In CR	In PR	Description
officer name	✓		
number	✓		
date	✓		
place	✓		
type	✓		

Table 5.4: Union Attributes

Attribute Name	In CR	In PR	Description
date of union	✓		
place of union	✓		
conjoint1 UIN	✓		
conjoint2 UIN	✓		
date of divorce	✓		

Table 5.5: Filiation Attributes

Attribute Name	In CR	In PR	Description
parent1 UIN	✓		
parent2 UIN	✓		

Table 5.6: Document Type

Document Type	Description
birth certificate	To be completed
death certificate	To be completed
marriage certificate	To be completed

5.3 UIN Management

See *UIN Management* for the technical details of this interface.

5.3.1 Services

`createUIN`(*attributes*)
Generate a new UIN.

Authorization: To be defined

Parameters **attributes** (*list[(str,str)]*) – A list of pair (attribute name, value) that can be used to allocate a new UIN

Returns a new UIN or an error if the generation is not possible

This service is synchronous.

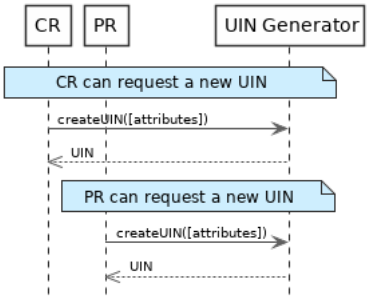


Fig. 5.7: `createUIN` Sequence Diagram

5.4 Biometrics

This interface is described biometric services in the context of an identity system. It is based on the following principles:

- It supports only multi-encounter model, meaning that an identity can have multiple set of biometric data, one for each encounter.
- It does not expose templates (only images) for CRUD services, with one exception to support the use case of documents with biometrics.
- Images can be passed by value or reference. When passed by value, they are base64-encoded.
- Existing standards are used whenever possible, for instance image preferred format is ISO-19794.

Note: Synchronous and Asynchronous Processing

Some services can be very slow depending on the algorithm used, the system workload, etc. Services are described so that:

- If possible, the answer is provided synchronously in the response of the service.
- If not possible for some reason, a status *PENDING* is returned and the answer, when available, is pushed to a callback provided by the client.

If no callback is provided, this indicates that the client wants a synchronous answer, whatever the time it takes.

If a callback is provided, the server will decide if the processing is done synchronously or asynchronously.

See *Biometrics* for the technical details of this interface.

5.4.1 Services

insert (*subjectID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *clientData*, *callback*, *options*)

Insert a new encounter. No identify is performed. This service is synchronous.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs
- **biographicData** (*dict*) – The biographic data (ex: name, date of birth, gender, etc.)
- **contextualData** (*dict*) – The contextual data (ex: encounter date, location, etc.)
- **biometricData** (*list*) – the biometric data (images)
- **clientData** (*bytes*) – additional data not interpreted by the server but stored as is and returned when encounter data is requested.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `algorithm`.

Returns a status indicating success, error, or pending operation. In case of success, the subject ID and the encounter ID are returned. In case of pending operation, the result will be sent later.

read (*subjectID*, *encounterID*, *callback*, *options*)

Retrieve the data of an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are returned.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of success, the encounter data is returned. In case of pending operation, the result will be sent later.

update (*subjectID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *callback*, *options*)

Update an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs
- **biographicData** (*dict*) – The biographic data (ex: name, date of birth, gender, etc.)
- **contextualData** (*dict*) – The contextual data (ex: encounter date, location, etc.)
- **biometricData** (*list*) – the biometric data (images)
- **clientData** (*bytes*) – additional data not interpreted by the server but stored as is and returned when encounter data is requested.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `algorithm`.

Returns a status indicating success, error, or pending operation. In case of success, the subject ID and the encounter ID are returned. In case of pending operation, the result will be sent later.

delete (*subjectID*, *encounterID*, *callback*, *options*)
Delete an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are deleted.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of pending operation, the operation status will be sent later.

getTemplate (*subjectID*, *encounterID*, *biometricType*, *biometricSubType*, *callback*, *options*)
Retrieve the data of an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are returned.
- **biometricType** (*str*) – The type of biometrics to consider
- **biometricSubType** (*str*) – The subtype of biometrics to consider
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of success, a list of template data is returned. In case of pending operation, the result will be sent later.

---

identify (*galleryID*, *filter*, *biometricData*, *callback*, *options*)
Identify a subject using biometrics data and filters on biographic or contextual data. This may include multiple operations, including manual operations.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery.
- **filter** (*dict*) – The input data (filters and biometric data)
- **biometricData** – the biometric data.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `maxNbCand`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A list of candidates is returned, either synchronously or using the callback.

identify (*galleryID*, *filter*, *subjectID*, *callback*, *options*)
Identify a subject using biometrics data of a subject existing in the system and filters on biographic or contextual data. This may include multiple operations, including manual operations.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery.
- **filter** (*dict*) – The input data (filters and biometric data)
- **subjectID** – the subject ID
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `maxNbCand`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A list of candidates is returned, either synchronously or using the callback.

verify (*galleryID*, *subjectID*, *biometricData*, *callback*, *options*)
Verify an identity using biometrics data.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery. If the subject does not belong to this gallery, an error is returned.
- **subjectID** (*str*) – The subject ID
- **biometricData** – The biometric data
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A status (boolean) is returned, either synchronously or using the callback. Optionally, details about the matching result can be provided like the score per biometric and per encounter.

verify (*biometricData1*, *biometricData2*, *callback*, *options*)
Verify that two sets of biometrics data correspond to the same subject.

Authorization: To be defined

Parameters

- **biometricData1** – The first set of biometric data
- **biometricData2** – The second set of biometric data
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A status (boolean) is returned, either synchronously or using the callback. Optionally, details about the matching result can be provided like the score per the biometric.

---

getGalleries (*callback*, *options*)
Get the ID os all the galleries.

Authorization: To be defined

Parameters

- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. A list of gallery ID is returned, either synchronously or using the callback.

getGalleryContent (*galleryID*, *callback*, *options*)

Get the content of one gallery, i.e. the IDs of all the records linked to this gallery.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Gallery whose content will be returned.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. A list of subjects/encounters is returned, either synchronously or using the callback.

5.4.2 Options

Table 5.7: Biometric Services Options

Name	Description
`transactionID`	A string provided by the client application to identity the request being submitted. It is optional in most cases. When provided, it can be used for tracing and debugging. It is mandatory for asynchronous services and is included in the response pushed asynchronously.
`priority`	Priority of the request. Values range from 0 to 9
`maxNbCand`	The maximum number of candidates to return.
`threshold`	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
`algorithm`	Specify the type of algorithm to be used.
`accuracyLevel`	Specify the accuracy expected of the request. This is to support different use cases, when different behavior of the ABIS is expected (response time, accuracy, consolidation/fusion, etc.).

5.4.3 Data Model

Table 5.8: Biometric Data Model

Type	Description	Example
Gallery	A group of subjects related by a common purpose, designation, or status. A subject can belong to multiple galleries.	TBD
Subject	Person who is known to an identity assurance system.	TBD
Encounter	Event in which the client application interacts with a subject resulting in data being collected during or about the encounter. An encounter is characterized by an *identifier* and a *type* (also called *purpose* in some context).	TBD
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	TBD
Filters	a dictionary (list of names and values or *range* of values) describing the filters during a search. Filters can apply on biographic data, contextual data or encounter type.	TBD
Biometric Data	Digital representation of biometric characteristics. As an example, a record containing the image of a finger is a biometric data. All images can be passed by value (image buffer is in the request) or by reference (the address of the image is in the request). All images are compliant with ISO 19794. ISO 19794 allows multiple encoding and supports additional metadata specific to fingerprint, palmprint, portrait or iris.	TBD
Candidate	Information about a candidate found during an identification	TBD
CandidateScore	Detailed information about a candidate found during an identification. It includes the score for the biometrics used.	TBD
Template	A computed buffer corresponding to a biometric and allowing the comparison of biometrics. A template has a format that can be a standard format or a vendor-specific format.	N/A

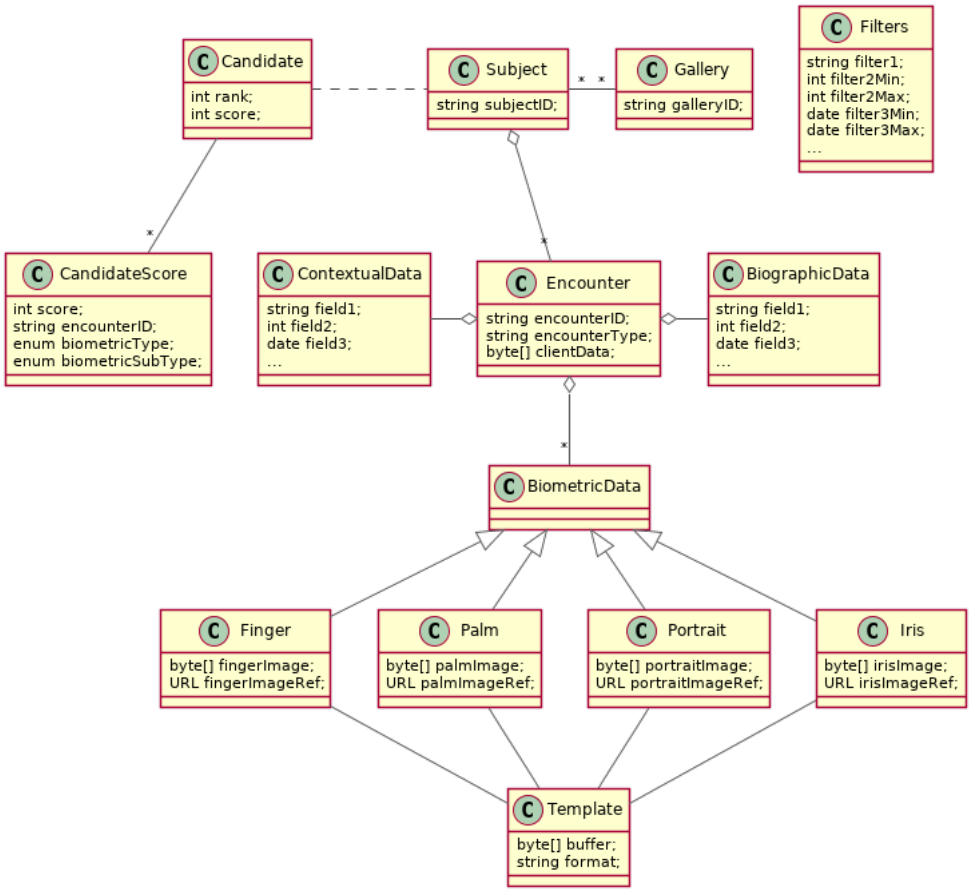


Fig. 5.8: Biometric Data Model

5.5 Document Services

To be defined

5.6 Third Party Services

5.6.1 Services

verifyIdentity (*UIN*[, *IDAttribute*])

Verify Identity based on UIN and set of Identity Attributes. Attributes can be Biometric data, Civil data or a credential.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **IDAttribute** (*list[str]*) – A list of list of pair (name,value) requested

Returns Y or N

In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

identify ([*inIDAttribute*][, *outIDAttribute*])

Identify a person based on a set of inIDAttribute Identity Attributes. Attributes can be Biometric data, Civil data or a credential. Returns list of identities with attributes specified in outIDAttribute

Authorization: To be defined

Parameters

- **inIDAttribute** (*list[str]*) – A list of list of pair (name,value) requested
- **outIDAttribute** (*list[str]*) – A list of list of attribute names requested

Returns Y or N

In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

getAttributes (*UIN*, *names*)

Retrieve person attributes.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **names** (*list[str]*) – The names of the attributes requested

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

getAttributeSet (*UIN*, *setName*)

Retrieve person attributes corresponding to a predefined set name.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **setName** (*str*) – The name of predefined attributes set name

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

CHAPTER 6

Components

This chapter describes for each component the interfaces that it must implement.

6.1 Enrolment Component

6.2 Population Registry

The population registry component MAY implement the following interfaces:

6.2.1 Notification

See *Notification* for the technical details of this interface.

The subscription & notification process is managed by a middleware and is described in the following diagram:

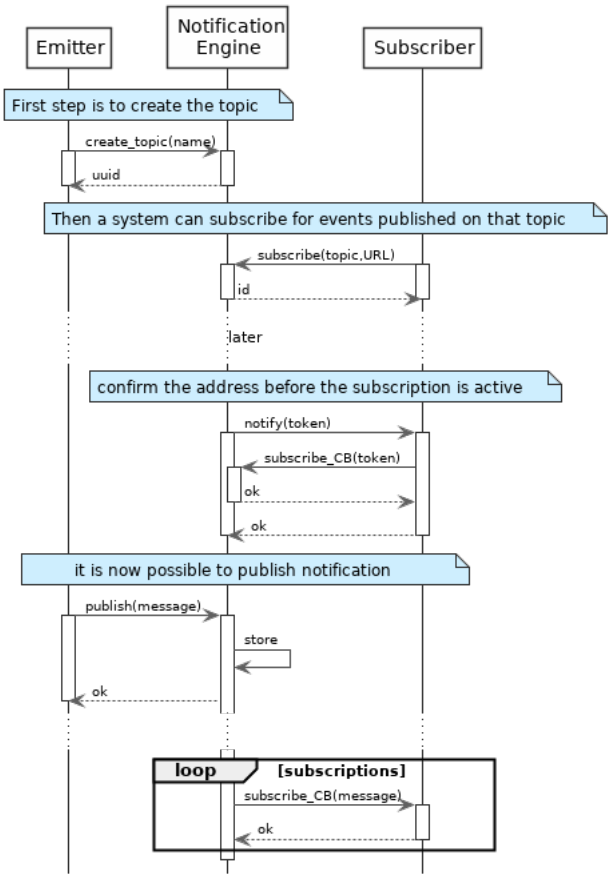


Fig. 6.1: Subscription & Notification Process

Services

subscribe(*topic*, *URL*)
Subscribe a URL to receive notifications sent to one topic

Parameters

- **topic** (*str*) – Topic
- **URL** (*str*) – URL to be called when a notification is available

Returns a subscription ID

This service is synchronous.

unsubscribe(*id*)
Unsubscribe a URL from the list of receiver for one topic

Parameters **id** (*str*) – Subscription ID

Returns bool

This service is synchronous.

confirm(*token*)
Confirm that the URL used during the subscription is valid

Parameters **token** (*str*) – A token send through the URL.

Returns bool

This service is synchronous.

publish (*topic*, *subject*, *message*)
Notify of a new event all systems that subscribed to this topic

Parameters

- **topic** (*str*) – Topic
- **subject** (*str*) – The subject of the message
- **message** (*str*) – The message itself (a string buffer)

Returns N/A

This service is asynchronous (systems that subscribed on this topic are notified asynchronously).

Dictionaries

As an example, below there is a list of events that each component might handle.

Table 6.1: Event Type

Event Type	Emitted by CR	Emitted by PR
Live birth	✓	
Death	✓	
Fœtal Death	✓	
Marriage	✓	
Divorce	✓	
Annulment	✓	
Separation, judicial	✓	
Adoption	✓	
Legitimation	✓	
Recognition	✓	
Change of name	✓	
Change of gender	✓	
New person		✓
Duplicate person	✓	✓

6.2.2 Data Access

See *Data Access* for the technical details of this interface.

Services

getPersonAttributes (*UIN*, *names*)
Retrieve person attributes.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **names** (*list[str]*) – The names of the attributes requested

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

This service is synchronous. It can be used to retrieve attributes from CR or from PR.

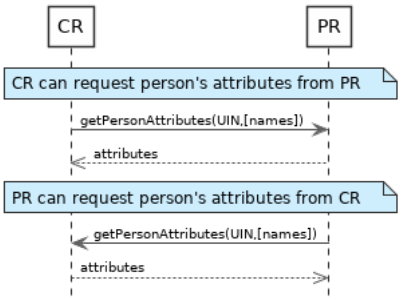


Fig. 6.2: `getPersonAttributes` Sequence Diagram

`matchPersonAttributes` (*UIN*, *attributes*)

Match person attributes. This service is used to check the value of attributes without exposing private data

Authorization: To be defined

Parameters

- **`UIN`** (*`str`*) – The person's UIN
- **`attributes`** (*`list[(str,str)]`*) – The attributes to match. Each attribute is described with its name and the expected value

Returns If all attributes match, a *Yes* is returned. If one attribute does not match, a *No* is returned along with a list of (name,reason) for each non-matching attribute.

This service is synchronous. It can be used to match attributes in CR or in PR.

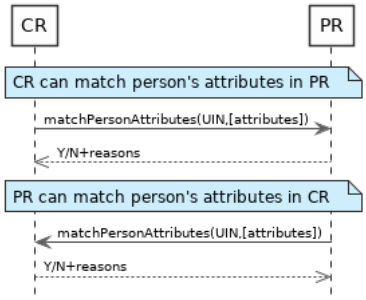


Fig. 6.3: `matchPersonAttributes` Sequence Diagram

`verifyPersonAttributes` (*UIN*, *expressions*)

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data

Authorization: To be defined

Parameters

- **`UIN`** (*`str`*) – The person's UIN
- **`expressions`** (*`list[(str,str,str)]`*) – The expressions to evaluate. Each expression is described with the attribute's name, the operator (one of <, >, =, >=, <=) and the attribute value

Returns A *Yes* if all expressions are true, a *No* if one expression is false, a *Unknown* if To be defined

This service is synchronous. It can be used to verify attributes in CR or in PR.

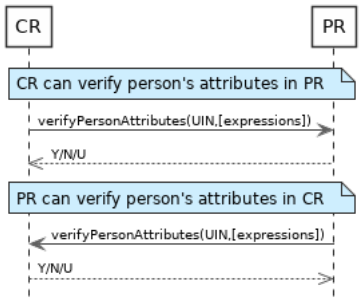


Fig. 6.4: `verifyPersonAttributes` Sequence Diagram

`getPersonUIN` (*attributes*)

Retrieve UIN based on a set of attributes. This service is used when the UIN is unknown.

Authorization: To be defined

Parameters **`attributes`** (*`list[(str,str)]`*) – The attributes to be used to find UIN. Each attribute is described with its name and its value

Returns a list of matching UIN

This service is synchronous. It can be used to get the UIN of a person.

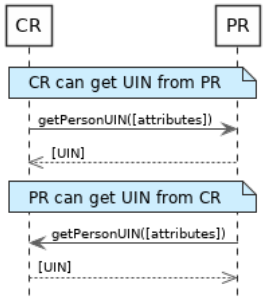


Fig. 6.5: `getPersonUIN` Sequence Diagram

`getDocument` (*UINs*, *documentType*, *format*)

Retrieve in a selected format (PDF, image, . . .) a document such as a marriage certificate.

Authorization: To be defined

Parameters

- **`UIN`** (*`list[str]`*) – The list of UINs for the persons concerned by the document
- **`documentType`** (*`str`*) – The type of document (birth certificate, etc.)
- **`format`** (*`str`*) – The format of the returned/requested document

Returns The list of the requested documents

This service is synchronous. It can be used to get the documents for a person.

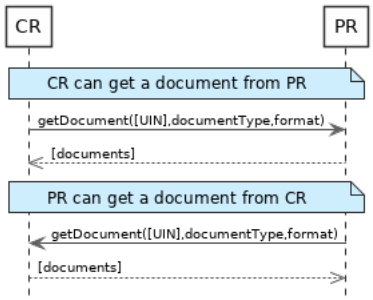


Fig. 6.6: `getDocument` Sequence Diagram

Dictionaries

As an example, below there is a list of attributes/documents that each component might handle.

Table 6.2: Person Attributes

Attribute Name	In CR	In PR	Description
UIN	✓	✓	
first name	✓	✓	
last name	✓	✓	
spouse name	✓	✓	
date of birth	✓	✓	
place of birth	✓	✓	
gender	✓	✓	
date of death	✓	✓	
place of death	✓		
reason of death	✓		
status		✓	Example: missing, wanted, dead, etc.

Table 6.3: Certificate Attributes

Attribute Name	In CR	In PR	Description
officer name	✓		
number	✓		
date	✓		
place	✓		
type	✓		

Table 6.4: Union Attributes

Attribute Name	In CR	In PR	Description
date of union	✓		
place of union	✓		
conjoint1 UIN	✓		
conjoint2 UIN	✓		
date of divorce	✓		

Table 6.5: Filiation Attributes

Attribute Name	In CR	In PR	Description
parent1 UIN	✓		
parent2 UIN	✓		

Table 6.6: Document Type

Document Type	Description
birth certificate	To be completed
death certificate	To be completed
marriage certificate	To be completed

6.3 Civil Registry

The civil registry component MAY implement the following interfaces:

6.3.1 Notification

See *Notification* for the technical details of this interface.

The subscription & notification process is managed by a middleware and is described in the following diagram:

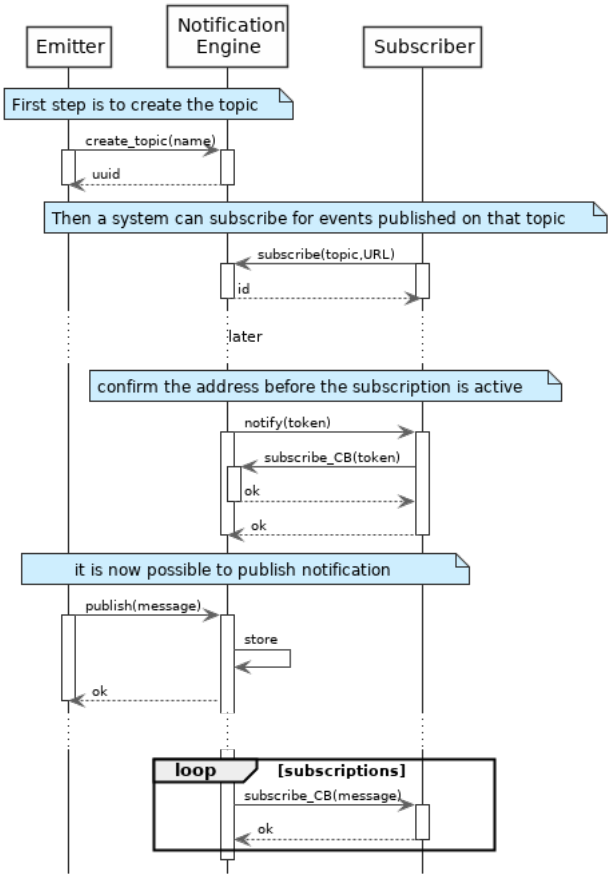


Fig. 6.7: Subscription & Notification Process

Services

`subscribe`(*topic*, *URL*)

Subscribe a URL to receive notifications sent to one topic

Parameters

- **topic** (*str*) – Topic
- **URL** (*str*) – URL to be called when a notification is available

Returns a subscription ID

This service is synchronous.

unsubscribe(*id*)
Unsubscribe a URL from the list of receiver for one topic

Parameters **id** (*str*) – Subscription ID

Returns bool

This service is synchronous.

confirm(*token*)
Confirm that the URL used during the subscription is valid

Parameters **token** (*str*) – A token send through the URL.

Returns bool

This service is synchronous.

publish(*topic*, *subject*, *message*)
Notify of a new event all systems that subscribed to this topic

Parameters

- **topic** (*str*) – Topic
- **subject** (*str*) – The subject of the message
- **message** (*str*) – The message itself (a string buffer)

Returns N/A

This service is asynchronous (systems that subscribed on this topic are notified asynchronously).

Dictionaries

As an example, below there is a list of events that each component might handle.

Table 6.7: Event Type

Event Type	Emitted by CR	Emitted by PR
Live birth	✓	
Death	✓	
Fœtal Death	✓	
Marriage	✓	
Divorce	✓	
Annulment	✓	
Separation, judicial	✓	
Adoption	✓	
Legitimation	✓	
Recognition	✓	
Change of name	✓	
Change of gender	✓	
New person		✓
Duplicate person	✓	✓

6.3.2 Data Access

See *Data Access* for the technical details of this interface.

Services

getPersonAttributes(*UIN*, *names*)

Retrieve person attributes.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **names** (*list[str]*) – The names of the attributes requested

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

This service is synchronous. It can be used to retrieve attributes from CR or from PR.

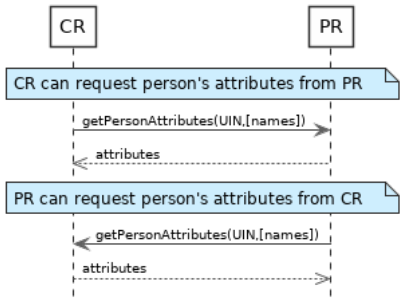


Fig. 6.8: getPersonAttributes Sequence Diagram

---

matchPersonAttributes(*UIN*, *attributes*)

Match person attributes. This service is used to check the value of attributes without exposing private data

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **attributes** (*list[(str,str)]*) – The attributes to match. Each attribute is described with its name and the expected value

Returns If all attributes match, a *Yes* is returned. If one attribute does not match, a *No* is returned along with a list of (name,reason) for each non-matching attribute.

This service is synchronous. It can be used to match attributes in CR or in PR.

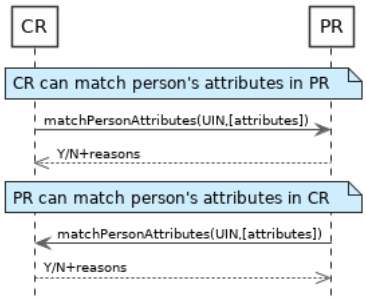


Fig. 6.9: `matchPersonAttributes` Sequence Diagram

`verifyPersonAttributes` (*UIN*, *expressions*)

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data

Authorization: To be defined

Parameters

- **`UIN`** (*`str`*) – The person's UIN
- **`expressions`** (*`list[(str,str,str)]`*) – The expressions to evaluate. Each expression is described with the attribute's name, the operator (one of <, >, =, >=, <=) and the attribute value

Returns A *Yes* if all expressions are true, a *No* if one expression is false, a *Unknown* if To be defined

This service is synchronous. It can be used to verify attributes in CR or in PR.

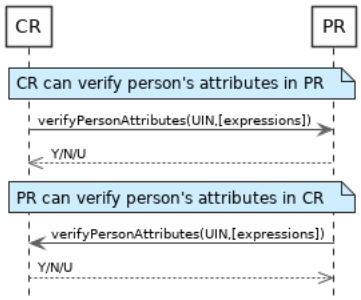


Fig. 6.10: `verifyPersonAttributes` Sequence Diagram

`getPersonUIN` (*attributes*)

Retrieve UIN based on a set of attributes. This service is used when the UIN is unknown.

Authorization: To be defined

Parameters **`attributes`** (*`list[(str,str)]`*) – The attributes to be used to find UIN. Each attribute is described with its name and its value

Returns a list of matching UIN

This service is synchronous. It can be used to get the UIN of a person.

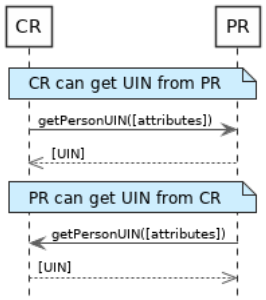


Fig. 6.11: `getPersonUIN` Sequence Diagram

`getDocument` (*UINs*, *documentType*, *format*)

Retrieve in a selected format (PDF, image, . . .) a document such as a marriage certificate.

Authorization: To be defined

Parameters

- **`UIN`** (*`list[str]`*) – The list of UINs for the persons concerned by the document
- **`documentType`** (*`str`*) – The type of document (birth certificate, etc.)
- **`format`** (*`str`*) – The format of the returned/requested document

Returns The list of the requested documents

This service is synchronous. It can be used to get the documents for a person.

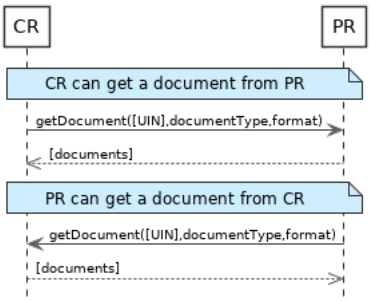


Fig. 6.12: `getDocument` Sequence Diagram

Dictionaries

As an example, below there is a list of attributes/documents that each component might handle.

Table 6.8: Person Attributes

Attribute Name	In CR	In PR	Description
UIN	✓	✓	
first name	✓	✓	
last name	✓	✓	
spouse name	✓	✓	
date of birth	✓	✓	
place of birth	✓	✓	
gender	✓	✓	
date of death	✓	✓	
place of death	✓		
reason of death	✓		
status		✓	Example: missing, wanted, dead, etc.

Table 6.9: Certificate Attributes

Attribute Name	In CR	In PR	Description
officer name	✓		
number	✓		
date	✓		
place	✓		
type	✓		

Table 6.10: Union Attributes

Attribute Name	In CR	In PR	Description
date of union	✓		
place of union	✓		
conjoint1 UIN	✓		
conjoint2 UIN	✓		
date of divorce	✓		

Table 6.11: Filiation Attributes

Attribute Name	In CR	In PR	Description
parent1 UIN	✓		
parent2 UIN	✓		

Table 6.12: Document Type

Document Type	Description
birth certificate	To be completed
death certificate	To be completed
marriage certificate	To be completed

6.4 UIN Generator

The UIN generator component MAY implement the following interfaces:

6.4.1 UIN Management

See *UIN Management* for the technical details of this interface.

Services

createUIN (*attributes*)
Generate a new UIN.

Authorization: To be defined

Parameters **attributes** (*list[(str,str)]*) – A list of pair (attribute name, value) that can be used to allocate a new UIN

Returns a new UIN or an error if the generation is not possible

This service is synchronous.

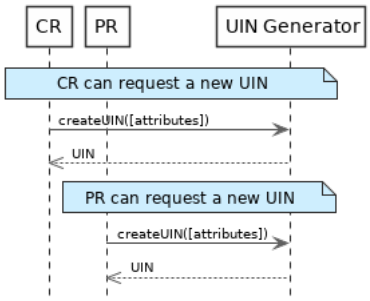


Fig. 6.13: createUIN Sequence Diagram

6.5 ABIS

The ABIS component MAY implement the following interfaces:

6.5.1 Biometrics

This interface is described biometric services in the context of an identity system. It is based on the following principles:

- It supports only multi-encounter model, meaning that an identity can have multiple set of biometric data, one for each encounter.
- It does not expose templates (only images) for CRUD services, with one exception to support the use case of documents with biometrics.
- Images can be passed by value or reference. When passed by value, they are base64-encoded.
- Existing standards are used whenever possible, for instance image preferred format is ISO-19794.

Note: Synchronous and Asynchronous Processing

Some services can be very slow depending on the algorithm used, the system workload, etc. Services are described so that:

- If possible, the answer is provided synchronously in the response of the service.
- If not possible for some reason, a status *PENDING* is returned and the answer, when available, is pushed to a callback provided by the client.

If no callback is provided, this indicates that the client wants a synchronous answer, whatever the time it takes.

If a callback is provided, the server will decide if the processing is done synchronously or asynchronously.

See *Biometrics* for the technical details of this interface.

Services

insert (*subjectID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *clientData*, *callback*, *options*)

Insert a new encounter. No identify is performed. This service is synchronous.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs
- **biographicData** (*dict*) – The biographic data (ex: name, date of birth, gender, etc.)
- **contextualData** (*dict*) – The contextual data (ex: encounter date, location, etc.)
- **biometricData** (*list*) – the biometric data (images)
- **clientData** (*bytes*) – additional data not interpreted by the server but stored as is and returned when encounter data is requested.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `algorithm`.

Returns a status indicating success, error, or pending operation. In case of success, the subject ID and the encounter ID are returned. In case of pending operation, the result will be sent later.

read (*subjectID*, *encounterID*, *callback*, *options*)

Retrieve the data of an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are returned.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of success, the encounter data is returned. In case of pending operation, the result will be sent later.

update (*subjectID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *callback*, *options*)

Update an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs
- **biographicData** (*dict*) – The biographic data (ex: name, date of birth, gender, etc.)

- **contextualData** (*dict*) – The contextual data (ex: encounter date, location, etc.)
- **biometricData** (*list*) – the biometric data (images)
- **clientData** (*bytes*) – additional data not interpreted by the server but stored as is and returned when encounter data is requested.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `algorithm`.

Returns a status indicating success, error, or pending operation. In case of success, the subject ID and the encounter ID are returned. In case of pending operation, the result will be sent later.

delete (*subjectID*, *encounterID*, *callback*, *options*)
Delete an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are deleted.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of pending operation, the operation status will be sent later.

getTemplate (*subjectID*, *encounterID*, *biometricType*, *biometricSubType*, *callback*, *options*)
Retrieve the data of an encounter.

Authorization: To be defined

Parameters

- **subjectID** (*str*) – The subject ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the subject are returned.
- **biometricType** (*str*) – The type of biometrics to consider
- **biometricSubType** (*str*) – The subtype of biometrics to consider
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. In case of success, a list of template data is returned. In case of pending operation, the result will be sent later.

---

identify (*galleryID*, *filter*, *biometricData*, *callback*, *options*)
Identify a subject using biometrics data and filters on biographic or contextual data. This may include multiple operations, including manual operations.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery.

- **filter** (*dict*) – The input data (filters and biometric data)
- **biometricData** – the biometric data.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `maxNbCand`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A list of candidates is returned, either synchronously or using the callback.

identify (*galleryID*, *filter*, *subjectID*, *callback*, *options*)

Identify a subject using biometrics data of a subject existing in the system and filters on biographic or contextual data. This may include multiple operations, including manual operations.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery.
- **filter** (*dict*) – The input data (filters and biometric data)
- **subjectID** – the subject ID
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `maxNbCand`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A list of candidates is returned, either synchronously or using the callback.

verify (*galleryID*, *subjectID*, *biometricData*, *callback*, *options*)

Verify an identity using biometrics data.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Search only in this gallery. If the subject does not belong to this gallery, an error is returned.
- **subjectID** (*str*) – The subject ID
- **biometricData** – The biometric data
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A status (boolean) is returned, either synchronously or using the callback. Optionally, details about the matching result can be provided like the score per biometric and per encounter.

verify (*biometricData1*, *biometricData2*, *callback*, *options*)

Verify that two sets of biometrics data correspond to the same subject.

Authorization: To be defined

Parameters

- **biometricData1** – The first set of biometric data
- **biometricData2** – The second set of biometric data
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`, `threshold`, `accuracyLevel`.

Returns a status indicating success, error, or pending operation. A status (boolean) is returned, either synchronously or using the callback. Optionally, details about the matching result can be provided like the score per the biometric.

---

getGalleries (*callback*, *options*)
Get the ID os all the galleries.

Authorization: To be defined

Parameters

- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. A list of gallery ID is returned, either synchronously or using the callback.

getGalleryContent (*galleryID*, *callback*, *options*)
Get the content of one gallery, i.e. the IDs of all the records linked to this gallery.

Authorization: To be defined

Parameters

- **galleryID** (*str*) – Gallery whose content will be returned.
- **callback** – The address of a service to be called when the result is available.
- **options** (*dict*) – the processing options. Supported options are `transactionID`, `priority`.

Returns a status indicating success, error, or pending operation. A list of subjects/encounters is returned, either synchronously or using the callback.

Options

Table 6.13: Biometric Services Options

Name	Description
`transactionID`	A string provided by the client application to identity the request being submitted. It is optional in most cases. When provided, it can be used for tracing and debugging. It is mandatory for asynchronous services and is included in the response pushed asynchronously.
`priority`	Priority of the request. Values range from 0 to 9
`maxNbCand`	The maximum number of candidates to return.
`threshold`	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
`algorithm`	Specify the type of algorithm to be used.
`accuracyLevel`	Specify the accuracy expected of the request. This is to support different use cases, when different behavior of the ABIS is expected (response time, accuracy, consolidation/fusion, etc.).

Data Model

Table 6.14: Biometric Data Model

Type	Description	Example
Gallery	A group of subjects related by a common purpose, designation, or status. A subject can belong to multiple galleries.	TBD
Subject	Person who is known to an identity assurance system.	TBD
Encounter	Event in which the client application interacts with a subject resulting in data being collected during or about the encounter. An encounter is characterized by an *identifier* and a *type* (also called *purpose* in some context).	TBD
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	TBD
Filters	a dictionary (list of names and values or *range* of values) describing the filters during a search. Filters can apply on biographic data, contextual data or encounter type.	TBD
Biometric Data	Digital representation of biometric characteristics. As an example, a record containing the image of a finger is a biometric data. All images can be passed by value (image buffer is in the request) or by reference (the address of the image is in the request). All images are compliant with ISO 19794. ISO 19794 allows multiple encoding and supports additional metadata specific to fingerprint, palmprint, portrait or iris.	TBD
Candidate	Information about a candidate found during an identification	TBD
CandidateScore	Detailed information about a candidate found during an identification. It includes the score for the biometrics used.	TBD
Template	A computed buffer corresponding to a biometric and allowing the comparison of biometrics. A template has a format that can be a standard format or a vendor-specific format.	N/A

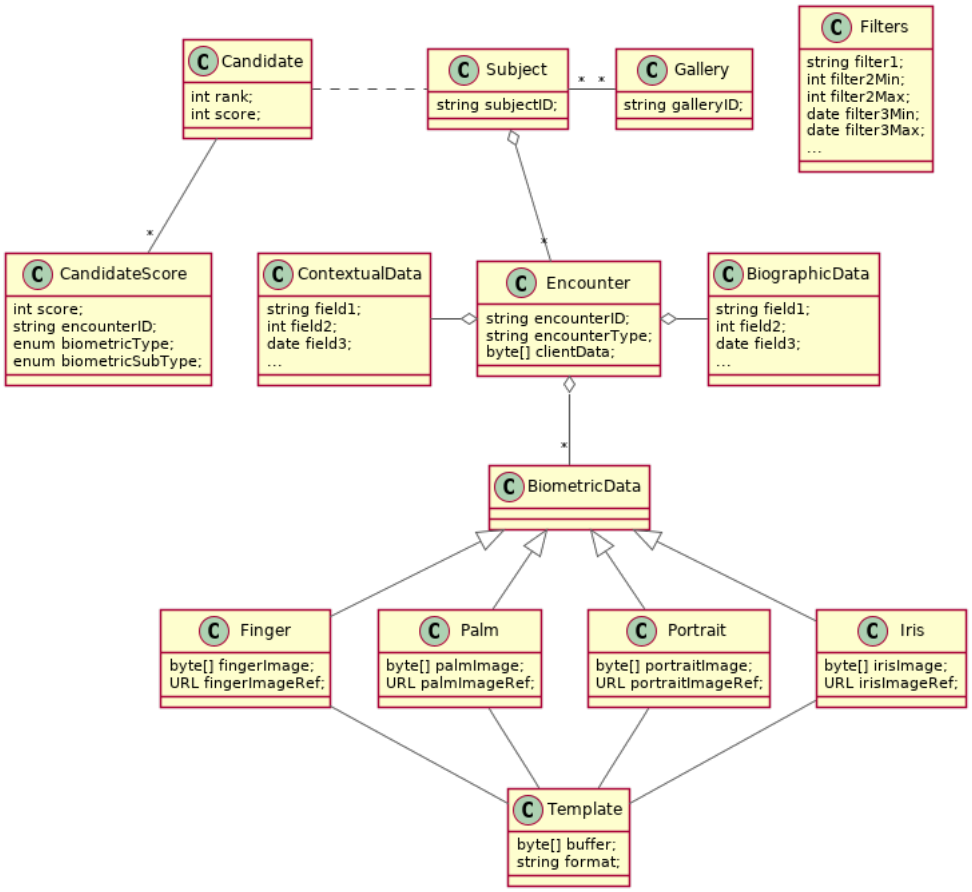


Fig. 6.14: Biometric Data Model

6.6 Document Management System

The document management system component MAY implement the following interfaces:

To be defined

6.7 Third Party

The third party component MAY implement the following interfaces:

6.7.1 Third Party Services

Services

verifyIdentity (*UIN*[, *IDAttribute*])

Verify Identity based on UIN and set of Identity Attributes. Attributes can be Biometric data, Civil data or a credential.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **IDAttribute** (*list[str]*) – A list of list of pair (name,value) requested

Returns Y or N

In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

identify(*inIDAttribute*[, *outIDAttribute*])

Identify a person based on a set of inIDAttribute Identity Attributes. Attributes can be Biometric data, Civil data or a credential. Returns list of identities with attributes specified in outIDAttribute

Authorization: To be defined

Parameters

- **inIDAttribute** (*list[str]*) – A list of list of pair (name,value) requested
- **outIDAttribute** (*list[str]*) – A list of list of attribute names requested

Returns Y or N

In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

getAttributes(*UIN*, *names*)

Retrieve person attributes.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **names** (*list[str]*) – The names of the attributes requested

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

getAttributeSet(*UIN*, *setName*)

Retrieve person attributes corresponding to a predefined set name.

Authorization: To be defined

Parameters

- **UIN** (*str*) – The person's UIN
- **setName** (*str*) – The name of predefined attributes set name

Returns a list of pair (name,value). In case of error (unknown attributes, unauthorized access, etc.) the value is replaced with an error

CHAPTER 7

Annexes

7.1 Glossary

ABIS Automated Biometric Identification System

CR Civil Registry. The system in charge of the continuous, permanent, compulsory and universal recording of the occurrence and characteristics of vital events pertaining to the population, as provided through decree or regulation in accordance with the legal requirements in each country.

DMS Document Management System

Functional systems and registries Managing data including voter rolls, land registry, vehicle registration, passport, residence registry, education, health and benefits.

HTTP Status Codes The HTTP Status Codes are used to indicate the status of the executed operation. The available status codes are described by RFC 7231 and in the IANA Status Code Registry.

Mime Types Mime type definitions are spread across several resources. The mime type definitions should be in compliance with RFC 6838.

Some examples of possible mime type definitions:

```
text/plain; charset=utf-8
application/json
application/vnd.github+json
application/vnd.github.v3+json
application/vnd.github.v3.raw+json
application/vnd.github.v3.text+json
application/vnd.github.v3.html+json
application/vnd.github.v3.full+json
application/vnd.github.v3.diff
application/vnd.github.v3.patch
```

OSIA Open Standard Identity APIs

PR Population Registry. The system in charge of the recording of selected information pertaining to each member of the resident population of a country.

UIN Unique Identity Number.

7.2 Data Format

Conventions about data format in the interface: json, standards for date, images; structure of biographic data

7.3 Technical Specifications

7.3.1 Notification

Download the OpenAPI file for this interface notification.yaml.

Services

Publisher

POST /v1/topics

Create a topic

Create a new topic. This service is idempotent.

Query Parameters

- **name** (*string*) – The topic name (Required)

Status Codes

- 200 OK – Topic was created.
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "uuid": "string",
    "name": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

GET /v1/topics

Get all topics

Status Codes

- 200 OK – Get all topics
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/topics HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "uuid": "string",
        "name": "string"
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

DELETE /v1/topics/{uuid}

Delete a topic

Delete a topic

Parameters

- **uuid** (*string*) – the unique ID returned when the topic was created

Status Codes

- 204 No Content – Topic successfully removed
- 404 Not Found – Topic not found
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/topics/{uuid}/publish

Post a notification to a topic.

Parameters

- **uuid** (*string*) – the unique ID of the topic

Query Parameters

- **subject** (*string*) – the subject of the message.

Status Codes

- 200 OK – Notification published
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
```

(continues on next page)

(continued from previous page)

```
    "message": "string"
}
```

Subscriber

POST /v1/subscriptions

Subscribe to a topic

Subscribes a client to receive event notification.

Subscriptions are idempotent. Subscribing twice for the same topic and endpoint (protocol, address) will return the same subscription ID and the subscriber will receive only once the notifications.

Query Parameters

- **topic** (*string*) – The name of the topic for which notifications will be sent (Required)
- **protocol** (*string*) – The protocol used to send the notification
- **address** (*string*) – the endpoint address, where the notifications will be sent. (Required)
- **policy** (*string*) – The delivery policy, expressing what happens when the message cannot be delivered.

  If not specified, retry will be done every hour for 7 days.

  The value is a set of integer separated by comma:

  – countdown: the number of seconds to wait before retrying. Default: 3600.

  – max: the maximum max number of retry. -1 indicates infinite retry. Default: 168

Status Codes

- 200 OK – Subscription successfully created. Waiting for confirmation message.
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "uuid": "string",
    "topic": "string",
    "protocol": "http",
    "address": "string",
    "policy": "string",
    "active": true
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: onEvent

POST {$request.query.address}

Status Codes

- 200 OK – Message received and processed.
- 500 Internal Server Error – Unexpected error

Request Headers

- *message-type* – the type of the message (Required)
- *subscription-id* – the unique ID of the subscription
- *message-id* – the unique ID of the message (Required)
- *topic-id* – the unique ID of the topic (Required)

Example request:

```
POST {$request.query.address} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "type": "SubscriptionConfirmation",
    "token": "string",
    "topic": "string",
    "message": "string",
    "messageId": "string",
    "subject": "string",
    "subscribeURL": "https://example.com",
    "timestamp": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

`GET /v1/subscriptions`

Get all subscriptions

Status Codes

- 200 OK – Get all subscriptions
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/subscriptions HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "uuid": "string",
        "topic": "string",
        "protocol": "http",
        "address": "string",
        "policy": "string",
        "active": true
```

(continues on next page)

(continued from previous page)

```
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

DELETE /v1/subscriptions/{uuid}

Unsubscribe from a topic

Unsubscribes a client from receiving notifications for a topic

Parameters

- **uuid** (*string*) – the unique ID returned when the subscription was done

Status Codes

- 204 No Content – Subscription successfully removed
- 404 Not Found – Subscription not found
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

GET /v1/subscriptions/confirm

Confirm the subscription

Confirm a subscription

Query Parameters

- **token** (*string*) – the token sent to the endpoint (Required)

Status Codes

- 200 OK – Subscription successfully confirmed
- 400 Bad Request – Invalid token
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/subscriptions/confirm?token=string HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Notification Message

This section describes the messages exchanged through notification. All messages are encoded in `json`. They are generated by the emitter (the source of the event) and received by zero, one, or many receivers that have subscribed to the type of event.

Table 7.1: Event Type & Message

Event Type	Message
`liveBirth`	• `source`: identification of the system emitting the event • `uin` of the new born • `uin1` of the first parent (optional if parent is unknown) • `uin2` of the second parent (optional if parent is unknown) Example: `{ "source": "systemX", "uin": "123456789", "uin1": "123456789", "uin2": "234567890" }`
`death`	• `source`: identification of the system emitting the event • `uin` of the dead person Example: `{ "source": "systemX", "uin": "123456789" }`
`birthCancellation`	• `source`: identification of the system emitting the event • `uin` of the person whose birth declaration is being cancelled Example: `{ "source": "systemX", "uin": "123456789", }`
`foetalDeath`	• `source`: identification of the system emitting the event • `uin` of the new born Example: `{ "source": "systemX", "uin": "123456789" }`
`marriage`	• `source`: identification of the system emitting the event • `uin1` of the first conjoint • `uin2` of the second conjoint Example: `{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }`

Continued on next page

Table 7.1 – continued from previous page

Event Type	Message
`divorce`	• `source`: identification of the system emitting the event • `uin1` of the first conjoint • `uin2` of the second conjoint Example: `{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }`
`annulment`	• `source`: identification of the system emitting the event • `uin1` of the first conjoint • `uin2` of the second conjoint Example: `{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }`
`separation`	• `source`: identification of the system emitting the event • `uin1` of the first conjoint • `uin2` of the second conjoint Example: `{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }`
`adoption`	• `source`: identification of the system emitting the event • `uin` of the child • `uin1` of the first parent • `uin2` of the second parent (optional) Example: `{ "source": "systemX", "uin": "123456789", "uin1": "234567890" }`
`legitimation`	• `source`: identification of the system emitting the event • `uin` of the child • `uin1` of the first parent • `uin2` of the second parent (optional) Example: `{ "source": "systemX", "uin": "987654321", "uin1": "123456789", "uin2": "234567890" }`

Continued on next page

Table 7.1 – continued from previous page

Event Type	Message
`recognition`	• `source`: identification of the system emitting the event • `uin` of the child • `uin1` of the first parent • `uin2` of the second parent (optional) Example: `{ "source": "systemX", "uin": "123456789", "uin2": "234567890" }`
`changeOfName`	• `source`: identification of the system emitting the event • `uin` of the person Example: `{ "source": "systemX", "uin": "123456789" }`
`changeOfGender`	• `source`: identification of the system emitting the event • `uin` of the person Example: `{ "source": "systemX", "uin": "123456789" }`
`updatePerson`	• `source`: identification of the system emitting the event • `uin` of the person Example: `{ "source": "systemX", "uin": "123456789" }`
`newPerson`	• `source`: identification of the system emitting the event • `uin` of the person Example: `{ "source": "systemX", "uin": "123456789" }`

Continued on next page

Table 7.1 – continued from previous page

Event Type	Message
`duplicatePerson`	• `source`: identification of the system emitting the event • `uin` of the person to be kept • `duplicates`: list of uin for records identified as duplicates Example: `{ "source": "systemX", "uin": "123456789", "duplicates": [ "234567890", "345678901" ] }`

Note: Anonymized notification of events will be treated separately.

7.3.2 UIN Management

Download the OpenAPI file for this interface uin.yaml.

Services

`POST /v1/uin`

Request the generation of a new UIN.

The request body should contain a list of attributes and their value, formatted as a json dictionary.

Status Codes

- 200 OK – UIN is generated
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Service forbidden
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
POST http://server.com/v1/uin HTTP/1.1
Host: server.com
Content-Type: application/json

{
      "firstName": "John",
      "lastName": "Doo",
      "dateOfBirth": "1984-11-19"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1235567890
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
      "code": 1,
      "message": "string"
}
```

7.3.3 Data Access

Download the OpenAPI file for this interface dataaccess.yaml.

Services

GET /v1/persons
Retrieve a UIN based on a set of attributes. This service is used when the UIN is unknown.

Query Parameters

- **attributes** (*object*) – The attributes used to retrieve the UIN (Required)

Status Codes

- 200 OK – All UIN found (a list of at least one UIN)
- 400 Bad Request – Invalid parameter
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Service forbidden
- 404 Not Found – No UIN found
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
GET /v1/persons?firstName=John&lastName=Do HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    "1235567890"
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

GET /v1/persons/{uin}
Retrieve attributes for a person.

Parameters

- **uin** (*string*) – Unique Identity Number

Query Parameters

- **attributeNames** (*array*) – The names of the attributes requested for this person (Required)

Status Codes

- 200 OK – Requested attributes values or *Error* description.
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Service forbidden
- 404 Not Found – Unknown uin
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
GET /v1/persons/{uin}?attributeNames=firstName&attributeNames=lastName&
↪attributeNames=dob HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "firstName": "John",
    "lastName": "Doo",
    "dob": {
        "code": 1023,
        "message": "Unknown attribute name"
    }
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/persons/{uin}/match

Match person attributes. This service is used to check the value of attributes without exposing private data.

The request body should contain a list of attributes and their value, formatted as a json dictionary.

Parameters

- **uin** (*string*) – Unique Identity Number

Status Codes

- 200 OK – Information about non matching attributes. Returns a list of matching result (See *Matching Error*) An empty list indicates all attributes were matching.
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Service forbidden
- 404 Not Found – Unknown uin
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
POST /v1/persons/{uin}/match HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1984-11-19"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "attributeName": "firstName",
        "errorCode": 1
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

`POST /v1/persons/{uin}/verify`

Evaluate expressions (See *Expression*) on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data

The request body should contain a list of *Expression*.

Parameters

- **uin** (*string*) – Unique Identity Number

Status Codes

- 200 OK – The expressions are all true (true is returned) or one is false (false is returned)
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Forbidden access. The service is forbidden or one of the attributes is forbidden.
- 404 Not Found – Unknown uin
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
POST /v1/persons/{uin}/verify HTTP/1.1
Host: example.com
Content-Type: application/json

[
    {
        "attributeName": "firstName",
        "operator": "=",
        "value": "John"
    },
    {
        "attributeName": "dateOfBirth",
        "operator": "<",
```

(continues on next page)

(continued from previous page)

```
        "value": "1990-12-31"
    }
]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

true
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

GET /v1/persons/{uin}/document

Retrieve in an unstructured format (PDF, image) a document such as a marriage certificate.

Parameters

- **uin** (*string*) – Unique Identity Number

Query Parameters

- **secondaryUin** (*string*) – Unique Identity Number of a second person linked to the requested document. Example: wife, husband
- **doctype** (*string*) – The type of document (Required)
- **format** (*string*) – The expected format of the document. If the document is not available at this format, it must be converted. TBD: one format for certificate data. (Required)

Status Codes

- 200 OK – The document(s) is/are found and returned, as binary data in a MIME multi-part structure.
- 401 Unauthorized – Client must be authenticated
- 403 Forbidden – Service forbidden
- 404 Not Found – Unknown uin
- 415 Unsupported Media Type – Unsupported format
- 500 Internal Server Error – Unexpected error (See *Error*)

Example request:

```
GET /v1/persons/{uin}/document?doctype=marriage&secondaryUin=234567890&format=pdf HTTP/1.
↪1
Host: example.com
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Data Model

Person Attributes

When exchanged in the services described in this document, the persons attributes will apply the following rules:

Table 7.2: Person Attributes

Attribute Name	Description	Format
`uin`	Unique Identity Number	Text
`firstName`	First name	Text
`lastName`	Last name	Text
`spouseName`	Spouse name	Text
`dateOfBirth`	Date of birth	Date (iso8601). Example: `1987-11-17`
`placeOfBirth`	Place of birth	Text
`gender`	Gender	Number (iso5218). One of 0 (Not known), 1 (Male), 2 (Female), 9 (Not applicable)
`dateOfDeath`	Date of death	Date (iso8601). Example: `2018-11-17`
`placeOfDeath`	Place of death	Text
`reasonOfDeath`	Reason of death	Text
`status`	Status. Example: missing, wanted, dead, etc.	Text

Matching Error

A list of:

Table 7.3: Matching Error Object

Attribute	Type	Description	Mandatory
`attributeName`	String	Attribute name (See *Person Attributes*)	Yes
`errorCode`	32 bits integer	Error code. Possible values: `0` (attribute does not exist); `1` (attribute exists but does not match)	Yes

Expression

Table 7.4: Expression Object

Attribute	Type	Description	Mandatory
`attributeName`	String	Attribute name (See *Person Attributes*)	Yes
`operator`	String	Operator to apply. Possible values: <, >, =, >=, <=	Yes
`value`	string, or integer, or boolean	The value to be evaluated	Yes

Error

Table 7.5: Error Object

Attribute	Type	Description	Mandatory
`code`	32 bits integer	Error code	Yes
`message`	String	Error message	Yes

7.3.4 Biometrics

Download the OpenAPI file for this interface abis.yaml.

Services

CRUD

GET /v1/subjects/{subjectId}/{encounterId}/templates
Get biometrics templates

Parameters

- **subjectId** (*string*) – the id of the subject
- **encounterId** (*string*) – the id of the encounter

Query Parameters

- **biometricType** (*string*) – the type of biometrics to return
- **biometricSubType** (*string*) – the sub-type of biometrics to return
- **templateFormat** (*string*) – the format of the template to return
- **qualityFormat** (*string*) – the format of the quality to return
- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 200 OK – Operation successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 404 Not Found – Unknown record or unkown biometrics
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/subjects/{subjectId}/{encounterId}/templates HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "template": "c3RyaW5n",
        "templateFormat": "ISO_19794_2",
        "quality": 1,
        "qualityFormat": "ISO_19794",
        "vendor": "string",
```

(continues on next page)

(continued from previous page)

```
        "algorithm": "string"
    }
]
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: getTemplateResponse

`POST ${request.query.callback}`

Response callback

Query Parameters

- **`transactionId`** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
    {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "template": "c3RyaW5n",
        "templateFormat": "ISO_19794_2",
        "quality": 1,
        "qualityFormat": "ISO_19794",
        "vendor": "string",
        "algorithm": "string"
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/subjects/{subjectId}/{encounterId}

Insert one encounter

Parameters

- **subjectId** (*string*) – the id of the subject
- **encounterId** (*string*) – the id of the encounter

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **algorithm** (*string*) – Hint about the algorithm to be used

Status Codes

- 201 Created – Insertion successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Insertion not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/subjects/{subjectId}/{encounterId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": [
        {
            "encounterType": "string",
            "clientData": "c3RyaW5n",
            "contextualData": {
                "date": "2019-05-21"
            },
            "biographicData": {
                "dateOfBirth": "2019-05-21",
                "gender": "M",
                "nationality": "string"
            },
            "biometricData": [
                {
                    "biometricType": "FACE",
                    "biometricSubType": "UNKNOWN",
                    "image": "c3RyaW5n",
                    "imageRef": "https://example.com",
                    "captureDate": "2019-05-21",
                    "captureDevice": "string",
                    "impressionType": "LIVE_SCAN_PLAIN",
                    "width": 1,
                    "height": 1,
                    "bitdepth": 1,
                    "resolution": 1,
                    "compression": "NONE"
                }
            ]
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: insertResponse

`POST ${request.query.callback}`

Response callback

Query Parameters

- **`transactionId`** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "status": "OK",
    "subjectId": "string",
    "encounterId": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

`GET /v1/subjects/{subjectId}/{encounterId}`

Read one encounter

Parameters

- **`subjectId`** (*string*) – the id of the subject
- **`encounterId`** (*string*) – the id of the encounter

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 200 OK – Read successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 404 Not Found – Unknown record
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/subjects/{subjectId}/{encounterId} HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": {
        "encounterType": "string",
        "clientData": "c3RyaW5n",
        "contextualData": {
            "date": "2019-05-21"
        },
        "biographicData": {
            "dateOfBirth": "2019-05-21",
            "gender": "M",
            "nationality": "string"
        },
        "biometricData": [
            {
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN",
                "image": "c3RyaW5n",
                "imageRef": "https://example.com",
                "captureDate": "2019-05-21",
                "captureDevice": "string",
                "impressionType": "LIVE_SCAN_PLAIN",
                "width": 1,
                "height": 1,
                "bitdepth": 1,
                "resolution": 1,
                "compression": "NONE"
            }
        ]
    }
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: readResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": {
        "encounterType": "string",
        "clientData": "c3RyaW5n",
        "contextualData": {
            "date": "2019-05-21"
        },
        "biographicData": {
            "dateOfBirth": "2019-05-21",
            "gender": "M",
            "nationality": "string"
        },
        "biometricData": [
            {
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN",
                "image": "c3RyaW5n",
                "imageRef": "https://example.com",
                "captureDate": "2019-05-21",
                "captureDevice": "string",
                "impressionType": "LIVE_SCAN_PLAIN",
                "width": 1,
                "height": 1,
                "bitdepth": 1,
                "resolution": 1,
                "compression": "NONE"
            }
        ]
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

PUT /v1/subjects/{subjectId}/{encounterId}
Update one encounter

Parameters

- **subjectId** (*string*) – the id of the subject
- **encounterId** (*string*) – the id of the encounter

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **algorithm** (*string*) – Hint about the algorithm to be used

Status Codes

- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 204 No Content – Update successful
- 400 Bad Request – Bad request
- 403 Forbidden – Update not allowed
- 404 Not Found – Unknown record
- 500 Internal Server Error – Unexpected error

Example request:

```
PUT /v1/subjects/{subjectId}/{encounterId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": [
        {
            "encounterType": "string",
            "clientData": "c3RyaW5n",
            "contextualData": {
                "date": "2019-05-21"
            },
            "biographicData": {
                "dateOfBirth": "2019-05-21",
                "gender": "M",
```

(continues on next page)

(continued from previous page)

```
                "nationality": "string"
            },
            "biometricData": [
                {
                    "biometricType": "FACE",
                    "biometricSubType": "UNKNOWN",
                    "image": "c3RyaW5n",
                    "imageRef": "https://example.com",
                    "captureDate": "2019-05-21",
                    "captureDevice": "string",
                    "impressionType": "LIVE_SCAN_PLAIN",
                    "width": 1,
                    "height": 1,
                    "bitdepth": 1,
                    "resolution": 1,
                    "compression": "NONE"
                }
            ]
        }
    ]
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: updateResponse

`POST ${request.query.callback}`

Response callback

Query Parameters

- **`transactionId`** (*`string`*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

"OK"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "code": 1,
    "message": "string"
}
```

DELETE /v1/subjects/{subjectId}/{encounterId}

Delete one encounter

Parameters

- **subjectId** (*string*) – the id of the subject
- **encounterId** (*string*) – the id of the encounter

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 204 No Content – Delete successful
- 400 Bad Request – Bad request
- 403 Forbidden – Delete not allowed
- 404 Not Found – Unknown record
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: deleteResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.

- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

"OK"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

`POST /v1/subjects`

Insert one encounter and generate ID for both the subject and the encounter

Query Parameters

- **`transactionId`** (*string*) – The id of the transaction
- **`callback`** (*string*) – the callback address, where the result will be sent when available
- **`priority`** (*integer*) – the request priority
- **`algorithm`** (*string*) – Hint about the algorithm to be used

Status Codes

- 200 OK – Insertion successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Insertion not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/subjects HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": [
        {
            "encounterType": "string",
            "clientData": "c3RyaW5n",
            "contextualData": {
                "date": "2019-05-21"
            },
            "biographicData": {
                "dateOfBirth": "2019-05-21",
                "gender": "M",
```

(continues on next page)

(continued from previous page)

```
                "nationality": "string"
            },
            "biometricData": [
                {
                    "biometricType": "FACE",
                    "biometricSubType": "UNKNOWN",
                    "image": "c3RyaW5n",
                    "imageRef": "https://example.com",
                    "captureDate": "2019-05-21",
                    "captureDevice": "string",
                    "impressionType": "LIVE_SCAN_PLAIN",
                    "width": 1,
                    "height": 1,
                    "bitdepth": 1,
                    "resolution": 1,
                    "compression": "NONE"
                }
            ]
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "subjectId": "string",
    "encounterId": "string"
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: insertResponse

`POST ${request.query.callback}`

Response callback

Query Parameters

- **`transactionId`** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "status": "OK",
    "subjectId": "string",
    "encounterId": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/subjects/{subjectId}

Insert one encounter and generate its ID

Parameters

- **subjectId** (*string*) – the id of the subject

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **algorithm** (*string*) – Hint about the algorithm to be used

Status Codes

- 200 OK – Insertion successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Insertion not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/subjects/{subjectId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "galleries": [
        "string"
    ],
    "encounter": [
        {
            "encounterType": "string",
            "clientData": "c3RyaW5n",
            "contextualData": {
                "date": "2019-05-21"
            },
            "biographicData": {
```

(continues on next page)

(continued from previous page)

```
                "dateOfBirth": "2019-05-21",
                "gender": "M",
                "nationality": "string"
            },
            "biometricData": [
                {
                    "biometricType": "FACE",
                    "biometricSubType": "UNKNOWN",
                    "image": "c3RyaW5n",
                    "imageRef": "https://example.com",
                    "captureDate": "2019-05-21",
                    "captureDevice": "string",
                    "impressionType": "LIVE_SCAN_PLAIN",
                    "width": 1,
                    "height": 1,
                    "bitdepth": 1,
                    "resolution": 1,
                    "compression": "NONE"
                }
            ]
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "OK",
    "subjectId": "string",
    "encounterId": "string"
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: insertResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "status": "OK",
    "subjectId": "string",
    "encounterId": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

DELETE /v1/subjects/{subjectId}

Delete a subject and all its encounters

Parameters

- **subjectId** (*string*) – the id of the subject

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 204 No Content – Delete successful
- 400 Bad Request – Bad request
- 403 Forbidden – Delete not allowed
- 404 Not Found – Unknown record
- 500 Internal Server Error – Unexpected error

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: deleteResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

"OK"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Gallery

GET /v1/galleries

Get the ID of all the galleries

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 200 OK – Operation successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/galleries HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    "string"
]
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: getGalleriesResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
    "string"
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

GET /v1/galleries/{galleryId}

Get the content of one gallery

Parameters

- **galleryId** (*string*) – the id of the gallery

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority

Status Codes

- 200 OK – Operation successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 404 Not Found – Unknown record
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/galleries/{galleryId} HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "subjectId": "string",
        "encounterId": "string"
    }
]
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: getGalleryContentResponse

POST ${request.query.callback}

Response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
    {
        "subjectId": "string",
        "encounterId": "string"
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Search

POST /v1/identify/{galleryId}

Biometric identification

Identification based on biometric data from one gallery

Parameters

- **galleryId** (*string*) – the id of the gallery

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **maxNbCand** (*integer*) – the maximum number of candidates
- **threshold** (*number*) – the algorithm threshold
- **accuracyLevel** (*string*) – the accuracy level expected for this request

Status Codes

- 200 OK – Request executed. Identification result is returned.
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Identification not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/identify/{galleryId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "filter": {
        "dateOfBirthMin": "2019-05-21",
        "dateOfBirthMax": "2019-05-21"
    },
    "biometricData": [
        {
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN",
            "image": "c3RyaW5n",
            "imageRef": "https://example.com",
            "captureDate": "2019-05-21",
            "captureDevice": "string",
            "impressionType": "LIVE_SCAN_PLAIN",
            "width": 1,
            "height": 1,
            "bitdepth": 1,
            "resolution": 1,
            "compression": "NONE"
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "subjectId": "string",
        "rank": 1,
        "score": 1.0,
        "scoreList": [
            {
                "score": 1.0,
                "encounterId": "string",
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN"
            }
        ]
    }
]
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: identifyResponse

POST ${request.query.callback}
: **Identification response callback**

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
    {
        "subjectId": "string",
        "rank": 1,
        "score": 1.0,
        "scoreList": [
            {
                "score": 1.0,
                "encounterId": "string",
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN"
            }
        ]
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/identify/{galleryId}/{subjectId}

Biometric identification

Identification based on existing data from one gallery

Parameters

- **galleryId** (*string*) – the id of the gallery
- **subjectId** (*string*) – the id of the subject

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **maxNbCand** (*integer*) – the maximum number of candidates
- **threshold** (*number*) – the algorithm threshold
- **accuracyLevel** (*string*) – the accuracy level expected for this request

Status Codes

- 200 OK – Request executed. Identification result is returned.
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Identification not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/identify/{galleryId}/{subjectId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "dateOfBirthMin": "2019-05-21",
    "dateOfBirthMax": "2019-05-21"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "subjectId": "string",
        "rank": 1,
        "score": 1.0,
        "scoreList": [
            {
                "score": 1.0,
                "encounterId": "string",
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN"
            }
        ]
    }
]
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: identifyResponse

POST ${request.query.callback}

Identification response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
    {
        "subjectId": "string",
        "rank": 1,
        "score": 1.0,
        "scoreList": [
            {
                "score": 1.0,
                "encounterId": "string",
                "biometricType": "FACE",
                "biometricSubType": "UNKNOWN"
            }
        ]
    }
]
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

`POST /v1/verify/{galleryId}/{subjectId}`

Biometric verification

Verification of one set of biometric data and a record in the system

Parameters

- **`galleryId`** (*string*) – the id of the gallery
- **`subjectId`** (*string*) – the id of the subject

Query Parameters

- **`transactionId`** (*string*) – The id of the transaction
- **`callback`** (*string*) – the callback address, where the result will be sent when available
- **`priority`** (*integer*) – the request priority
- **`threshold`** (*number*) – the algorithm threshold
- **`accuracyLevel`** (*string*) – the accuracy level expected for this request

Status Codes

- 200 OK – Verification execution successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)

- 400 Bad Request – Bad request
- 404 Not Found – Unknown record
- 403 Forbidden – Verification not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/verify/{galleryId}/{subjectId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "biometricData": [
        {
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN",
            "image": "c3RyaW5n",
            "imageRef": "https://example.com",
            "captureDate": "2019-05-21",
            "captureDevice": "string",
            "impressionType": "LIVE_SCAN_PLAIN",
            "width": 1,
            "height": 1,
            "bitdepth": 1,
            "resolution": 1,
            "compression": "NONE"
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "decision": true,
    "scores": [
        {
            "score": 1.0,
            "encounterId": "string",
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN"
        }
    ]
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Callback: verifyResponse

POST ${request.query.callback}

Verification response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "decision": true,
    "scores": [
        {
            "score": 1.0,
            "encounterId": "string",
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN"
        }
    ]
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

POST /v1/verify

Biometric verification

Verification of two sets of biometric data

Query Parameters

- **transactionId** (*string*) – The id of the transaction
- **callback** (*string*) – the callback address, where the result will be sent when available
- **priority** (*integer*) – the request priority
- **threshold** (*number*) – the algorithm threshold
- **accuracyLevel** (*string*) – the accuracy level expected for this request

Status Codes

- 200 OK – Verification execution successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned (if no transaction ID was provided, one is generated by the server)
- 400 Bad Request – Bad request
- 403 Forbidden – Verification not allowed

- 500 Internal Server Error – Unexpected error

Example request:

```
POST /v1/verify HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "biometricData1": [
        {
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN",
            "image": "c3RyaW5n",
            "imageRef": "https://example.com",
            "captureDate": "2019-05-21",
            "captureDevice": "string",
            "impressionType": "LIVE_SCAN_PLAIN",
            "width": 1,
            "height": 1,
            "bitdepth": 1,
            "resolution": 1,
            "compression": "NONE"
        }
    ],
    "biometricData2": [
        {
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN",
            "image": "c3RyaW5n",
            "imageRef": "https://example.com",
            "captureDate": "2019-05-21",
            "captureDevice": "string",
            "impressionType": "LIVE_SCAN_PLAIN",
            "width": 1,
            "height": 1,
            "bitdepth": 1,
            "resolution": 1,
            "compression": "NONE"
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "decision": true,
    "scores": [
        {
            "score": 1.0,
            "encounterId": "string",
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN"
        }
    ]
}
```

Example response:

```
HTTP/1.1 202 Accepted
Content-Type: application/json

"123e4567-e89b-12d3-a456-426655440000"
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "code": 1,
    "message": "string"
}
```

Callback: verifyResponse

POST ${request.query.callback}

Verification response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)

Status Codes

- 204 No Content – Response is received and accepted.
- 403 Forbidden – Forbidden access to the service
- 500 Internal Server Error – Unexpected error

Example request:

```
POST ${request.query.callback}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "decision": true,
    "scores": [
        {
            "score": 1.0,
            "encounterId": "string",
            "biometricType": "FACE",
            "biometricSubType": "UNKNOWN"
        }
    ]
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
    "code": 1,
    "message": "string"
}
```

Data Model

To be completed

7.3.5 Third Party Services

Services

To be defined

List of Tables

List of Figures

/${request.query.callback}

/v1

/{$request.query.address}

Index