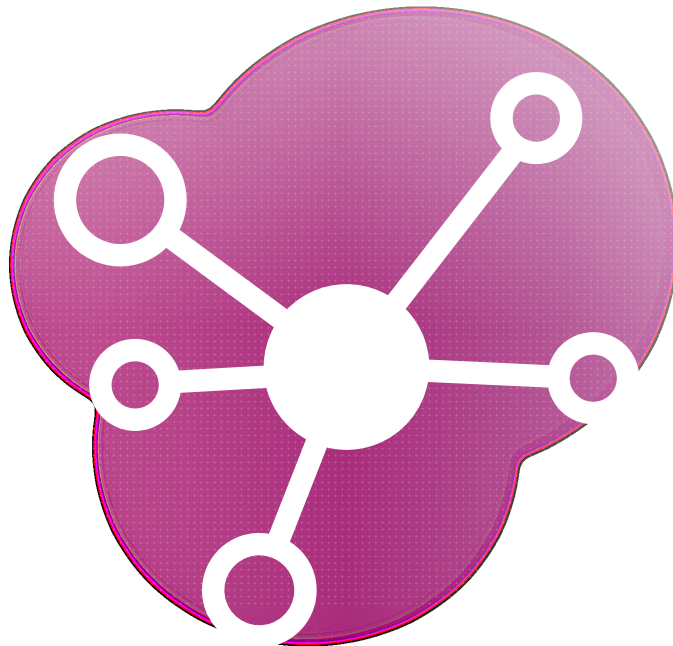


**OSIA**

## **Specifications version 3.0.0**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement: vendor lock-in	1
1.2	The OSIA Initiative	2
1.3	Diffusion, Audience, and Access	3
1.4	Document Overview	3
1.5	Convention and Typographical Rules	3
1.6	Revision History	4
<b>2</b>	<b>Functional View</b>	<b>5</b>
2.1	Components: Standardized Definition and Scope	5
2.2	Interfaces	7
2.3	Components vs Interfaces Mapping	10
2.4	Use Cases - How to Use OSIA	11
2.4.1	Birth Use Case	12
2.4.2	Death Use Case	13
2.4.3	Marriage Use Case	13
2.4.4	Deduplication Use Case	13
2.4.5	ID Card Request Use Case	13
2.4.6	Bank account opening Use Case	14
2.4.7	Police identity control Use Case	14
<b>3</b>	<b>Security &amp; Privacy</b>	<b>15</b>
3.1	Introduction	15
3.2	Virtual UIN	15
3.3	Authorization	15
3.4	GDPR	15
<b>4</b>	<b>OSIA Versions &amp; Referencing</b>	<b>16</b>
<b>5</b>	<b>Interfaces</b>	<b>17</b>
5.1	Notification	17
5.1.1	Services	18
5.1.2	Dictionaries	19
5.2	Data Access	19
5.2.1	Services	19
5.2.2	Dictionaries	22
5.3	UIN Management	23
5.3.1	Services	23
5.4	Enrollment Services	24
5.4.1	Services	24
5.4.2	Filter	25

5.4.3	Transaction ID	25
5.4.4	Data Model	26
5.5	Population Registry Services	26
5.5.1	Services	26
5.5.2	Data Model	30
5.6	Biometrics	31
5.6.1	Services	32
5.6.2	Options	36
5.6.3	Data Model	36
5.7	Credential Services	37
5.7.1	Services	37
5.8	ID Usage	39
5.8.1	Services	39
5.9	Under discussion	40
5.9.1	Services	40
5.9.2	Filter	44
5.9.3	Transaction ID	44
5.9.4	Data Model	45
<b>6</b>	<b>Components</b>	<b>46</b>
6.1	Enrollment Component	46
6.1.1	Enrollment Services	46
6.2	Population Registry	48
6.2.1	Notification	48
6.2.2	Data Access	50
6.2.3	Population Registry Services	54
6.3	Civil Registry	60
6.3.1	Notification	60
6.3.2	Data Access	62
6.4	UIN Generator	66
6.4.1	UIN Management	66
6.5	ABIS	67
6.5.1	Biometrics	67
6.6	Credential Management System	73
6.6.1	Credential Services	73
6.7	Third Party Services	75
6.7.1	ID Usage	75
<b>7</b>	<b>Annexes</b>	<b>77</b>
7.1	Glossary	77
7.2	Data Format	78
7.3	Technical Specifications	78
7.3.1	Notification	78
7.3.2	UIN Management	86
7.3.3	Data Access	87
7.3.4	Population Registry Management	92
7.3.5	Biometrics	106
7.3.6	Third Party Services	137
	<b>HTTP Routing Table</b>	<b>140</b>
	<b>Index</b>	<b>141</b>

### 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 vendor 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. Main 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.

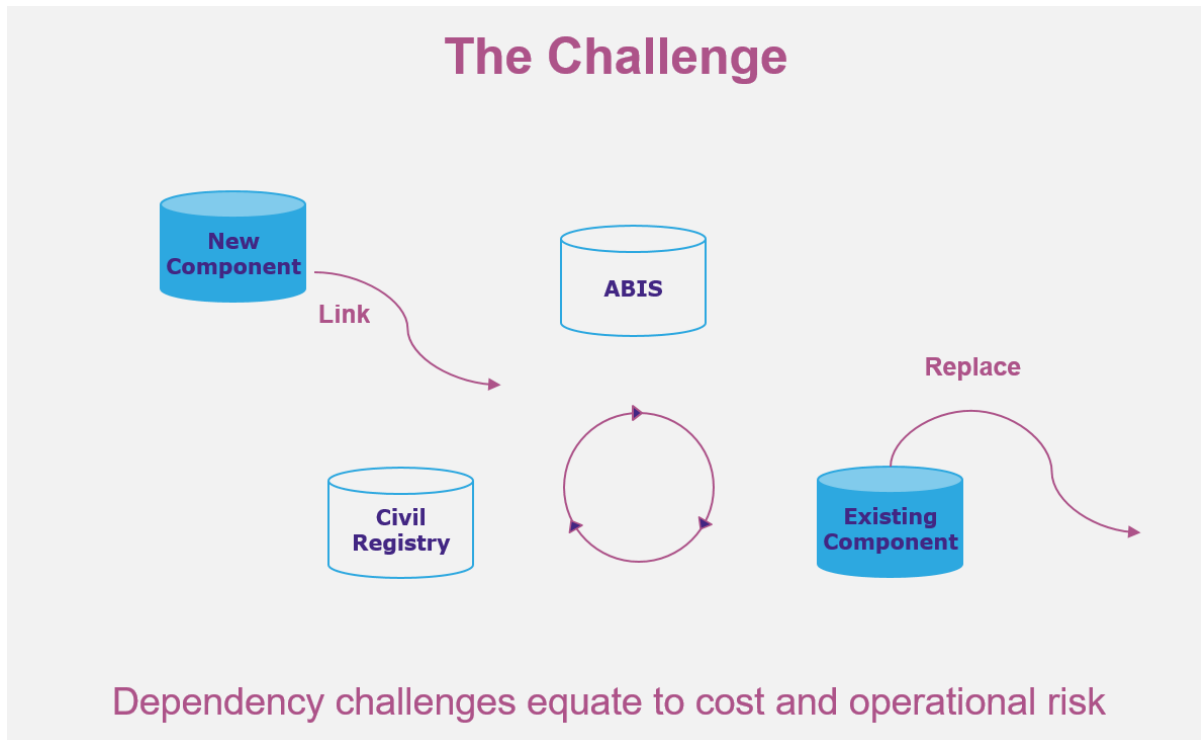


Fig. 1.1: The dependency challenge

## 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](#).

This specification is licensed under [The MIT License](#).

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 interfaces 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: This chapter introduces the problem statement and the OSIA initiative.
- Chapter 2 Functional View: This chapter provides an overview of OSIA interfaces and how they can be mapped against the various identity ecosystem components. Finally, the chapter describes a series of use cases where different OSIA interfaces are implemented between multiple identity ecosystem components.
- Chapter 3 Security and Privacy: This chapter lists a set of Privacy and Security features embedded in OSIA interfaces specifications.
- Chapter 4 OSIA Versions and Referencing: This chapter describes the way OSIA interfaces can be referenced in documents and tenders.
- Chapter 5 Interfaces: This chapter describes the specifications of all OSIA interfaces.
- Chapter 6 Components: This chapter describes OSIA interfaces that each component of the identity ecosystem may implement.

## 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.

---

<b>Warning:</b> Indicates that the specific condition or procedure must be respected.
---

## 1.6 Revision History

Version	Date	Notes
1.0.0	2018-12	First release
3.0.0	2019-06	Second release

## 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 *Enrollment* 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.”<sup>1</sup>

- 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 in accordance with the legal requirement in each country. Civil registration is carried out primarily

---

<sup>1</sup> *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.*



for the purpose of establishing the documents provided by the law.”<sup>2</sup>

- The *Credential Management System* (CMS) component.

CMS is defined as a system to manage the production and issuance of credentials such as ID Cards, passports, driving licenses, digital ID, etc.

- The *Third Party Services* component.

TBD

Table 2.1: Components

ID Ecosystem Component	Data	Functions
Enrollment	<ul style="list-style-type: none"> <li>• Alpha</li> <li>• UIN</li> <li>• History</li> <li>• Supporting documents</li> </ul>	<ul style="list-style-type: none"> <li>• Recording application</li> <li>• Collecting personal data</li> </ul>
PR	<ul style="list-style-type: none"> <li>• Alpha</li> <li>• UIN</li> <li>• History</li> <li>• Supporting documents</li> </ul>	<ul style="list-style-type: none"> <li>• Identity attributes storage</li> <li>• Identity Life cycle management</li> </ul>
UIN Gen	<ul style="list-style-type: none"> <li>• Alpha</li> <li>• UIN</li> </ul>	<ul style="list-style-type: none"> <li>• UIN generation</li> </ul>
ABIS	<ul style="list-style-type: none"> <li>• UIN</li> <li>• Biometric data (images and templates)</li> </ul>	<ul style="list-style-type: none"> <li>• Authentication (1:1)</li> <li>• Identification (1:N)</li> <li>• Quality control and adjudication</li> </ul>
CR	<ul style="list-style-type: none"> <li>• Events</li> <li>• UIN</li> <li>• History</li> <li>• Supporting documents</li> </ul>	<ul style="list-style-type: none"> <li>• Events storage</li> <li>• Certificate production</li> <li>• Workflow</li> </ul>
CMS	<ul style="list-style-type: none"> <li>• Alpha</li> <li>• UIN</li> <li>• History</li> <li>• Supporting documents</li> </ul>	<ul style="list-style-type: none"> <li>• Credential data storage</li> <li>• Credential Life cycle management</li> <li>• Credential Production</li> <li>• Workflow</li> <li>• SMS and email server</li> </ul>
Third Party Services	TBD	KYC/auth

The components are represented on the following diagram:

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

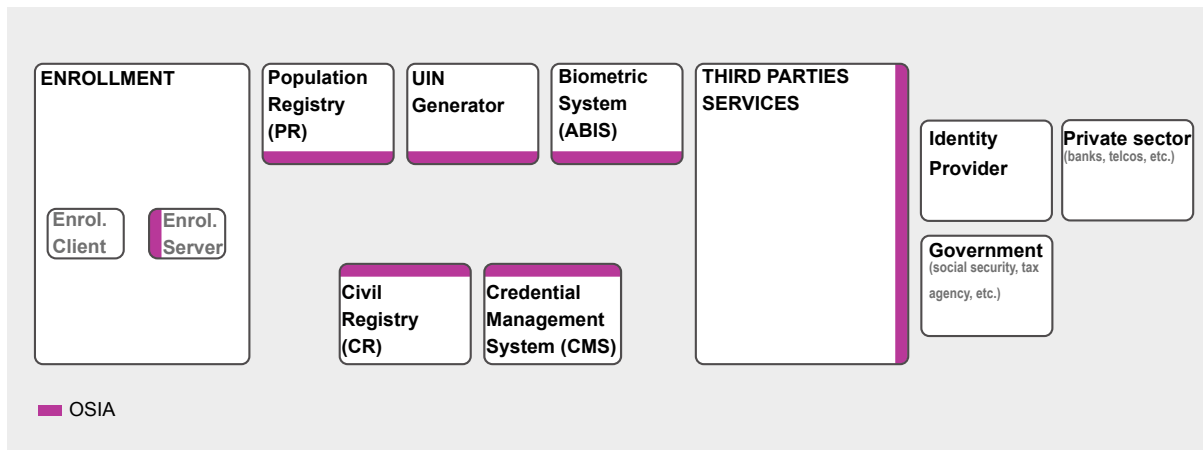


Fig. 2.1: Components identified as part of the identity ecosystem

## 2.2 Interfaces

This chapter describes the following interfaces:

- Notification

A set of services to manage notifications for different types of events as for instance birth and death.

- 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. Please note that the UIN is the same throughout all registries (see Chapter 3 - Security & Privacy).
2. The registries (civil, population, or other) are considered as centralized systems that are connected. If one registry is architected 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.

- UIN Management

A set of services to manage the unique identifier.

- Enrollment Services

A set of services to manage biographic and biometric data upon collection.

- Population Registry Services

A set of services to manage a registry of the population.

- Biometrics

A set of services to manage biometric data and databases.

- Credential Services

A set of services to manage credentials, physical and digital.

- ID Usage

A set of services implemented on top of identity systems to favour third parties consumption of identity data.

- Under discussion

A set of services under discussion and not yet linked to any specific tag.

The following table describes in detail the interfaces and associated services.

Table 2.2: Interfaces List

Services	Description
<b>Notification</b>	
Subscribe	Subscribe a URL to receive notifications sent to one topic
Unsubscribe	Unsubscribe a URL from the list of receiver for one topic
Confirm	Confirm that the URL used during the subscription is valid
Publish	Notify of a new event all systems that subscribed to this topic
<b>Data Access</b>	
Read Person Attributes	Read person attributes
Match Person Attributes	Check the value of attributes without exposing private data
Verify Person Attributes	Evaluate simple expressions on person's attributes without exposing private data
Query Person UIN	Query the persons by a set of attributes, used when the UIN is unknown
Query Person List	Query the persons by a list of attributes and their values
Read document	Read in a selected format (PDF, image, etc.) a document such as a marriage certificate
<b>UIN Management</b>	
Generate UIN	Generate a new UIN
<b>Enrollment Services</b>	
Create Person	Insert a new person
Read Person	Retrieve the attributes of a person
Update Person	Update a person
Delete Person	Delete a person
Find People	Retrieve a list of people who match passed in search criteria
<b>Population Registry Services</b>	
Create Person	Create a new person
Read Person	Read the attributes of a person
Update Person	Update a person
Delete Person	Delete a person and all its identities
Create Identity	Create a new identity in a person
Read Identity	Read one or all the identities of one person
Update Identity	Update an identity. An identity can be updated only in the status claimed
Partial Update Identity	Update part of an identity. Not all attributes are mandatory.
Delete Identity	Delete an identity
Set Identity Status	Set an identity status
Define Reference	Define the reference identity of one person
Read Reference	Read the reference identity of one person
Read Galleries	Read the ID of all the galleries
Read Gallery Content	Read the content of one gallery, i.e. the IDs of all the records linked to this gallery
<b>Biometrics</b>	
Create	Create a new encounter. No identify is performed
Read	Read the data of an encounter

Continued on next page

Table 2.2 – continued from previous page

Update	Update an encounter
Delete	Delete an encounter
Read Template	Read the generated template
Read Galleries	Read the ID of all the galleries
Read Gallery content	Read the content of one gallery, i.e. the IDs of all the records linked to this gallery
Identify	Identify a person using biometrics data and filters on biographic or contextual data
Verify	Verify an identity using biometrics data
<b>Credential Services</b>	
Create Credential	Request issuance of a secure document / credential
Read Credential Issuance	Retrieve the data/status of an issuance
Update Credential	Update the requested issuance of a secure document / credential
Delete Credential	Delete/cancel the requested issuance of a secure document / credential
Read Credential	Retrieve the attributes/status of an issued credential (smart card, mobile, passport, etc.)
Suspend Credential	Suspend an issued credential. For electronic credentials this will suspend any PKI certificates that are present
Unsuspend Credential	Unsuspend an issued credential. For electronic credentials this will unsuspend any PKI certificates that are present
Cancel Credential	Cancel an issued credential. For electronic credentials this will revoke any PKI certificates that are present
<b>ID Usage</b>	
Verify ID	Verify Identity based on UIN and set of attributes (biometric data, demographics, credential)
Identify	Identify a person based on a set of attributes (biometric data, demographics, credential)
Read Attributes	Read person attributes
Read Attributes set	Read person attributes corresponding to a predefined set name
<b>Under discussion</b>	
List Credential Profiles	Retrieve the list of credential profiles
Read Credential Profiles	Retrieve the credential profile
Create Document	Add a new document for a person
Read Document	Retrieve document data
Update Document	Update a document for a person
Delete Document	Delete a document for a person
Update Document Val Status	Updates the status of a document validation
Read Document Val Status	Retrieve the status of a document validation
Create Biometric	Add a new biometric for a person
Read Biometric Metadata	Retrieve biometric data
Update Biometric	Update a biometric for a person
Delete Biometric	Delete a biometric for a person
Update Biometric Val Status	Updates the status of a biometric validation
Read Biometric Val Status	Retrieve the status of a biometric validation
Create Biographic	Add a new biographic for a person
Read Biographic	Retrieve biographic data
Update Biographic	Update a biographic for a person
Delete Biographic	Delete a biographic for a person
Update Biographic Val Status	Updates the status of a biographic validation
Read Biographic Val Status	Retrieve the status of a biographic validation

## 2.3 Components vs Interfaces Mapping

The interfaces described in the following chapter can be mapped against ID ecosystem components as per the table below:

Table 2.3: Components vs Interfaces Mapping

Interfaces	Components						
	Enroll	PR	UIN Gen	ABIS	CR	CMS	3rd PS
<b>Notification</b>							
Subscribe		U		U	U	U	
Unsubscribe		U		U	U	U	
Confirm							
Publish		I		I	I	I	
<b>Data Access</b>							
Read Person Attributes	U	IU		U	IU		U
Match Person Attributes	U	IU			IU		U
Verify Person Attributes	U	IU			IU		U
Query Person UIN	U	IU			IU		
Query Person List					U		
Read Document	U	IU			IU		
<b>UIN Management</b>							
Generate UIN		U	I		U		
<b>Enrollment Services</b>							
Create Person	I						
Read Person	I						
Update Person	I						
Delete Person	I						
Find People	I						
<b>Population Registry Services</b>							
Create Person		I		I		U	
Read Person		I		I		U	U
Update Person		I		I		U	
Delete Person		I		I		U	
Create Identity		I					
Read Identity		I					
Update Identity		I					
Partial Update Identity		I					
Delete Identity		I					
Set Identity Status		I					
Define Reference		I					
Read Reference		I					
Read Galleries		I					
Read Gallery Content		I					
<b>Biometrics</b>							
Create	U	U		I			
Read	U	U		I			U
Update	U	U		I			
Delete	U	U		I			
Read Template	U	U		I			
Read Galleries							
Read Gallery Content	U	U		I			
Identify	U			I			U
Verify	U			I			U

Continued on next page

Table 2.3 – continued from previous page

Interfaces	Components						
	Enroll	PR	UIN Gen	ABIS	CR	CMS	3rd PS
<b>Credential Services</b>							
Create Credential							
Read Credential Issuance							
Update Credential							
Delete Credential							
Read Credential							
Suspend Credential							
Unsuspend Credential							
Cancel Credential							
<b>ID Usage</b>							
Verify ID							I
Identify ID							I
Read Attributes							I
Read Attributes set							I
<b>Under discussion</b>							
List Cred Profiles							
Read Cred Profiles							
Create Document							
Read Document							
Update Document							
Delete Document							
Update Document Val Status							
Read Document Val Status							
Create Biometric							
Read Biometric Metadata							
Update Biometric							
Delete Biometric							
Update Biometric Val Status							
Read Biometric Val Status							
Create Biographic							
Read Biographic							
Update Biographic							
Delete Biographic							
Update Biographic Val Status							
Read Biographic Val Status							

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

Below are a set of examples of how OSIA interfaces could be implemented in various use cases.

## 2.4.1 Birth Use Case

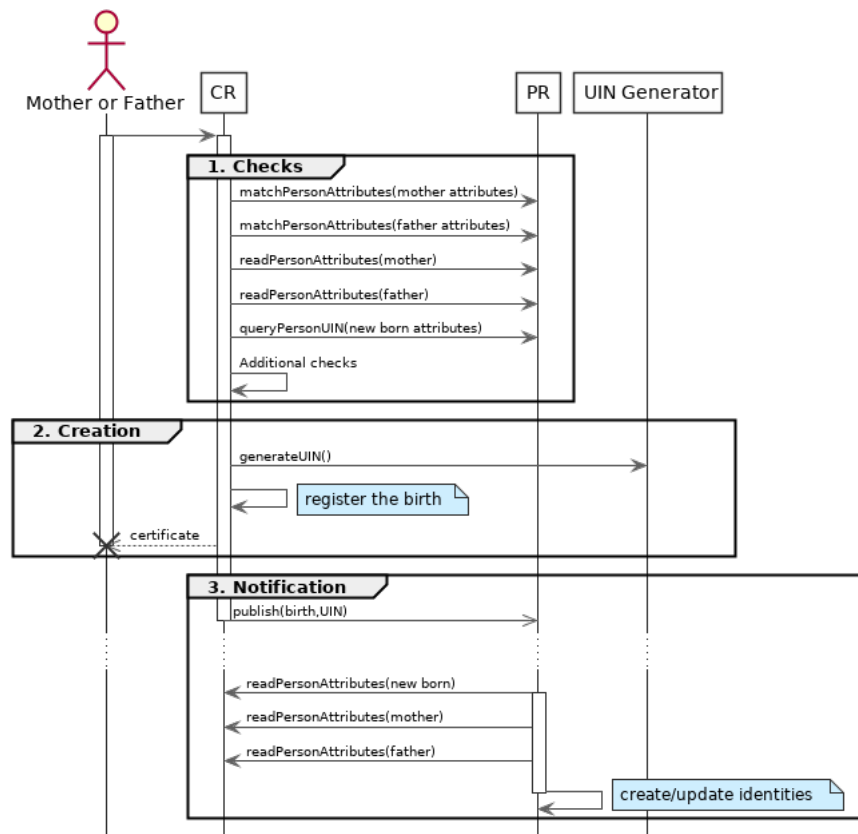


Fig. 2.2: Birth Use Case

### 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
- `readPersonAttributes`: to get missing data about the parents's identity
- `queryPersonUIN`: 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 first step after the checks is to generate a new UIN. To do so, the CR requests a new UIN to the PR using `generateUIN` service. At this point the birth registration takes place. How the CR will process the birth registration is specific to each CR implementation and not in the scope of this document.

### 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:

- `publish`: 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:

- `readPersonAttributes`: to get the attributes of interest to the PR for the parents if relevant 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 Use Case

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.

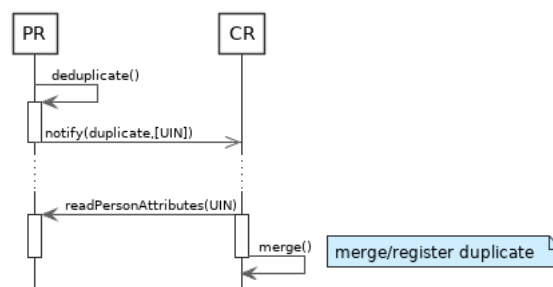


Fig. 2.3: Deduplication Use Case

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

## 2.4.5 ID Card Request Use Case

To be completed



## 2.4.6 Bank account opening Use Case

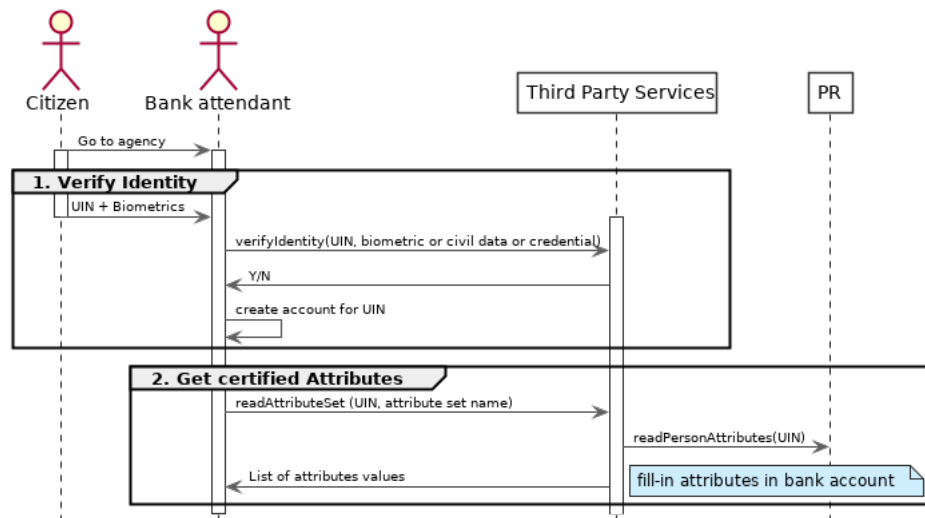


Fig. 2.4: Bank account opening Use Case

## 2.4.7 Police identity control Use Case

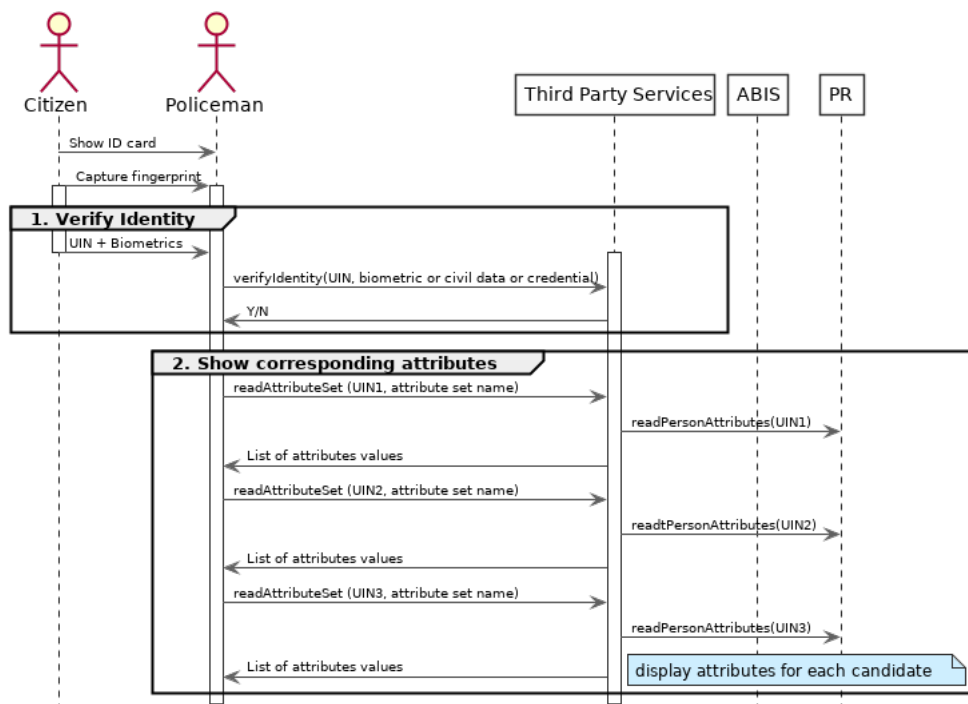


Fig. 2.5: Collaborative identity control

#### 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

---

## 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. 1.0.0
```

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

- OSIA - Notification - v. 3.0.0
- OSIA - Data Access - v. 3.0.0
- OSIA - UIN Management - v. 3.0.0
- OSIA - Enrollment Services - v. 3.0.0
- OSIA - Identity Management - v. 3.0.0
- OSIA - Population Registry Services - v. 3.0.0
- OSIA - Biometrics - v. 3.0.0
- OSIA - Credential Services - v. 3.0.0
- OSIA - ID Usage - v. 3.0.0

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. 1.0.0
- OSIA - Data Access - v. 1.0.0

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

```
OSIA - CR v. 1.0.0
```

The chapter below describes the specifications of all OSIA interfaces and related services.

### 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:

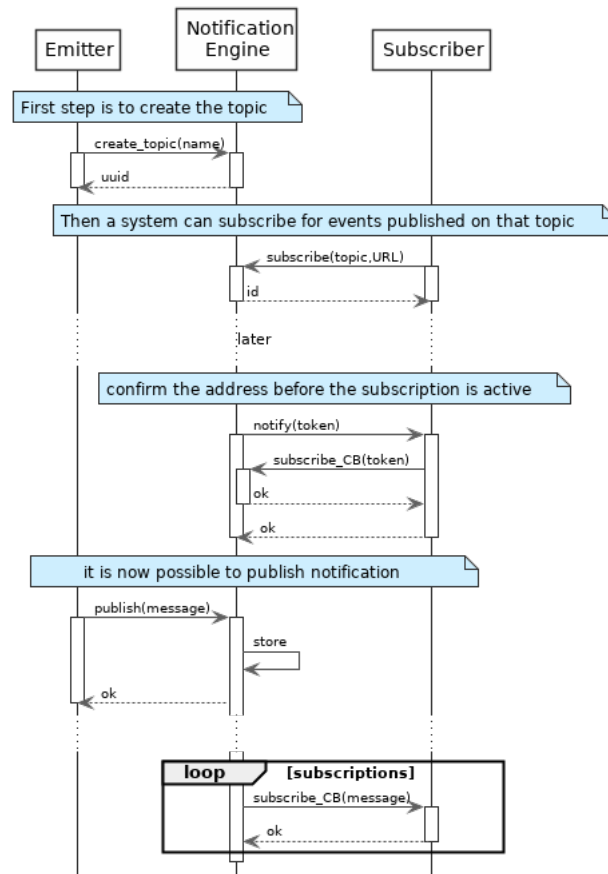


Fig. 5.1: Subscription &amp; Notification Process

### 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	✓	
Foetal 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

**readPersonAttributes** (*UIN*, *names*)

Read 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.

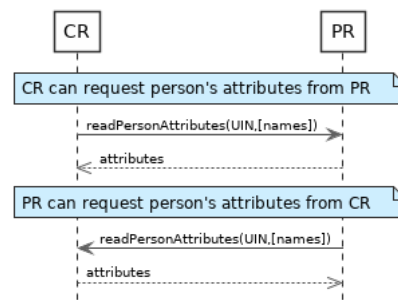


Fig. 5.2: readPersonAttributes Sequence Diagram

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

Match person attributes. This service is used to check the value of attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

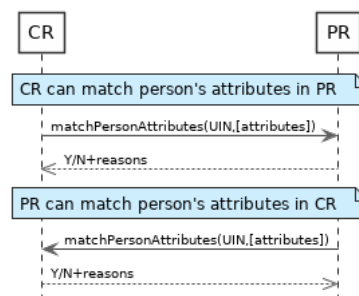


Fig. 5.3: matchPersonAttributes Sequence Diagram

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

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

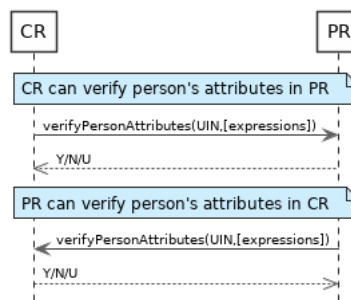


Fig. 5.4: verifyPersonAttributes Sequence Diagram

#### queryPersonUIN (attributes)

Query the persons by a set of attributes. This service is used when the UIN is unknown. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

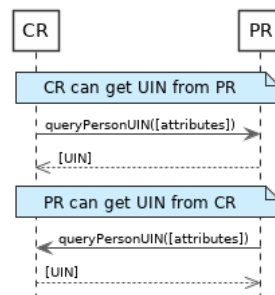


Fig. 5.5: queryPersonUIN Sequence Diagram

#### queryPersonList (attributes, names)

Query the persons by a list of attributes and their values. This service is proposed as an optimization of a sequence of calls to queryPersonUIN() and readPersonAttributes().

**Authorization:** **To be defined**

##### Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find the persons.  
 Each attribute is described with its name and its value
- **names** (*list[str]*) – The names of the attributes requested

**Returns** a list of lists of pairs (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.

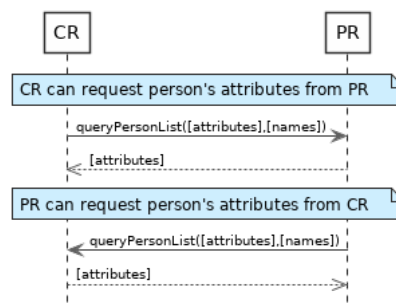


Fig. 5.6: queryPersonList Sequence Diagram

**readDocument** (*UINs, documentType, format*)

Read 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.

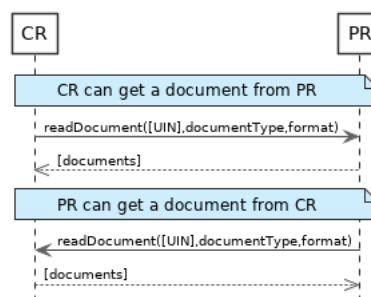


Fig. 5.7: readDocument Sequence Diagram

## 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

**generateUIN** (*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.

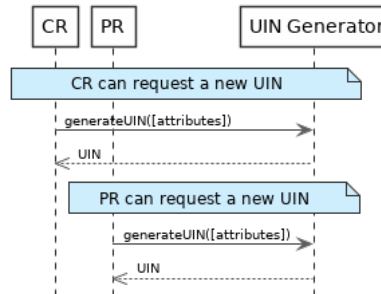


Fig. 5.8: generateUIN Sequence Diagram

## 5.4 Enrollment Services

### 5.4.1 Services

**createPerson** (*personID*, *personData*, *transactionID*)

Insert a new person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person. If the person already exists for the ID an error is returned.
- **personData** (*dict*) – The person attributes.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**readPerson** (*personID*, *filter*, *transactionID*)

Retrieve the attributes of a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error and in case of success the person data.

**updatePerson** (*personID*, *personData*, *transactionID*)

Update a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personData** (*dict*) – The person data, this can be partial data.

- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**deletePerson** (*personID*, *transactionID*)

Delete a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**findPeople** (*filter*, *transactionID*)

Retrieve a list of people who match passed in search criteria.

**Authorization:** To be defined

**Parameters**

- **filter** (*dict*) – The search criteria to match on.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error and in case of success the matching person list.

---

## 5.4.2 Filter

The “filter” parameter used in “read” calls is used to provide a set of identifiers that limit the amount of data that is returned. It is often the case that the whole data set is not required, but instead, a subset of that data. Where possible, existing standards based identifiers should be used for the attributes to retrieve.

E.g. For surname/familyname, use OID 2.5.4.4 or id-at-surname.

Some calls may require new filter attributes to be defined. E.g. when retrieving biometric data, the caller may only want the meta data about that biometric, rather than the actual biometric data.

## 5.4.3 Transaction ID

The “transactionID” is 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.

## 5.4.4 Data Model

Table 5.7: Enrolment Data Model

Type	Description	Example
Person	Person who is known to an identity assurance system.	TBD
Document Data	a dictionary (list of names and values) giving the document data of interest for the document services.	TBD
Biometric Data	Digital representation of biometric characteristics. 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.	fingerprint, portrait, iris
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biographic services.	TBD

## 5.5 Population Registry Services

This interface describes services to manage a registry of the population in the context of an identity system. It is based on the following principles:

- It supports a history of identities, meaning that a person has one identity and this identity has a history.
- Images can be passed by value or reference. When passed by value, they are base64-encoded.
- Existing standards are used whenever possible.
- This interface is complementary to the data access interface. The data access interface is used to query the persons and uses the reference identity to return attributes.
- The population registry can store the biometric data or can rely on the ABIS subsystem to do it. The preferred solution, for a clean separation of data of different nature and by application of GDPR principles, is to put the biometric data only in the ABIS. Yet many existing systems store biometric data with the biographic data and this specification gives the flexibility to do it.

See *Population Registry Management* for the technical details of this interface.

### 5.5.1 Services

**createPerson** (*personID*, *personData*, *transactionID*)

Create a new person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person. If the person already exists for the ID an error is returned.
- **personData** – The person attributes.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**readPerson** (*personID*, *transactionID*)

Read the attributes of a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error and in case of success the person data.

**updatePerson** (*personID*, *personData*, *transactionID*)

Update a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personData** (*dict*) – The person data.

**Returns** a status indicating success or error.

**deletePerson** (*personID*, *transactionID*)

Delete a person and all its identities.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**createIdentity** (*personID*, *identityID*, *identity*, *transactionID*)

Create a new identity in a person. If no identityID is provided, a new one is generated. If identityID is provided, it is checked for uniqueness and used for the identity if unique. An error is returned if the provided identityID is not unique.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – The ID of the identity.
- **identity** – The new identity data.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**readIdentity** (*personID*, *identityID*, *transactionID*)

Read one or all the identities of one person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **identityID** – The ID of the identity. If not provided, all identities are returned.

- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error, and in case of success a list of identities.

**updateIdentity** (*personID, identityID, identity, transactionID*)

Update an identity. An identity can be updated only in the status `claimed`.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **identity** – The identity data.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**partialUpdateIdentity** (*personID, identityID, identity, transactionID*)

Update part of an identity. Not all attributes are mandatory. The payload is defined as per [RFC 7396](#). An identity can be updated only in the status `claimed`.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **identity** – Part of the identity data.

**Returns** a status indicating success or error.

**deleteIdentity** (*personID, identityID, transactionID*)

Delete an identity.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**setIdentityStatus** (*personID, identityID, status, transactionID*)

Set an identity status.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **status** (*str*) – The new status of the identity.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**defineReference** (*personID*, *identityID*, *transactionID*)

Define the reference identity of one person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity being now the reference.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**readReference** (*personID*, *transactionID*)

Read the reference identity of one person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error and in case of success the reference identity.

**readGalleries** (*transactionID*)

Read the ID of all the galleries.

**Authorization:** To be defined

**Parameters** **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error, and in case of success a list of gallery ID.

**readGalleryContent** (*galleryID*, *transactionID*)

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

**Authorization:** To be defined

**Parameters**

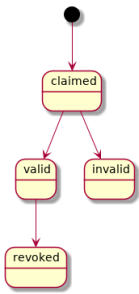
- **galleryID** (*str*) – Gallery whose content will be returned.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error. In case of success a list of person/identity IDs.



## 5.5.2 Data Model

Table 5.8: Population Registry Data Model

Type	Description	Example
Gallery	A group of persons related by a common purpose, designation, or status. A person can belong to multiple galleries.	VIP, Wanted, etc.
Person	<p>Person who is known to an identity assurance system.</p> <p>A person record has:</p> <ul style="list-style-type: none"> <li>• a status, such as <code>active</code> or <code>inactive</code>, defining the status of the record (the record can be excluded from queries based on this status),</li> <li>• a physical status, such as <code>alive</code> or <code>dead</code>, defining the status of the person,</li> <li>• a set of identities, keeping track of all identity data submitted by the person during the life of the system,</li> <li>• a reference identity, i.e. a consolidated view of all the identities defining the current correct identity of the person. It corresponds usually to the last valid identity but it can also include data from previous identities.</li> </ul>	N/A
Identity	<p>The attributes describing an identity of a person.</p> <p>An identity has a status such as: <code>claimed</code> (identity not yet validated), <code>valid</code> (the identity is valid), <code>invalid</code> (the identity is not valid), <code>revoked</code> (the identity cannot be used any longer).</p> <p>An identity can be updated only in the status <code>claimed</code>.</p> <p>The allowed transitions for the status are represented below:</p>  <pre> graph TD     Start(( )) --&gt; claimed[claimed]     claimed --&gt; valid[valid]     claimed --&gt; invalid[invalid]     valid --&gt; revoked[revoked]   </pre> <p>The attributes are separated into two categories: the biographic data and the contextual data.</p>	N/A
Biographic Data	A dictionary (list of names and values) giving the biographic data of the identity	<code>firstName</code> , <code>lastName</code> , <code>dateOfBirth</code> , etc.
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	<code>operatorName</code> , <code>enrolmentDate</code> , etc.
Biometric Data	Digital representation of biometric characteristics. 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.	<code>fingerprint</code> , <code>portrait</code> , <code>iris</code>
Document	The document data (images) attached to the identity and used to validate it.	Birth certificate, invoice

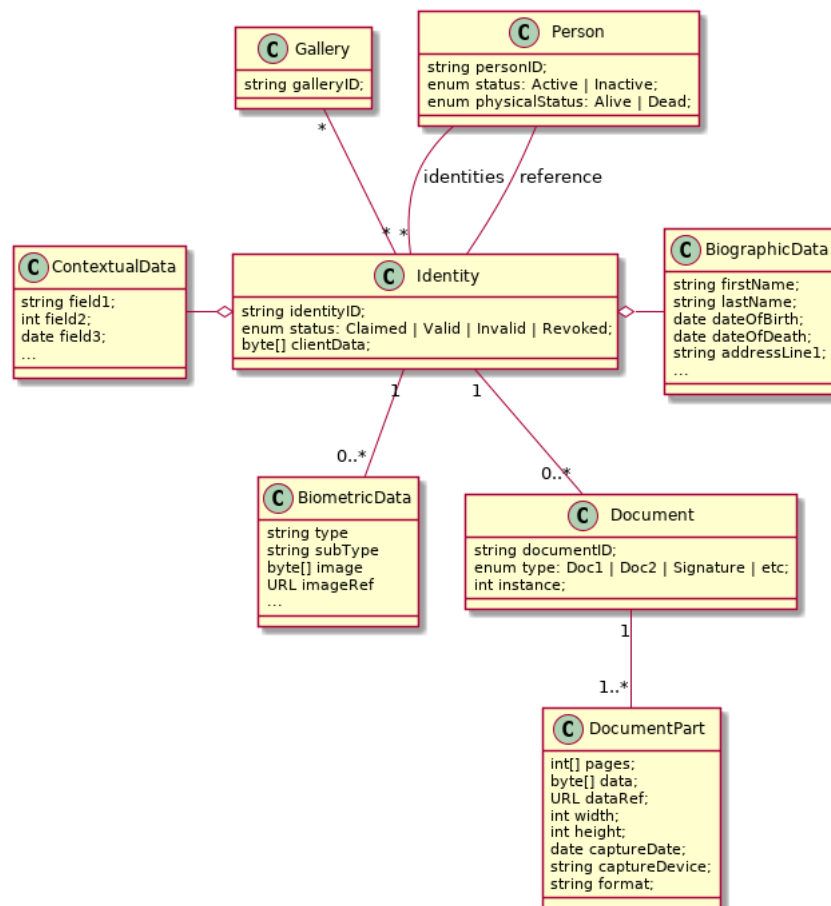


Fig. 5.9: Population Registry Data Model

## 5.6 Biometrics

This interface describes 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 credentials 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 preferred image format for biometric data is ISO-19794.

### About 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.

### 5.6.1 Services

**create** (*personID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *clientData*, *callback*, *transactionID*, *options*)

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

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The person ID. This is optional and will be generated if not provided
- **encounterID** (*str*) – The encounter ID. This is optional and will be generated if not provided
- **galleryID** (*list(str)*) – the gallery ID to which this encounter belongs. A minimum of one gallery must be provided
- **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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are *priority*, *algorithm*.

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

**read** (*personID*, *encounterID*, *callback*, *transactionID*, *options*)

Read the data of an encounter.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the person are returned.
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are *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** (*personID, encounterID, galleryID, biographicData, contextualData, biometricData, callback, transactionID, options*)  
Update an encounter.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs. A minimum of one gallery must be provided
- **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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`, `algorithm`.

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

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

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the person are deleted.
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**readTemplate** (*personID, encounterID, biometricType, biometricSubType, callback, transactionID, options*)  
Read the generated template.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID.
- **biometricType** (*str*) – The type of biometrics to consider (optional)

- **biometricSubType** (*str*) – The subtype of biometrics to consider (optional)
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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.

**readGalleries** (*callback, transactionID, options*)

Read the ID of all the galleries.

**Authorization:** To be defined

#### Parameters

- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**readGalleryContent** (*galleryID, callback, transactionID, options*)

Read 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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**identify** (*galleryID, filter, biometricData, callback, transactionID, options*)

Identify a person 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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *personID*, *callback*, *transactionID*, *options*)

Identify a person using biometrics data of a person 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)
- **personID** – the person ID
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *personID*, *biometricData*, *callback*, *transactionID*, *options*)

Verify an identity using biometrics data.

**Authorization:** To be defined

#### Parameters

- **galleryID** (*str*) – Search only in this gallery. If the person does not belong to this gallery, an error is returned.
- **personID** (*str*) – The person ID
- **biometricData** – The biometric data
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *transactionID*, *options*)

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

**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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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.

## 5.6.2 Options

Table 5.9: Biometric Services Options

Name	Description
<code>priority</code>	Priority of the request. Values range from 0 to 9
<code>maxNbCand</code>	The maximum number of candidates to return.
<code>threshold</code>	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
<code>algorithm</code>	Specify the type of algorithm to be used.
<code>accuracyLevel</code>	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.).

## 5.6.3 Data Model

Table 5.10: Biometric Data Model

Type	Description	Example
Gallery	A group of persons related by a common purpose, designation, or status. A person can belong to multiple galleries.	TBD
Person	Person who is known to an identity assurance system.	TBD
Encounter	Event in which the client application interacts with a person resulting in data being collected during or about the encounter. An encounter is characterized by an <i>identifier</i> and a <i>type</i> (also called <i>purpose</i> 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 <i>range</i> 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. 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.	fingerprint, portrait, iris
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

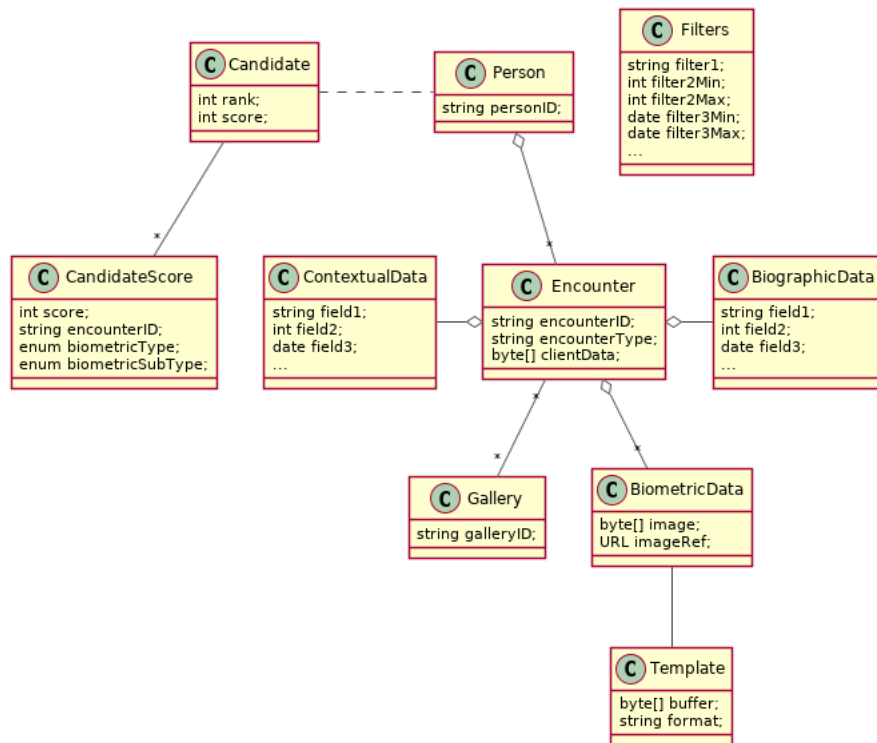


Fig. 5.10: Biometric Data Model

## 5.7 Credential Services

### 5.7.1 Services

**createCredential** (*personID*, *credentialProfileID*, *additionalData*, *transactionID*)

Request issuance of a secure document / credential.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **credentialProfileID** (*str*) – The ID of the credential profile to issue to the person.
- **additionalData** (*dict*) – Additional data relating to the requested credential profile, e.g. credential lifetime if overriding default, delivery addresses, etc.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error. In the case of success, an issuance identifier.

**readCredentialIssuance** (*issuanceID*, *filter*, *transactionID*)

Retrieve the data/status of an issuance.

**Authorization:** To be defined

#### Parameters

- **issuanceID** (*str*) – The ID of the issuance.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The client generated transactionID.



**Returns** a status indicating success or error, and in case of success the issuance data/status.

**updateCredential** (*issuanceID*, *additionalData*, *transactionID*)

Update the requested issuance of a secure document / credential.

**Authorization:** To be defined

**Parameters**

- **issuanceID** (*str*) – The ID of the issuance.
- **transactionID** (*string*) – The client generated transactionID.
- **additionalData** (*dict*) – Additional data relating to the requested credential profile, e.g. credential lifetime if overriding default, delivery addresses, etc.

**Returns** a status indicating success or error.

**deleteCredential** (*issuanceID*, *transactionID*)

Delete/cancel the requested issuance of a secure document / credential.

**Authorization:** To be defined

**Parameters**

- **issuanceID** (*str*) – The ID of the issuance.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

**readCredential** (*credentialID*, *filter*, *transactionID*)

Retrieve the attributes/status of an issued credential. A wide range of information may be returned, dependant on the type of credential that was issued, smart card, mobile, passport, etc.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error, in the case of success the requested data will be returned.

**suspendCredential** (*credentialID*, *transactionID*)

Suspend an issued credential. For electronic credentials this will suspend any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**unsuspendCredential** (*credentialID*, *transactionID*)

Unsuspend an issued credential. For electronic credentials this will unsuspend any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.

- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

**cancelCredential** (*credentialID*, *transactionID*)

Cancel an issued credential. For electronic credentials this will revoke any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

## 5.8 ID Usage

### 5.8.1 Services

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

Verify Identity based on UIN and set of Identity Attributes (biometric data, credential, etc.)

**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 Identity Attributes (biometric data, credential, etc.)

**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

**readAttributes** (*UIN*, *names*)

Read 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

**readAttributeSet** (*UIN*, *setName*)

Read 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

## 5.9 Under discussion

### 5.9.1 Services

**listCredentialProfiles** (*filter, transactionID*)

Retrieve the list of credential profiles.

**Authorization:** To be defined

**Parameters**

- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the credential profile list.

**readCredentialProfile** (*credentialProfileID, filter, transactionID*)

Retrieve the credential profile.

**Authorization:** To be defined

**Parameters**

- **credentialProfileID** (*str*) – The ID of the credential profile.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the credential profile.

---

**createDocument** (*personID, documentID, documentData, transactionID*)

Add a new document for a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **documentID** (*str*) – The ID of the document.
- **documentData** – The content and attributes of the document.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error. In the case of success, a document identifier.

**readDocument** (*documentID, filter, transactionID*)

Retrieve document data.

**Authorization:** To be defined

**Parameters**

- **documentID** (*str*) – The ID of the document.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.

- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the document data.

**updateDocument** (*documentID*, *documentData*, *transactionID*)

Update a document for a person.

**Authorization:** To be defined

#### Parameters

- **documentID** (*str*) – The ID of the document.
- **documentData** – The content and attributes of the document, this can be partial data.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**deleteDocument** (*documentID*, *transactionID*)

Delete a document for a person.

**Authorization:** To be defined

#### Parameters

- **documentID** (*str*) – The ID of the document.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**updateDocumentValStatus** (*documentID*, *status*, *transactionID*)

Updates the status of a document validation.

**Authorization:** To be defined

#### Parameters

- **documentID** (*str*) – The ID of the document.
- **status** – The status of the document validation, e.g. 'ready' to validate.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**readDocumentValStatus** (*documentID*, *transactionID*)

Retrieve the status of a document validation.

**Authorization:** To be defined

#### Parameters

- **documentID** (*str*) – The ID of the document.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the document validation status and its metadata.

---

**createBiometric** (*personID*, *biometricID*, *biometricData*, *transactionID*)

Add a new biometric for a person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **biometricID** (*str*) – The ID of the biometric.
- **biometricData** – The content and attributes of the biometric.

- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error. In the case of success, a biometric identifier.

**readBiometric** (*biometricID, filter, transactionID*)

Retrieve biometric data.

NOTE - do we want this method in the system? We don't believe that this data should be retrievable. A separate method is provided for reading enrolled biometric metadata (see below).

**Authorization:** To be defined

#### Parameters

- **biometricID** (*str*) – The ID of the biometric.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biometric data.

**readBiometricMetadata** (*biometricID, filter, transactionID*)

Retrieve biometric data.

**Authorization:** To be defined

#### Parameters

- **biometricID** (*str*) – The ID of the biometric.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biometric metadata.

**updateBiometric** (*biometricID, biometricData, transactionID*)

Update a biometric for a person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **biometricID** (*str*) – The ID of the biometric.
- **biometricData** – The content and attributes of the biometric, this can be partial data.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**deleteBiometric** (*biometricID, transactionID*)

Delete a biometric for a person.

**Authorization:** To be defined

#### Parameters

- **biometricID** (*str*) – The ID of the biometric.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**updateBiometricValStatus** (*biometricID, status, transactionID*)

Updates the status of a biometric validation.

**Authorization:** To be defined

#### Parameters

- **biometricID** (*str*) – The ID of the biometric.
- **status** – The status of the biometric validation, e.g. 'ready' to validate.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biometric validation status.

**readBiometricValStatus** (*biometricID, transactionID*)

Retrieve the status of a biometric validation.

**Authorization:** To be defined

#### Parameters

- **biometricID** (*str*) – The ID of the biometric.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biometric validation status and metadata.

**createBiographic** (*personID, biographicID, biographicData, transactionID*)

Add a new biographic for a person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **biographicID** (*str*) – The ID of the biographic.
- **biographicData** – The content and attributes of the biographic.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error. In the case of success, a biographic identifier.

**readBiographic** (*biographicID, filter, transactionID*)

Retrieve biographic data.

**Authorization:** To be defined

#### Parameters

- **biographicID** (*str*) – The ID of the biographic.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biographic data.

**updateBiographic** (*biographicID, biographicData, transactionID*)

Update a biographic for a person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **biographicID** (*str*) – The ID of the biographic.
- **biographicData** – The content and attributes of the biographic, this can be partial data.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**deleteBiographic** (*biographicID*, *transactionID*)

Delete a biographic for a person.

**Authorization:** To be defined

**Parameters**

- **biographicID** (*str*) – The ID of the biographic.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**updateBiographicValStatus** (*biographicID*, *status*, *transactionID*)

Updates the status of a biographic validation.

**Authorization:** To be defined

**Parameters**

- **biographicID** (*str*) – The ID of the biographic.
- **status** – The status of the biographic validation, e.g. ‘ready’ to validate.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biographic validation status.

**readBiographicValStatus** (*biographicID*, *transactionID*)

Retrieve the status of a biographic validation.

**Authorization:** To be defined

**Parameters**

- **biographicID** (*str*) – The ID of the biographic.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error, and in case of success the biographic validation status and metadata.

## 5.9.2 Filter

The “filter” parameter used in “read” calls is used to provide a set of identifiers that limit the amount of data that is returned. It is often the case that the whole data set is not required, but instead, a subset of that data. Where possible, existing standards based identifiers should be used for the attributes to retrieve.

E.g. For surname/familyname, use OID 2.5.4.4 or id-at-surname.

Some calls may require new filter attributes to be defined. E.g. when retrieving biometric data, the caller may only want the meta data about that biometric, rather than the actual biometric data.

## 5.9.3 Transaction ID

The “transactionID” is 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.

### 5.9.4 Data Model

Table 5.11: Enrolment Data Model

Type	Description	Example
Person	Person who is known to an identity assurance system.	TBD
Document Data	a dictionary (list of names and values) giving the document data of interest for the document services.	TBD
Biometric Data	Digital representation of biometric characteristics. 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.	fingerprint, portrait, iris
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biographic services.	TBD



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

## 6.1 Enrollment Component

The enrolment component MAY implement the following interfaces:

### 6.1.1 Enrollment Services

#### Services

**createPerson** (*personID*, *personData*, *transactionID*)

Insert a new person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person. If the person already exists for the ID an error is returned.
- **personData** (*dict*) – The person attributes.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**readPerson** (*personID*, *filter*, *transactionID*)

Retrieve the attributes of a person.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error and in case of success the person data.

**updatePerson** (*personID*, *personData*, *transactionID*)

Update a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personData** (*dict*) – The person data, this can be partial data.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**deletePerson** (*personID*, *transactionID*)

Delete a person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**findPeople** (*filter*, *transactionID*)

Retrieve a list of people who match passed in search criteria.

**Authorization:** To be defined

**Parameters**

- **filter** (*dict*) – The search criteria to match on.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error and in case of success the matching person list.

## Filter

The “filter” parameter used in “read” calls is used to provide a set of identifiers that limit the amount of data that is returned. It is often the case that the whole data set is not required, but instead, a subset of that data. Where possible, existing standards based identifiers should be used for the attributes to retrieve.

E.g. For surname/familyname, use OID 2.5.4.4 or id-at-surname.

Some calls may require new filter attributes to be defined. E.g. when retrieving biometric data, the caller may only want the meta data about that biometric, rather than the actual biometric data.

## Transaction ID

The “transactionID” is 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.

## Data Model

Table 6.1: Enrolment Data Model

Type	Description	Example
Person	Person who is known to an identity assurance system.	TBD
Document Data	a dictionary (list of names and values) giving the document data of interest for the document services.	TBD
Biometric Data	Digital representation of biometric characteristics. 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.	fingerprint, portrait, iris
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biographic services.	TBD

## 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:

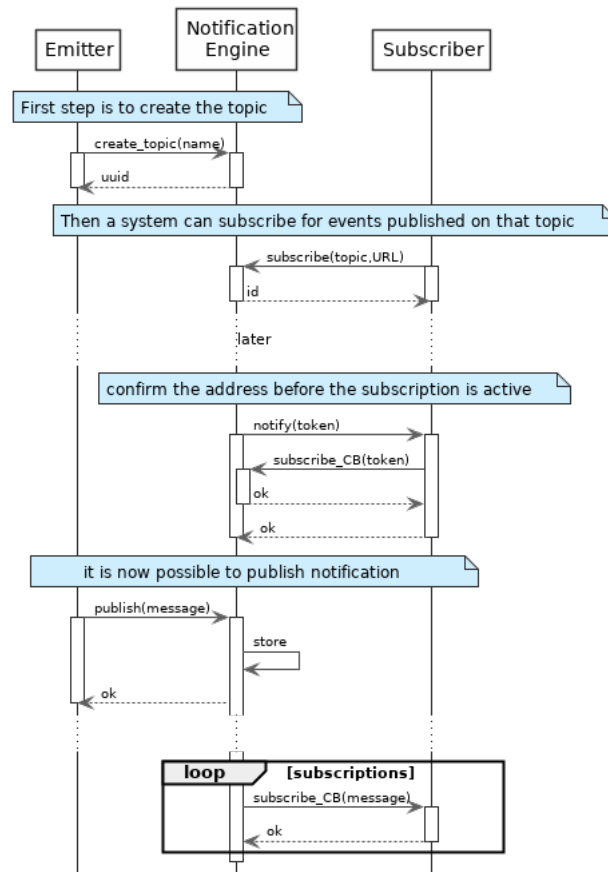


Fig. 6.1: Subscription &amp; Notification Process

## 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 6.2: 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

**readPersonAttributes** (*UIN*, *names*)

Read 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.

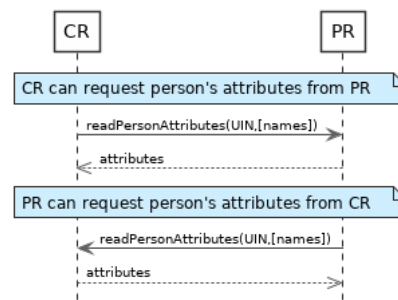


Fig. 6.2: readPersonAttributes Sequence Diagram

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

Match person attributes. This service is used to check the value of attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

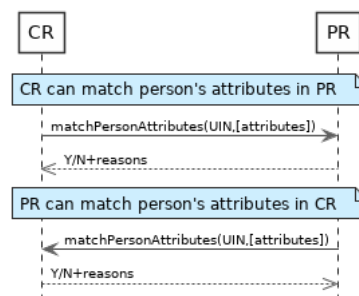


Fig. 6.3: matchPersonAttributes Sequence Diagram

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

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

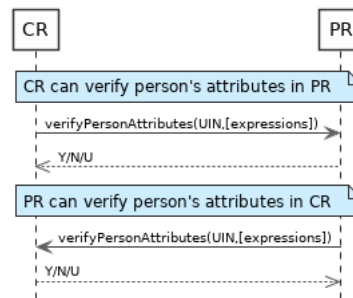


Fig. 6.4: verifyPersonAttributes Sequence Diagram

#### queryPersonUIN (attributes)

Query the persons by a set of attributes. This service is used when the UIN is unknown. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

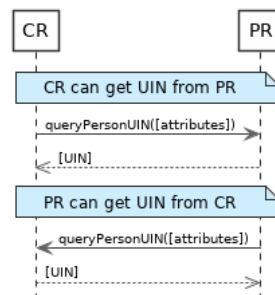


Fig. 6.5: queryPersonUIN Sequence Diagram

#### queryPersonList (attributes, names)

Query the persons by a list of attributes and their values. This service is proposed as an optimization of a sequence of calls to queryPersonUIN() and readPersonAttributes().

**Authorization:** **To be defined**

##### Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find the persons.  
 Each attribute is described with its name and its value
- **names** (*list[str]*) – The names of the attributes requested

**Returns** a list of lists of pairs (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.

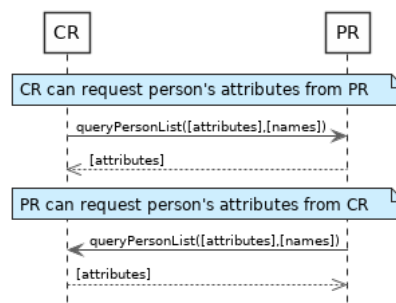


Fig. 6.6: queryPersonList Sequence Diagram

**readDocument** (*UINs, documentType, format*)

Read 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.

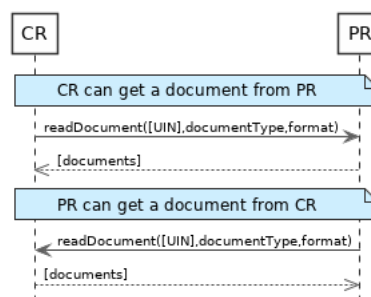


Fig. 6.7: readDocument Sequence Diagram

## Dictionary

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



Table 6.3: 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 6.4: Certificate Attributes

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

Table 6.5: Union Attributes

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

Table 6.6: Filiation Attributes

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

Table 6.7: Document Type

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

### 6.2.3 Population Registry Services

This interface describes services to manage a registry of the population in the context of an identity system. It is based on the following principles:

- It supports a history of identities, meaning that a person has one identity and this identity has a history.
- Images can be passed by value or reference. When passed by value, they are base64-encoded.
- Existing standards are used whenever possible.

- This interface is complementary to the data access interface. The data access interface is used to query the persons and uses the reference identity to return attributes.
- The population registry can store the biometric data or can rely on the ABIS subsystem to do it. The preferred solution, for a clean separation of data of different nature and by application of GDPR principles, is to put the biometric data only in the ABIS. Yet many existing systems store biometric data with the biographic data and this specification gives the flexibility to do it.

See *Population Registry Management* for the technical details of this interface.

## Services

**createPerson** (*personID*, *personData*, *transactionID*)

Create a new person.

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The ID of the person. If the person already exists for the ID an error is returned.
- **personData** – The person attributes.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**readPerson** (*personID*, *transactionID*)

Read the attributes of a person.

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error and in case of success the person data.

**updatePerson** (*personID*, *personData*, *transactionID*)

Update a person.

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The ID of the person.
- **personData** (*dict*) – The person data.

**Returns** a status indicating success or error.

**deletePerson** (*personID*, *transactionID*)

Delete a person and all its identities.

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**createIdentity** (*personID*, *identityID*, *identity*, *transactionID*)

Create a new identity in a person. If no *identityID* is provided, a new one is generated. If *identityID* is provided, it is checked for uniqueness and used for the identity if unique. An error is returned if the provided *identityID* is not unique.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – The ID of the identity.
- **identity** – The new identity data.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**readIdentity** (*personID*, *identityID*, *transactionID*)

Read one or all the identities of one person.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity. If not provided, all identities are returned.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error, and in case of success a list of identities.

**updateIdentity** (*personID*, *identityID*, *identity*, *transactionID*)

Update an identity. An identity can be updated only in the status `claimed`.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **identity** – The identity data.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

**partialUpdateIdentity** (*personID*, *identityID*, *identity*, *transactionID*)

Update part of an identity. Not all attributes are mandatory. The payload is defined as per [RFC 7396](#). An identity can be updated only in the status `claimed`.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **identity** – Part of the identity data.

**Returns** a status indicating success or error.

---

**deleteIdentity** (*personID*, *identityID*, *transactionID*)

Delete an identity.

**Authorization:** To be defined**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.**setIdentityStatus** (*personID*, *identityID*, *status*, *transactionID*)

Set an identity status.

**Authorization:** To be defined**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity.
- **status** (*str*) – The new status of the identity.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.

---

**defineReference** (*personID*, *identityID*, *transactionID*)

Define the reference identity of one person.

**Authorization:** To be defined**Parameters**

- **personID** (*str*) – The ID of the person.
- **personID** – The ID of the identity being now the reference.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error.**readReference** (*personID*, *transactionID*)

Read the reference identity of one person.

**Authorization:** To be defined**Parameters**

- **personID** (*str*) – The ID of the person.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error and in case of success the reference identity.

---

**readGalleries** (*transactionID*)

Read the ID of all the galleries.

**Authorization:** To be defined

**Parameters** **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error, and in case of success a list of gallery ID.

**readGalleryContent** (*galleryID, transactionID*)

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

**Authorization:** To be defined

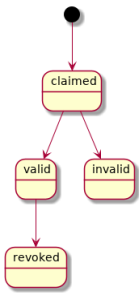
**Parameters**

- **galleryID** (*str*) – Gallery whose content will be returned.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

**Returns** a status indicating success or error. In case of success a list of person/identity IDs.

## Data Model

Table 6.8: Population Registry Data Model

Type	Description	Example
Gallery	A group of persons related by a common purpose, designation, or status. A person can belong to multiple galleries.	VIP, Wanted, etc.
Person	<p>Person who is known to an identity assurance system.</p> <p>A person record has:</p> <ul style="list-style-type: none"> <li>• a status, such as <code>active</code> or <code>inactive</code>, defining the status of the record (the record can be excluded from queries based on this status),</li> <li>• a physical status, such as <code>alive</code> or <code>dead</code>, defining the status of the person,</li> <li>• a set of identities, keeping track of all identity data submitted by the person during the life of the system,</li> <li>• a reference identity, i.e. a consolidated view of all the identities defining the current correct identity of the person. It corresponds usually to the last valid identity but it can also include data from previous identities.</li> </ul>	N/A
Identity	<p>The attributes describing an identity of a person.</p> <p>An identity has a status such as: <code>claimed</code> (identity not yet validated), <code>valid</code> (the identity is valid), <code>invalid</code> (the identity is not valid), <code>revoked</code> (the identity cannot be used any longer).</p> <p>An identity can be updated only in the status <code>claimed</code>.</p> <p>The allowed transitions for the status are represented below:</p>  <pre> graph TD     Start(( )) --&gt; claimed[claimed]     claimed --&gt; valid[valid]     claimed --&gt; invalid[invalid]     valid --&gt; revoked[revoked]   </pre> <p>The attributes are separated into two categories: the biographic data and the contextual data.</p>	N/A
Biographic Data	A dictionary (list of names and values) giving the biographic data of the identity	<code>firstName</code> , <code>lastName</code> , <code>dateOfBirth</code> , etc.
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	<code>operatorName</code> , <code>enrolmentDate</code> , etc.
Biometric Data	Digital representation of biometric characteristics. 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.	<code>fingerprint</code> , <code>portrait</code> , <code>iris</code>
Document	The document data (images) attached to the identity and used to validate it.	Birth certificate, invoice

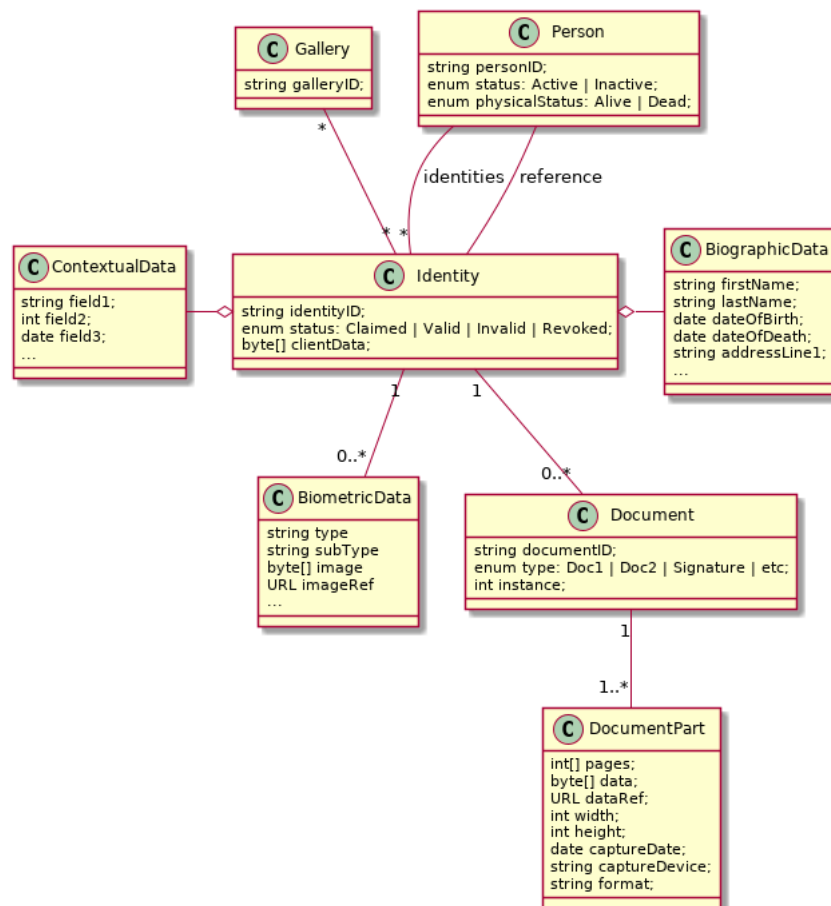


Fig. 6.8: Population Registry Data Model

## 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:

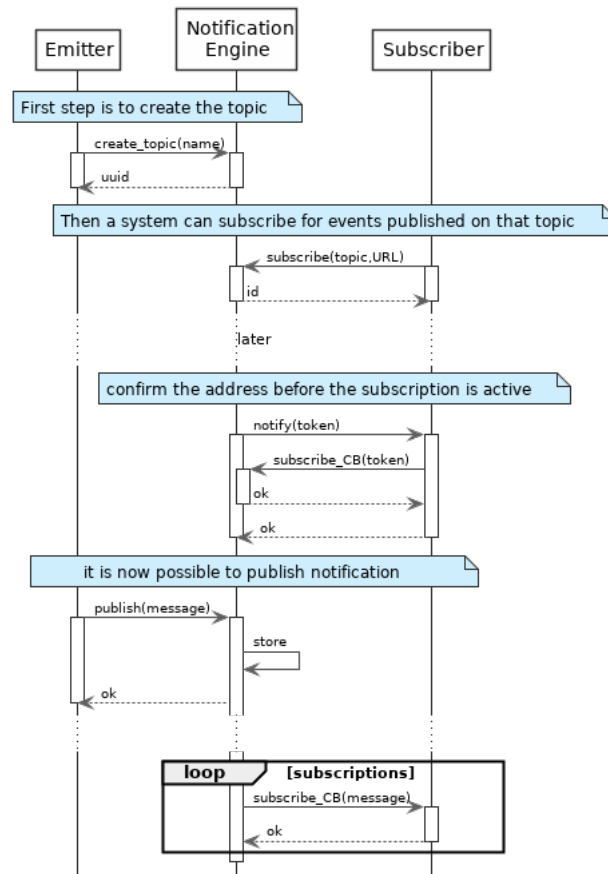


Fig. 6.9: Subscription &amp; Notification Process

## 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 6.9: 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

**readPersonAttributes** (*UIN, names*)

Read 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.

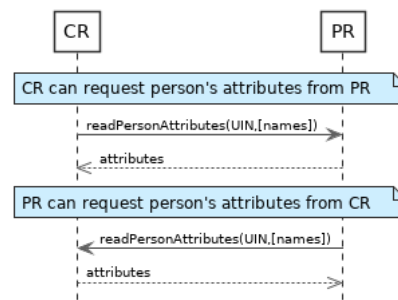


Fig. 6.10: readPersonAttributes Sequence Diagram

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

Match person attributes. This service is used to check the value of attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

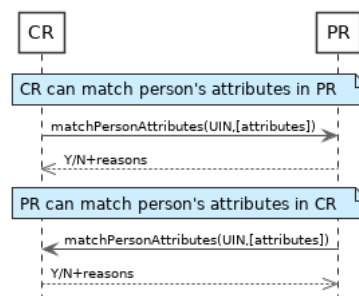


Fig. 6.11: matchPersonAttributes Sequence Diagram

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

Evaluate expressions on person attributes. This service is used to evaluate simple expressions on person's attributes without exposing private data. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

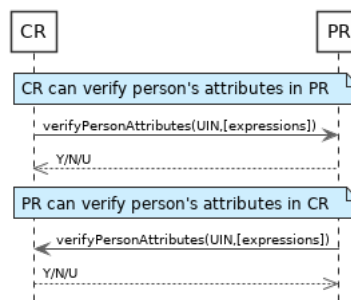


Fig. 6.12: verifyPersonAttributes Sequence Diagram

#### queryPersonUIN (attributes)

Query the persons by a set of attributes. This service is used when the UIN is unknown. The implementation can use a simple comparison or a more advanced technique (for example: phonetic comparison for names)

**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.

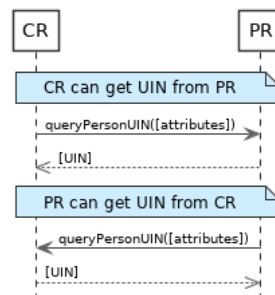


Fig. 6.13: queryPersonUIN Sequence Diagram

#### queryPersonList (attributes, names)

Query the persons by a list of attributes and their values. This service is proposed as an optimization of a sequence of calls to queryPersonUIN() and readPersonAttributes().

**Authorization:** **To be defined**

##### Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find the persons.  
 Each attribute is described with its name and its value
- **names** (*list[str]*) – The names of the attributes requested

**Returns** a list of lists of pairs (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.

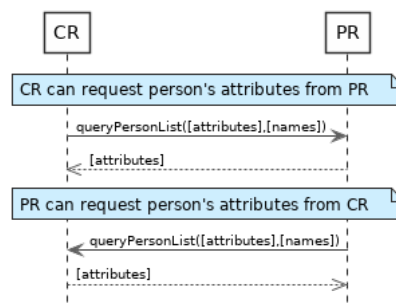


Fig. 6.14: queryPersonList Sequence Diagram

**readDocument** (*UINs, documentType, format*)

Read 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.

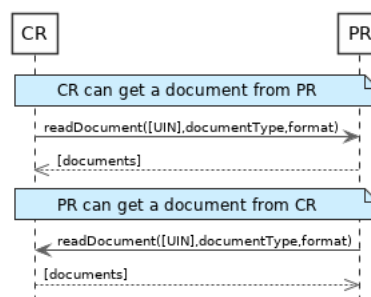


Fig. 6.15: readDocument Sequence Diagram

## Dictionary

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

Table 6.10: 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 6.11: Certificate Attributes

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

Table 6.12: Union Attributes

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

Table 6.13: Filiation Attributes

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

Table 6.14: 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

### **generateUIN** (*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.

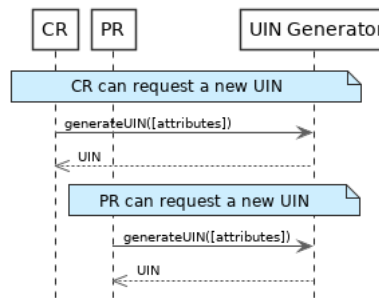


Fig. 6.16: `generateUIN` Sequence Diagram

## 6.5 ABIS

The ABIS component MAY implement the following interfaces:

### 6.5.1 Biometrics

This interface describes 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 credentials 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 preferred image format for biometric data is ISO-19794.

#### About 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

**create** (*personID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *clientData*, *callback*, *transactionID*, *options*)

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

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The person ID. This is optional and will be generated if not provided
- **encounterID** (*str*) – The encounter ID. This is optional and will be generated if not provided
- **galleryID** (*list (str)*) – the gallery ID to which this encounter belongs. A minimum of one gallery must be provided
- **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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`, `algorithm`.

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

**read** (*personID*, *encounterID*, *callback*, *transactionID*, *options*)

Read the data of an encounter.

**Authorization:** To be defined

### Parameters

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the person are returned.
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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** (*personID*, *encounterID*, *galleryID*, *biographicData*, *contextualData*, *biometricData*, *callback*, *transactionID*, *options*)

Update an encounter.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID
- **galleryID** (*list* (*str*)) – the gallery ID to which this encounter belongs. A minimum of one gallery must be provided
- **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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`, `algorithm`.

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

**delete** (*personID*, *encounterID*, *callback*, *transactionID*, *options*)

Delete an encounter.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID. This is optional. If not provided, all the encounters of the person are deleted.
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**readTemplate** (*personID*, *encounterID*, *biometricType*, *biometricSubType*, *callback*, *transactionID*, *options*)

Read the generated template.

**Authorization:** To be defined

**Parameters**

- **personID** (*str*) – The person ID
- **encounterID** (*str*) – The encounter ID.
- **biometricType** (*str*) – The type of biometrics to consider (optional)
- **biometricSubType** (*str*) – The subtype of biometrics to consider (optional)
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.



- **options** (*dict*) – the processing options. Supported options are `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.

**readGalleries** (*callback, transactionID, options*)

Read the ID of all the galleries.

**Authorization:** To be defined

#### Parameters

- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**readGalleryContent** (*galleryID, callback, transactionID, options*)

Read 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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `priority`.

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

**identify** (*galleryID, filter, biometricData, callback, transactionID, options*)

Identify a person 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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *personID*, *callback*, *transactionID*, *options*)

Identify a person using biometrics data of a person 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)
- **personID** – the person ID
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *personID*, *biometricData*, *callback*, *transactionID*, *options*)

Verify an identity using biometrics data.

**Authorization:** To be defined

#### Parameters

- **galleryID** (*str*) – Search only in this gallery. If the person does not belong to this gallery, an error is returned.
- **personID** (*str*) – The person ID
- **biometricData** – The biometric data
- **callback** – The address of a service to be called when the result is available.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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*, *transactionID*, *options*)

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

**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.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **options** (*dict*) – the processing options. Supported options are `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.

## Options

Table 6.15: Biometric Services Options

Name	Description
<code>priority</code>	Priority of the request. Values range from 0 to 9
<code>maxNbCand</code>	The maximum number of candidates to return.
<code>threshold</code>	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
<code>algorithm</code>	Specify the type of algorithm to be used.
<code>accuracyLevel</code>	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 6.16: Biometric Data Model

Type	Description	Example
Gallery	A group of persons related by a common purpose, designation, or status. A person can belong to multiple galleries.	TBD
Person	Person who is known to an identity assurance system.	TBD
Encounter	Event in which the client application interacts with a person resulting in data being collected during or about the encounter. An encounter is characterized by an <i>identifier</i> and a <i>type</i> (also called <i>purpose</i> 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 <i>range</i> 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. 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.	fingerprint, portrait, iris
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

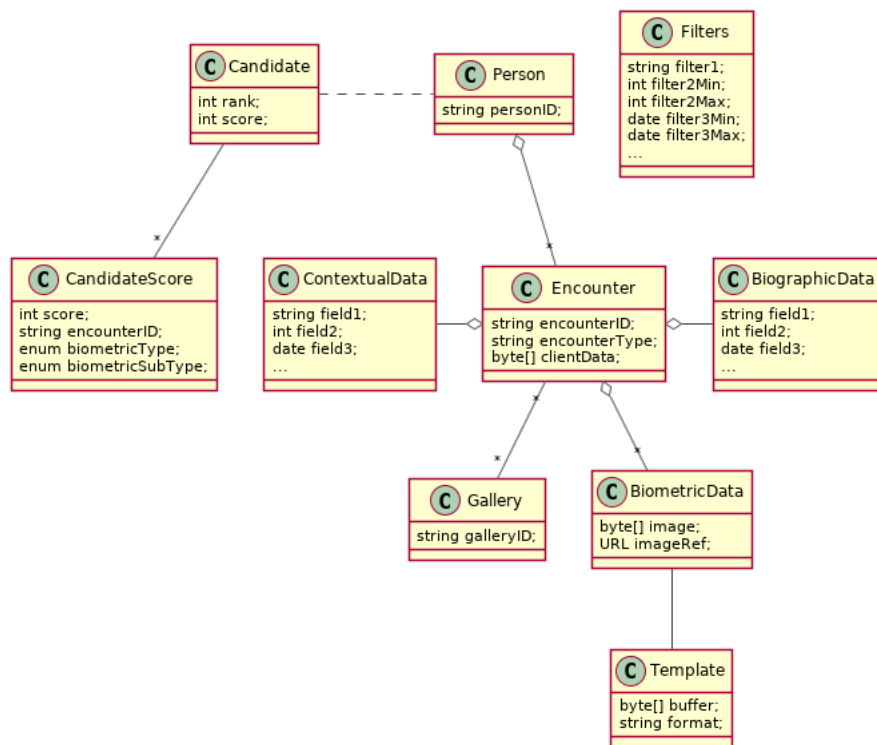


Fig. 6.17: Biometric Data Model

## 6.6 Credential Management System

The credential management system component MAY implement the following interfaces:

### 6.6.1 Credential Services

#### Services

**createCredential** (*personID*, *credentialProfileID*, *additionalData*, *transactionID*)

Request issuance of a secure document / credential.

**Authorization:** To be defined

#### Parameters

- **personID** (*str*) – The ID of the person.
- **credentialProfileID** (*str*) – The ID of the credential profile to issue to the person.
- **additionalData** (*dict*) – Additional data relating to the requested credential profile, e.g. credential lifetime if overriding default, delivery addresses, etc.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error. In the case of success, an issuance identifier.

**readCredentialIssuance** (*issuanceID*, *filter*, *transactionID*)

Retrieve the data/status of an issuance.

**Authorization:** To be defined

#### Parameters

- **issuanceID** (*str*) – The ID of the issuance.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error, and in case of success the issuance data/status.

**updateCredential** (*issuanceID*, *additionalData*, *transactionID*)

Update the requested issuance of a secure document / credential.

**Authorization:** To be defined

**Parameters**

- **issuanceID** (*str*) – The ID of the issuance.
- **transactionID** (*string*) – The client generated transactionID.
- **additionalData** (*dict*) – Additional data relating to the requested credential profile, e.g. credential lifetime if overriding default, delivery addresses, etc.

**Returns** a status indicating success or error.

**deleteCredential** (*issuanceID*, *transactionID*)

Delete/cancel the requested issuance of a secure document / credential.

**Authorization:** To be defined

**Parameters**

- **issuanceID** (*str*) – The ID of the issuance.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

**readCredential** (*credentialID*, *filter*, *transactionID*)

Retrieve the attributes/status of an issued credential. A wide range of information may be returned, dependant on the type of credential that was issued, smart card, mobile, passport, etc.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **filter** (*set*) – The (optional) set of required attributes to retrieve.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error, in the case of success the requested data will be returned.

**suspendCredential** (*credentialID*, *transactionID*)

Suspend an issued credential. For electronic credentials this will suspend any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **transactionID** (*string*) – The (optional) client generated transactionID.

**Returns** a status indicating success or error.

**unsuspendCredential** (*credentialID*, *transactionID*)

Unsuspend an issued credential. For electronic credentials this will unsuspend any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

**cancelCredential** (*credentialID*, *transactionID*)

Cancel an issued credential. For electronic credentials this will revoke any PKI certificates that are present.

**Authorization:** To be defined

**Parameters**

- **credentialID** (*str*) – The ID of the credential.
- **transactionID** (*string*) – The client generated transactionID.

**Returns** a status indicating success or error.

## 6.7 Third Party Services

The third party component MAY implement the following interfaces:

### 6.7.1 ID Usage

#### Services

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

Verify Identity based on UIN and set of Identity Attributes (biometric data, credential, etc.)

**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 Identity Attributes (biometric data, credential, etc.)

**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

**readAttributes** (*UIN, names*)

Read 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

**readAttributeSet** (*UIN, setName*)

Read 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

## 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.

**CMS** Credential Management System

**Credential** A document, object, or data structure that vouches for the identity of a person through some method of trust and authentication. Common types of identity credentials include - but are not limited to — ID cards, certificates, numbers, passwords, or SIM cards. A biometric identifier can also be used as a credential once it has been registered with the identity provider.

(Source: [ID4D Practioner's Guide](#))

**Encounter** Event in which the client application interacts with a person 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).

(Source: ISO-30108-1)

**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
```

(continues on next page)



(continued from previous page)

```
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

TBD: 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,
  "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
  }
]

```

**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	<ul style="list-style-type: none"><li>source: identification of the system emitting the event</li><li>uin of the new born</li><li>uin1 of the first parent (optional if parent is unknown)</li><li>uin2 of the second parent (optional if parent is unknown)</li></ul> Example: <pre>{   "source": "systemX",   "uin": "123456789",   "uin1": "123456789",   "uin2": "234567890" }</pre>
death	<ul style="list-style-type: none"><li>source: identification of the system emitting the event</li><li>uin of the dead person</li></ul> Example: <pre>{   "source": "systemX",   "uin": "123456789" }</pre>
birthCancellation	<ul style="list-style-type: none"><li>source: identification of the system emitting the event</li><li>uin of the person whose birth declaration is being cancelled</li></ul> Example: <pre>{   "source": "systemX",   "uin": "123456789", }</pre>

Continued on next page

Table 7.1 – continued from previous page

Event Type	Message
foetalDeath	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the new born</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789" }</pre>
marriage	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin1 of the first conjoint</li> <li>• uin2 of the second conjoint</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin1": "123456789",   "uin2": "234567890" }</pre>
divorce	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin1 of the first conjoint</li> <li>• uin2 of the second conjoint</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin1": "123456789",   "uin2": "234567890" }</pre>
annulment	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin1 of the first conjoint</li> <li>• uin2 of the second conjoint</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin1": "123456789",   "uin2": "234567890" }</pre>
separation	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin1 of the first conjoint</li> <li>• uin2 of the second conjoint</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin1": "123456789",   "uin2": "234567890" }</pre>

Continued on next page

Table 7.1 – continued from previous page

Event Type	Message
adoption	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the child</li> <li>• uin1 of the first parent</li> <li>• uin2 of the second parent (optional)</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789",   "uin1": "234567890" }</pre>
legitimation	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the child</li> <li>• uin1 of the first parent</li> <li>• uin2 of the second parent (optional)</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "987654321",   "uin1": "123456789",   "uin2": "234567890" }</pre>
recognition	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the child</li> <li>• uin1 of the first parent</li> <li>• uin2 of the second parent (optional)</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789",   "uin2": "234567890" }</pre>
changeOfName	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the person</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789" }</pre>
changeOfGender	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the person</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789" }</pre>

Continued on next page



Table 7.1 – continued from previous page

Event Type	Message
updatePerson	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the person</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789" }</pre>
newPerson	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the person</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789" }</pre>
duplicatePerson	<ul style="list-style-type: none"> <li>• source: identification of the system emitting the event</li> <li>• uin of the person to be kept</li> <li>• duplicates: list of uin for records identified as duplicates</li> </ul> <p>Example:</p> <pre>{   "source": "systemX",   "uin": "123456789",   "duplicates": [     "234567890",     "345678901"   ] }</pre>

**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

#### Person

**GET /v1/persons**

Query for persons using a set of attributes. Retrieve the UIN or the person attributes. This service is used when the UIN is unknown. Example: <http://registry.com/v1/persons?firstName=John&lastName=Do&names=firstName>

**Query Parameters**

- **attributes** (*object*) – The attributes (names and values) used to query (Required)
- **names** (*array*) – The names of the attributes to return. If not provided, only the UIN is returned
- **max** (*number*) – The maximum number of records to return. Default is 10

**Status Codes**

- **200 OK** – The requested attributes for all found persons (a list of at least one entry). If no names are given, a flat list of UIN is returned. If at least one name is given, a list of dictionaries (one dictionary per record) is returned.
- **400 Bad Request** – Invalid parameter
- **401 Unauthorized** – Client must be authenticated
- **403 Forbidden** – Service forbidden
- **404 Not Found** – No record found
- **500 Internal Server Error** – Unexpected 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

[
  "string"
]
```

**Example response:**

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

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

**GET /v1/persons/{uin}**

Read attributes for a person. Example: <http://registry.com/v1/persons/123456789?attributeNames=firstName&attributeNames=lastName&attributeNames=dob>

**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

**Example request:**

```
GET /v1/persons/{uin}?attributeNames=%5B%27string%27%5D 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. 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

**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 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 expressions.

**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

#### 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": "<",
    "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"
}
```

## Document

### GET /v1/persons/{uin}/document

Read in an unstructured format (PDF, image) a document such as a marriage certificate. Example: `http://registry.com/v1/persons/123456789/document?doctype=marriage&secondaryUin=234567890&format=pdf`

#### 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

#### Example request:

```
GET /v1/persons/{uin}/document?doctype=string&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 <i>Person Attributes</i> )	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 <i>Person Attributes</i> )	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 Population Registry Management

Download the OpenAPI file for this interface [pr.yaml](#).

### Services

#### Person

**POST /v1/persons/{personId}**

Create one person

##### Parameters

- **personId** (*string*) – the id of the person

##### Query Parameters

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

##### Status Codes

- 201 **Created** – Operation successful
- 400 **Bad Request** – Bad request
- 403 **Forbidden** – Operation not allowed
- 500 **Internal Server Error** – Unexpected error

**Example request:**

```
POST /v1/persons/{personId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "personId": {},
  "status": "ACTIVE",
  "physicalStatus": "DEAD"
}
```

**Example response:**

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

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

**GET /v1/persons/{personId}**  
Read one person

**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

**Status Codes**

- 200 OK – Read successful
- 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/persons/{personId}?transactionId=string HTTP/1.1
Host: example.com
```

**Example response:**

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

{
  "personId": {},
  "status": "ACTIVE",
  "physicalStatus": "DEAD"
}
```

**Example response:**

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

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

**PUT /v1/persons/{personId}**  
Update one person



**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

**Status Codes**

- 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/persons/{personId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "personId": {},
  "status": "ACTIVE",
  "physicalStatus": "DEAD"
}
```

**Example response:**

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

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

**DELETE /v1/persons/{personId}**  
Delete a person and all its identities

**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

**Status Codes**

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

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

## Identity

**GET /v1/persons/{personId}/identities**

Read all the identities of a person

### Parameters

- **personId** (*string*) – the id of the person

### Query Parameters

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

### Status Codes

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

### Example request:

```
GET /v1/persons/{personId}/identities?transactionId=string HTTP/1.1
Host: example.com
```

### Example response:

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

[
  {
    "identityId": "string",
    "status": "CLAIMED",
    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "enrolmentDate": "2019-01-11"
    },
    "biographicData": {
      "firstName": "John",
      "lastName": "Doo",
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA"
    },
    "biometricData": [
      {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "captureDevice": "string",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "resolution": 1,
        "compression": "NONE"
      }
    ],
    "documents": [
      {
        "documentId": "string",
        "documentType": "ID_CARD",

```

(continues on next page)

(continued from previous page)

```

        "instance": 1,
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "https://example.com",
                "width": 1,
                "height": 1,
                "format": "NONE",
                "captureDate": "2020-04-28",
                "captureDevice": "string"
            }
        ]
    }
]

```

**Example response:**

```

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

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

```

**POST /v1/persons/{personId}/identities**  
 Create one identity and generate its id
**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

**Status Codes**

- 200 OK – Insertion successful
- 400 Bad Request – Bad request
- 403 Forbidden – Insertion not allowed
- 500 Internal Server Error – Unexpected error

**Example request:**

```

POST /v1/persons/{personId}/identities?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "identityId": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrolmentDate": "2019-01-11"
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",

```

(continues on next page)

(continued from previous page)

```

    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "captureDevice": "string",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "resolution": 1,
      "compression": "NONE"
    }
  ],
  "documents": [
    {
      "documentId": "string",
      "documentType": "ID_CARD",
      "instance": 1,
      "parts": [
        {
          "pages": [
            1
          ],
          "data": "c3RyaW5n",
          "dataRef": "https://example.com",
          "width": 1,
          "height": 1,
          "format": "NONE",
          "captureDate": "2020-04-28",
          "captureDevice": "string"
        }
      ]
    }
  ]
}

```

**Example response:**

```

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

{
  "identityId": "string"
}

```

**Example response:**

```

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

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

```

**POST /v1/persons/{personId}/identities/{identityId}****Create one identity**

Create one new identity for a person. The provided identityId is checked for validity and used for the new identity.

**Parameters**

- **personId** (*string*) – the id of the person

- **identityId** (*string*) – the id of the identity

### Query Parameters

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

### Status Codes

- 201 Created – Insertion successful
- 400 Bad Request – Bad request
- 403 Forbidden – Insertion not allowed
- 500 Internal Server Error – Unexpected error

### Example request:

```
POST /v1/persons/{personId}/identities/{identityId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "identityId": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrolmentDate": "2019-01-11"
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "captureDevice": "string",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "resolution": 1,
      "compression": "NONE"
    }
  ],
  "documents": [
    {
      "documentId": "string",
      "documentType": "ID_CARD",
      "instance": 1,
      "parts": [
        {
          "pages": [
            1
          ],
          "data": "c3RyaW5n",
          "dataRef": "https://example.com",
          "width": 1,
          "height": 1,
          "format": "NONE",
          "captureDate": "2020-04-28",
          "captureDevice": "string"
        }
      ]
    }
  ]
}
```

(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"
}

```

**GET /v1/persons/{personId}/identities/{identityId}****Read one identity****Parameters**

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

**Query Parameters**

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

**Status Codes**

- 200 OK – Read successful
- 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/persons/{personId}/identities/{identityId}?transactionId=string HTTP/1.1
Host: example.com

```

**Example response:**

```

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

{
  "identityId": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrolmentDate": "2019-01-11"
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",

```

(continues on next page)

(continued from previous page)

```

        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "captureDevice": "string",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "resolution": 1,
        "compression": "NONE"
      }
    ],
    "documents": [
      {
        "documentId": "string",
        "documentType": "ID_CARD",
        "instance": 1,
        "parts": [
          {
            "pages": [
              1
            ],
            "data": "c3RyaW5n",
            "dataRef": "https://example.com",
            "width": 1,
            "height": 1,
            "format": "NONE",
            "captureDate": "2020-04-28",
            "captureDevice": "string"
          }
        ]
      }
    ]
  }
}

```

**Example response:**

```

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

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

```

**PUT /v1/persons/{personId}/identities/{identityId}****Update one identity****Parameters**

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

**Query Parameters**

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

**Status Codes**

- 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/persons/{personId}/identities/{identityId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "identityId": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrolmentDate": "2019-01-11"
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "captureDevice": "string",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "resolution": 1,
      "compression": "NONE"
    }
  ],
  "documents": [
    {
      "documentId": "string",
      "documentType": "ID_CARD",
      "instance": 1,
      "parts": [
        {
          "pages": [
            1
          ],
          "data": "c3RyaW5n",
          "dataRef": "https://example.com",
          "width": 1,
          "height": 1,
          "format": "NONE",
          "captureDate": "2020-04-28",
          "captureDevice": "string"
        }
      ]
    }
  ]
}

```

**Example response:**

```

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

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

```

**PATCH /v1/persons/{personId}/identities/{identityId}**  
 Update partially one identity



Update partially an identity. Payload content is a partial identity object compliant with RFC7396.

#### Parameters

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

#### Query Parameters

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

#### Status Codes

- 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:

```
PATCH /v1/persons/{personId}/identities/{identityId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "galleries": [
    "G1",
    "G2"
  ],
  "biographicData": {
    "gender": null,
    "nationality": "FRA"
  }
}
```

#### Example response:

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

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

**DELETE /v1/persons/{personId}/identities/{identityId}**

Delete one identity

#### Parameters

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

#### Query Parameters

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

#### Status Codes

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

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

**PUT /v1/persons/{personId}/identities/{identityId}/status**  
Change the status of an identity

**Parameters**

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

**Query Parameters**

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

**Status Codes**

- 204 No Content – Operation successful
- 400 Bad Request – Bad request
- 403 Forbidden – Operation not allowed
- 500 Internal Server Error – Unexpected error

**Example response:**

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

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

## Reference

**PUT /v1/persons/{personId}/identities/{identityId}/reference**  
Define the reference

**Parameters**

- **personId** (*string*) – the id of the person
- **identityId** (*string*) – the id of the identity

**Query Parameters**

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

**Status Codes**

- 204 No Content – Operation successful
- 400 Bad Request – Bad request
- 403 Forbidden – Operation not allowed
- 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/persons/{personId}/reference**  
 Read the reference
**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

**Status Codes**

- 200 OK – Read successful
- 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/persons/{personId}/reference?transactionId=string HTTP/1.1
Host: example.com

```

**Example response:**

```

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

{
  "identityId": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrolmentDate": "2019-01-11"
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "captureDevice": "string",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "resolution": 1,

```

(continues on next page)

(continued from previous page)

```

        "compression": "NONE"
    }
],
"documents": [
    {
        "documentId": "string",
        "documentType": "ID_CARD",
        "instance": 1,
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "https://example.com",
                "width": 1,
                "height": 1,
                "format": "NONE",
                "captureDate": "2020-04-28",
                "captureDevice": "string"
            }
        ]
    }
]
]
}

```

**Example response:**

```

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

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

```

**Gallery****GET /v1/galleries**

Read the ID of all the galleries

**Query Parameters**

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

**Status Codes**

- 200 OK – Operation successful
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 500 Internal Server Error – Unexpected error

**Example request:**

```

GET /v1/galleries?transactionId=string HTTP/1.1
Host: example.com

```

**Example response:**

```

HTTP/1.1 200 OK
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}**  
**Read the content of one gallery**

**Parameters**

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

**Query Parameters**

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

**Status Codes**

- 200 OK – Operation successful
- 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}?transactionId=string HTTP/1.1
Host: example.com

```

**Example response:**

```

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

[
  {
    "personId": "string",
    "identityId": "string"
  }
]

```

**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 Biometrics**

Download the OpenAPI file for this interface [abis.yaml](#).

## Services

### CRUD

#### POST /v1/persons

Create one encounter and generate ID for both the person and the encounter

##### Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)
- **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 – Operation successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned.
- 400 Bad Request – Bad request
- 403 Forbidden – Operation not allowed
- 500 Internal Server Error – Unexpected error

##### Example request:

```
POST /v1/persons?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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

{
  "personId": "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: createResponse****POST \${request.query.callback}****Create one encounter and generate both IDs 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

{
  "personId": "string",
  "encounterId": "string"
}

```

**Example request:**

```

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

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

```

**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"
}
```

**POST /v1/persons/{personId}/encounters/{encounterId}****Create one encounter**

Create one encounter in the person identified by his/her id. If the person does not yet exist, it is created automatically.

If the encounter already exists, an error 403 is returned.

**Parameters**

- **personId** (*string*) – the id of the person
- **encounterId** (*string*) – the id of the encounter

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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** – Creation successful
- 202 **Accepted** – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned
- 400 **Bad Request** – Bad request
- 403 **Forbidden** – Creation not allowed
- 500 **Internal Server Error** – Unexpected error

**Example request:**

```
POST /v1/persons/{personId}/encounters/{encounterId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",

```

(continues on next page)



(continued from previous page)

```

        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "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: createResponse**

**POST** \${request.query.callback}  
 Create one encounter 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

{
  "personId": "string",
  "encounterId": "string"
}

```

**Example request:**

```

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

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

```

**Example response:**

```

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

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

```

**GET /v1/persons/{personId}/encounters/{encounterId}**  
**Read one encounter**

**Parameters**

- **personId** (*string*) – the id of the person
- **encounterId** (*string*) – the id of the encounter

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- **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/persons/{personId}/encounters/{encounterId}?transactionId=string HTTP/1.1
Host: example.com

```

**Example response:**

```

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

{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",

```

(continues on next page)

(continued from previous page)

```

        "biometricSubType": "UNKNOWN",
        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "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}  
 Read one encounter 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

{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  }
}

```

(continues on next page)

(continued from previous page)

```

    },
    "biometricData": [
      {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "resolution": 1,
        "compression": "NONE"
      }
    ]
  }
}

```

**Example request:**

```

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

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

```

**Example response:**

```

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

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

```

**PUT /v1/persons/{personId}/encounters/{encounterId}****Update one encounter****Parameters**

- **personId** (*string*) – the id of the person
- **encounterId** (*string*) – the id of the encounter

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- **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/persons/{personId}/encounters/{encounterId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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}

Update one encounter 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 request:**

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

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

**Example response:**

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

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

**DELETE /v1/persons/{personId}/encounters/{encounterId}****Delete one encounter**

Delete one encounter from the person identified by his/her id. If this is the last encounter in the person, the person is also deleted.

**Parameters**

- **personId** (*string*) – the id of the person
- **encounterId** (*string*) – the id of the encounter

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 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}  
Delete one encounter 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 request:**

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

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

**Example response:**

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

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

**GET** /v1/persons/{personId}/encounters/{encounterId}/templates  
Read biometrics templates

**Parameters**

- **personId** (*string*) – the id of the person

- **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 (Required)
- **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
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 404 Not Found – Unknown record or unknown biometrics
- 500 Internal Server Error – Unexpected error

#### Example request:

```
GET /v1/persons/{personId}/encounters/{encounterId}/templates?transactionId=string 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",
    "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,
```

(continues on next page)



(continued from previous page)

```

    "message": "string"
  }

```

### Callback: readTemplateResponse

**POST** \${request.query.callback}  
 Read biometrics templates 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 request:

```

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

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

```

#### Example response:

```

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

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

```

**POST** /v1/persons/{personId}/encounters  
 Create one encounter and generate its ID

Create one encounter in the person identified by his/her id. If the person does not yet exist, it is created automatically.

#### Parameters

- **personId** (*string*) – the id of the person

#### Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)
- **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 – creation successful
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. The transaction ID is returned
- 400 Bad Request – Bad request
- 403 Forbidden – Creation not allowed
- 500 Internal Server Error – Unexpected error

#### Example request:

```
POST /v1/persons/{personId}/encounters?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "encounterId": "string",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "date": "2019-01-11"
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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

{
  "personId": "string",
```

(continues on next page)

(continued from previous page)

```

    "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: createResponse****POST** \${request.query.callback}

Create one encounter and generate its ID 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

{
  "personId": "string",
  "encounterId": "string"
}

```

**Example request:**

```

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

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

```

**Example response:**

```

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

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

```

## GET /v1/persons/{personId}/encounters

Read all encounters of one person

### Parameters

- **personId** (*string*) – the id of the person

### Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 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/persons/{personId}/encounters?transactionId=string HTTP/1.1
Host: example.com
```

### Example response:

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

[
  {
    "encounterId": "string",
    "encounterType": "string",
    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "date": "2019-01-11"
    },
    "biographicData": {
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA"
    },
    "biometricData": [
      {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "resolution": 1,

```

(continues on next page)

(continued from previous page)

```

        "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: readAllResponse**

**POST** \${request.query.callback}  
 Read all encounters 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

[
  {
    "encounterId": "string",
    "encounterType": "string",
    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "date": "2019-01-11"
    },
    "biographicData": {
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA"
    },
    "biometricData": [
      {
        "biometricType": "FACE",
        "biometricSubType": "UNKNOWN",
        "image": "c3RyaW5n",
        "imageRef": "https://example.com",
        "captureDate": "2020-04-28",

```

(continues on next page)

(continued from previous page)

```

        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "resolution": 1,
        "compression": "NONE"
      }
    ]
  }
}

```

**Example request:**

```

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

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

```

**Example response:**

```

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

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

```

## DELETE /v1/persons/{personId}

Delete a person and all its encounters

**Parameters**

- **personId** (*string*) – the id of the person

**Query Parameters**

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

```

(continues on next page)

(continued from previous page)

```
"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}

Delete a person 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 request:**

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

{
  "code": 1,
  "message": "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 (Required)
- **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
- **400 Bad Request** – Bad request
- **403 Forbidden** – Identification not allowed
- **500 Internal Server Error** – Unexpected error

### Example request:

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

{
  "filter": {
    "dateOfBirthMin": "1980-01-01",
    "dateOfBirthMax": "2019-12-31"
  },
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 1.0,
    "scoreList": [
      {

```

(continues on next page)



(continued from previous page)

```

        "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}  
**Biometric 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

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

```

**Example request:**

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

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

**Example response:**

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

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

**POST /v1/identify/{galleryId}/{personId}****Biometric identification based on existing data**

Identification based on existing data from one gallery

**Parameters**

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

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 400 Bad Request – Bad request
- 403 Forbidden – Identification not allowed
- 500 Internal Server Error – Unexpected error

**Example request:**

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

{
  "dateOfBirthMin": "1980-01-01",
  "dateOfBirthMax": "2019-12-31"
}
```

**Example response:**

```

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

[
  {
    "personId": "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}****Biometric identification based on existing data 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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 1.0,
    "scoreList": [
      {
        "score": 1.0,
        "encounterId": "string",
        "biometricType": "FACE",

```

(continues on next page)

(continued from previous page)

```

        "biometricSubType": "UNKNOWN"
      }
    ]
  }
]

```

**Example request:**

```

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

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

```

**Example response:**

```

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

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

```

**POST /v1/verify/{galleryId}/{personId}****Biometric verification**

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

**Parameters**

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

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 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}/{personId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "biometricData": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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}  
**Biometric 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 request:**

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

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

**Example response:**

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

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

**POST /v1/verify****Biometric verification with two sets of data**

Verification of two sets of biometric data

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 400 Bad Request – Bad request
- 403 Forbidden – Verification not allowed

- 500 Internal Server Error – Unexpected error

#### Example request:

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

{
  "biometricData1": [
    {
      "biometricType": "FACE",
      "biometricSubType": "UNKNOWN",
      "image": "c3RyaW5n",
      "imageRef": "https://example.com",
      "captureDate": "2020-04-28",
      "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": "2020-04-28",
      "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}**Biometric verification with two sets of data 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 request:**

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

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

**Example response:**

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

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



## Gallery

### GET /v1/galleries

Read the ID of all the galleries

#### Query Parameters

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 400 Bad Request – Bad request
- 403 Forbidden – Read not allowed
- 500 Internal Server Error – Unexpected error

Example request:

```
GET /v1/galleries?transactionId=string 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: readGalleriesResponse

POST \${request.query.callback}

Read the ID of all the galleries 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 request:**

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

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

**Example response:**

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

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

**GET** /v1/galleries/{galleryId}

Read the content of one gallery

**Parameters**

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

**Query Parameters**

- **transactionId** (*string*) – The id of the transaction (Required)
- **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
- 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}?transactionId=string HTTP/1.1
Host: example.com
```

**Example response:**

```

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

[
  {
    "personId": "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: readGalleryContentResponse****POST** \${request.query.callback}

Read the content of one gallery 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

[
  {
    "personId": "string",
    "encounterId": "string"
  }
]

```

**Example request:**

```

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

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

```

**Example response:**

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

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

---

**Data Model**

To be completed

**7.3.6 Third Party Services****Services**

To be defined

---

## List of Tables

---

2.1	Components . . . . .	6
2.2	Interfaces List . . . . .	8
2.3	Components vs Interfaces Mapping . . . . .	10
5.1	Event Type . . . . .	19
5.2	Person Attributes . . . . .	23
5.3	Certificate Attributes . . . . .	23
5.4	Union Attributes . . . . .	23
5.5	Filiation Attributes . . . . .	23
5.6	Document Type . . . . .	23
5.7	Enrolment Data Model . . . . .	26
5.8	Population Registry Data Model . . . . .	30
5.9	Biometric Services Options . . . . .	36
5.10	Biometric Data Model . . . . .	36
5.11	Enrolment Data Model . . . . .	45
6.1	Enrolment Data Model . . . . .	48
6.2	Event Type . . . . .	50
6.3	Person Attributes . . . . .	54
6.4	Certificate Attributes . . . . .	54
6.5	Union Attributes . . . . .	54
6.6	Filiation Attributes . . . . .	54
6.7	Document Type . . . . .	54
6.8	Population Registry Data Model . . . . .	59
6.9	Event Type . . . . .	62
6.10	Person Attributes . . . . .	66
6.11	Certificate Attributes . . . . .	66
6.12	Union Attributes . . . . .	66
6.13	Filiation Attributes . . . . .	66
6.14	Document Type . . . . .	66
6.15	Biometric Services Options . . . . .	72
6.16	Biometric Data Model . . . . .	72
7.1	Event Type & Message . . . . .	83
7.2	Person Attributes . . . . .	91
7.3	Matching Error Object . . . . .	92
7.4	Expression Object . . . . .	92
7.5	Error Object . . . . .	92

---

## List of Figures

---

1.1	The dependency challenge	2
2.1	Components identified as part of the identity ecosystem	7
2.2	Birth Use Case	12
2.3	Deduplication Use Case	13
2.4	Bank account opening Use Case	14
2.5	Collaborative identity control	14
5.1	Subscription & Notification Process	18
5.2	readPersonAttributes Sequence Diagram	20
5.3	matchPersonAttributes Sequence Diagram	20
5.4	verifyPersonAttributes Sequence Diagram	21
5.5	queryPersonUIN Sequence Diagram	21
5.6	queryPersonList Sequence Diagram	22
5.7	readDocument Sequence Diagram	22
5.8	generateUIN Sequence Diagram	24
5.9	Population Registry Data Model	31
5.10	Biometric Data Model	37
6.1	Subscription & Notification Process	49
6.2	readPersonAttributes Sequence Diagram	51
6.3	matchPersonAttributes Sequence Diagram	51
6.4	verifyPersonAttributes Sequence Diagram	52
6.5	queryPersonUIN Sequence Diagram	52
6.6	queryPersonList Sequence Diagram	53
6.7	readDocument Sequence Diagram	53
6.8	Population Registry Data Model	60
6.9	Subscription & Notification Process	61
6.10	readPersonAttributes Sequence Diagram	63
6.11	matchPersonAttributes Sequence Diagram	63
6.12	verifyPersonAttributes Sequence Diagram	64
6.13	queryPersonUIN Sequence Diagram	64
6.14	queryPersonList Sequence Diagram	65
6.15	readDocument Sequence Diagram	65
6.16	generateUIN Sequence Diagram	67
6.17	Biometric Data Model	73

---

## HTTP Routing Table

---

### /\${request.query.callback}

POST \${request.query.callback}, 136

### /v1

GET /v1/galleries, 105

GET /v1/galleries/{galleryId}, 106

GET /v1/persons, 87

GET /v1/persons/{personId}, 93

GET /v1/persons/{personId}/encounters, 121

GET /v1/persons/{personId}/encounters/{encounterId}, 111

GET /v1/persons/{personId}/encounters/{encounterId}/templates, 116

GET /v1/persons/{personId}/identities, 95

GET /v1/persons/{personId}/identities/{identityId}, 99

GET /v1/persons/{personId}/reference, 104

GET /v1/persons/{uin}, 88

GET /v1/persons/{uin}/document, 90

GET /v1/subscriptions, 81

GET /v1/subscriptions/confirm, 82

GET /v1/topics, 78

POST /v1/identify/{galleryId}, 124

POST /v1/identify/{galleryId}/{personId}, 127

POST /v1/persons, 107

POST /v1/persons/{personId}, 92

POST /v1/persons/{personId}/encounters, 118

POST /v1/persons/{personId}/encounters/{encounterId}, 109

POST /v1/persons/{personId}/identities, 96

POST /v1/persons/{personId}/identities/{identityId}, 97

POST /v1/persons/{uin}/match, 89

POST /v1/persons/{uin}/verify, 89

POST /v1/subscriptions, 80

POST /v1/topics, 78

POST /v1/topics/{uuid}/publish, 79

POST /v1/uin, 86

POST /v1/verify, 131

POST /v1/verify/{galleryId}/{personId}, 129

PUT /v1/persons/{personId}, 93

PUT /v1/persons/{personId}/encounters/{encounterId}, 113

PUT /v1/persons/{personId}/identities/{identityId}, 100

PUT /v1/persons/{personId}/identities/{identityId}, 103

PUT /v1/persons/{personId}/identities/{identityId}, 103

DELETE /v1/persons/{personId}, 94

DELETE /v1/persons/{personId}/encounters/{encounterId}, 115

DELETE /v1/persons/{personId}/identities/{identityId}, 102

DELETE /v1/subscriptions/{uuid}, 82

DELETE /v1/topics/{uuid}, 79

PATCH /v1/persons/{personId}/identities/{identityId}, 101

### /\${request.query.address}

POST \${request.query.address}, 81

## A

ABIS, [77](#)

## C

CMS, [77](#)

CR, [77](#)

Credential, [77](#)

## E

Encounter, [77](#)

## F

Functional systems and registries, [77](#)

## H

HTTP Status Codes, [77](#)

## M

Mime Types, [77](#)

## O

OSIA, [78](#)

## P

PR, [78](#)

## R

RFC

RFC 7396, [28](#), [56](#)

## U

UIN, [78](#)