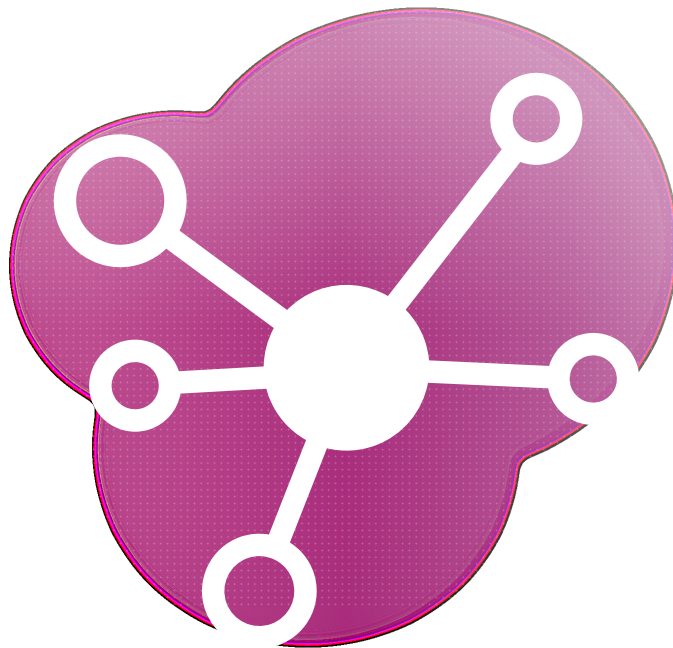




Specifications version 6.0.0



CONTENTS

1	Introduction	1
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
2	Functional View	5
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	12
2.4.1	Birth Use Case	12
2.4.2	Death Use Case	13
2.4.3	Deduplication Use Case	14
2.4.4	ID Card Request Use Case (1)	14
2.4.5	ID Card Request Use Case (2)	16
2.4.6	Credentials Issuance Use Case	16
2.4.7	Bank account opening Use Case	18
2.4.8	Police identity control Use Case	18
2.4.9	Telco Customer Enrollment with ID document	19
2.4.10	Telco Customer Enrollment with no ID document	20
3	Security & Privacy	21
3.1	Introduction	21
3.2	Virtual UIN	21
3.3	Authorization	21
3.3.1	Principles	21
3.3.2	Rules	22
3.3.3	Scopes	22
3.3.4	REST Interface Implementation	24
3.4	Privacy by Design	24
3.4.1	Privacy for end-to-end systems	25
3.4.2	PII actors	25
3.4.3	Data subject rights	26
3.4.4	What should OSIA API implementors do to prepare for safe PII?	28
4	OSIA Versions & Referencing	29
5	Interfaces	30
5.1	Notification	30
5.1.1	Services	31
5.1.2	Dictionaries	32
5.2	Data Access	33

5.2.1	Services	33
5.2.2	Dictionaries	36
5.3	UIN Management	37
5.3.1	Services	37
5.4	Enrollment Services	38
5.4.1	Services	38
5.4.2	Attributes	41
5.4.3	Transaction ID	41
5.4.4	Data Model	41
5.5	Population Registry Services	42
5.5.1	Services	42
5.5.2	Data Model	48
5.6	Biometrics	49
5.6.1	Services	50
5.6.2	Options	55
5.6.3	Data Model	56
5.7	Credential Services	57
5.7.1	Services	57
5.7.2	Attributes	60
5.7.3	Data Model	60
5.8	ID Usage	61
5.8.1	Relying Party API	62
5.8.2	Attribute set	63
5.8.3	Attribute set name	63
5.8.4	Output Attribute set	63
6	Components	64
6.1	Enrollment Component	64
6.1.1	Enrollment Services	64
6.2	Population Registry	69
6.2.1	Notification	69
6.2.2	Data Access	71
6.2.3	Population Registry Services	76
6.3	Civil Registry	82
6.3.1	Notification	82
6.3.2	Data Access	85
6.4	UIN Generator	90
6.4.1	UIN Management	90
6.5	ABIS	90
6.5.1	Biometrics	90
6.6	Credential Management System	98
6.6.1	Credential Services	98
6.7	ID Usage	102
6.7.1	ID Usage	102
7	Annexes	105
7.1	Glossary	105
7.2	Data Format	106
7.3	Technical Specifications	106
7.3.1	Notification	106
7.3.2	UIN Management	116
7.3.3	Data Access	118
7.3.4	Enrollment	124
7.3.5	Population Registry Management	144
7.3.6	Biometrics	173
7.3.7	Credential Services	232
7.3.8	ID Usage Services	252

INTRODUCTION

1.1 Problem Statement: vendor lock-in

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

The lack of 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.

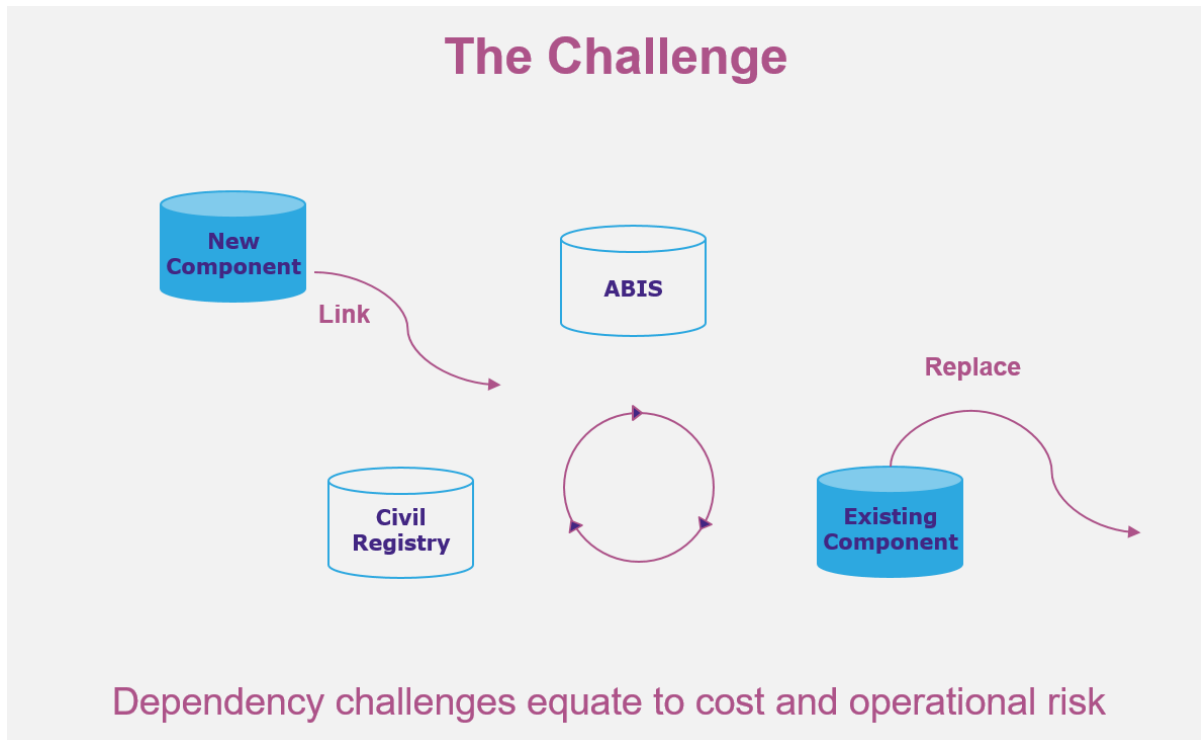


Fig. 1.1: The dependency challenge

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 SIA 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.

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

1.6 Revision History

Table 1.1: Revision History

OSIA Release	1.0.0	2.0.0	3.0.0	4.1.0	5.0.0	6.0.0
OSIA Release Date	mar-2019	jun-2019	nov-2019	jul-2020	dec-2020	dec-2021
Notification	.	1.0.0	1.0.0	1.1.0	1.2.0	1.2.0
UIN Management	1.0.0	1.0.0	1.0.0	1.1.0	1.2.0	1.2.0
Data Access	1.0.0	1.0.0	1.0.0	1.1.0	1.3.0	1.3.0
Enrollment Services	.	.	.	1.0.0	1.1.0	1.2.0
Population Registry Services	.	.	1.0.0	1.2.0	1.3.0	1.4.0
Biometrics Services	.	1.0.0	1.1.0	1.3.0	1.4.0	1.5.0
Credential Services	.	.	.	1.0.0	1.1.0	1.2.0
Relying Party Services	1.0.0	1.1.0

FUNCTIONAL VIEW

2.1 Components: Standardized Definition and Scope

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

The components are defined as follows:

- The *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.”¹

- The *UIN Generator* component.

UIN generator is defined as a system to generate and manage unique identifiers.

- The *Automated Biometric Identification System* (ABIS) component.

An ABIS is defined as a *system to detect the identity of an individual when it is unknown, or to verify the individual's identity when it is provided, through biometrics.*

- The *Civil Registry* (CR) component.

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

- 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.

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

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

This component interfaces with external systems to leverage the identity databases for the benefits of individuals. It provides services to authenticate, identify, and access identity attributes for use cases such as KYC.

- The *Digital Credential Issuance & Distribution System*.

This component is in charge of the issuance and delivery of the digital credentials built from the identity databases under the control of the CMS.

Table 2.1: Components

ID Ecosystem Component	Data	Functions
Enrollment	<ul style="list-style-type: none"> • Biographic data • UIN • History • Supporting documents 	<ul style="list-style-type: none"> • Recording application • Collecting personal data
PR	<ul style="list-style-type: none"> • Biographic data • UIN • History • Supporting documents 	<ul style="list-style-type: none"> • Identity attributes storage • Identity Life cycle management
UIN Gen	<ul style="list-style-type: none"> • Biographic data • UIN 	<ul style="list-style-type: none"> • UIN generation
ABIS	<ul style="list-style-type: none"> • UIN • Biometric data (images and templates) 	<ul style="list-style-type: none"> • Authentication (1:1) • Identification (1:N) • Quality control and adjudication
CR	<ul style="list-style-type: none"> • Events • UIN • History • Supporting documents 	<ul style="list-style-type: none"> • Events storage • Certificate production • Workflow
CMS	<ul style="list-style-type: none"> • Biographic data • UIN • Biometric data • Credential event history 	<ul style="list-style-type: none"> • Credential data storage • Credential Life cycle management • Credential Production • Workflow
Third Party Services	<ul style="list-style-type: none"> • Biographic data/Identity attributes • Biometric data 	<ul style="list-style-type: none"> • Authentication (1:1) • Identification (1:N) • Access to identity attributes
Digital Credential Issuance & Distribution System		<ul style="list-style-type: none"> • Issuance of a digital credential • Delivery of a digital credential

The components are represented on the following diagram:

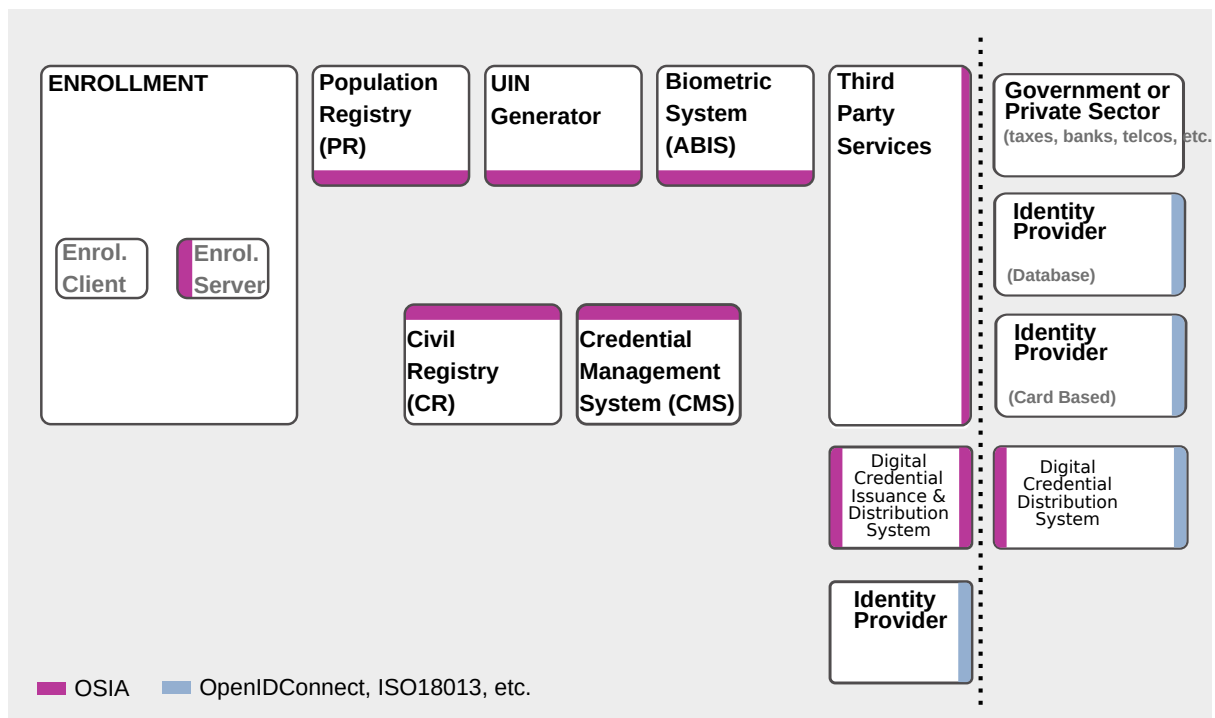


Fig. 2.1: Components identified as part of the identity ecosystem

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.

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

Table 2.2: Interfaces List

Services	Description
Notification	
Subscribe	Subscribe a URL to receive notifications sent to one topic
List Subscription	Get the list of all the subscriptions registered in the server
Unsubscribe	Unsubscribe a URL from the list of receiver for one topic
Confirm	Confirm that the URL used during the subscription is valid
Create Topic	Create a new topic
List Topics	List all the existing topics
Delete Topic	Delete a topic
Publish	Publish an event to all systems that have subscribed to this topic
Notify	Callback registered during subscription and called when an event is published
Data Access	
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
UIN Management	
Generate UIN	Generate a new UIN
Enrollment Services	
Create Enrollment	Insert a new enrollment
Read Enrollment	Retrieve an enrollment
Update Enrollment	Update an enrollment
Partial Update Enrollment	Update part of an enrollment
Finalize Enrollment	Finalize an enrollment (mark it as completed)
Delete Enrollment	Delete an enrollment
Find Enrollments	Retrieve a list of enrollments which match passed in search criteria
Send Buffer	Send a buffer (image, etc.)
Get Buffer	Get a buffer
Population Registry Services	
Find Persons	Query for persons, using all the available identities
Create Person	Create a new person
Read Person	Read the attributes of a person
Update Person	Update a person

continues on next page

Table 2.2 – continued from previous page

Services	Description
Delete Person	Delete a person and all its identities
Merge Persons	Merge two persons
Move Identity	Move one identity from one person to another one
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
Biometrics	
Create Encounter	Create a new encounter. No identify is performed
Read Encounter	Read the data of an encounter
Update Encounter	Update an encounter
Delete Encounter	Delete an encounter
Merge Encounters	Merge two sets of encounters
Move Encounter	Move one encounter from one person to another one
Update Encounter Status	Set an encounter status
Update Encounter Galleries	Set the galleries of 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
Credential Services	
Create Credential Request	Request issuance of a secure credential
Read Credential Request	Retrieve the data/status of a credential request
Update Credential Request	Update the requested issuance of a secure credential
Cancel Credential Request	Cancel the requested issuance of a secure credential
Find Credentials	Retrieve a list of credentials that match the passed in search criteria
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
Revoke Credential	Revoke an issued credential. For electronic credentials this will revoke any PKI certificates that are present
Set Credential Status	Change the credential status
Find Credential Profiles	Retrieve a list of credential profiles that match the passed in search criteria
ID Usage	
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

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	Enroll Clt	Enroll Srv	PR	UIN Gen	ABIS	CR	CMS	ID Us- age
Notification								
Subscribe			U		U	U	U	
List Subscription			U		U	U	U	
Unsubscribe			U		U	U	U	
Confirm			U		U	U	U	
Create Topic			U		U	U	U	
List Topics			U		U	U	U	
Delete Topic			U		U	U	U	
Publish			U		U	U	U	
Notify			I		I	I	I	
Data Access								
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		U
Query Person List						U		U
Read Document		U	IU			IU		U
UIN Management								
Generate UIN			U	I		U		
Enrollment Services								
Create Enrollment	U	I						
Read Enrollment	U	I						
Update Enrollment	U	I						
Partial Update Enrollment	U	I						
Finalize Enrollment	U	I						
Delete Enrollment	U	I						
Find Enrollments	U	I						
Send Buffer	U	I						
Get Buffer	U	I						
Population Registry Services								
Find Persons			I					
Create Person			I		U		U	
Read Person			I		U		U	U
Update Person			I		U		U	
Delete Person			I		U		U	
Merge Persons			I		U			
Move Identity			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					

continues on next page

Table 2.3 – continued from previous page

Interfaces	Enroll Clt	Enroll Srv	PR	UIN Gen	ABIS	CR	CMS	ID Usage
Read Galleries			I					
Read Gallery Content			I					
Biometrics								
Create Encounter		U	U		I			
Read Encounter		U	U		I			U
Update Encounter		U	U		I			
Delete Encounter		U	U		I			
Merge Encounters			U		I			
Move Encounters			U		I			
Update Encounter Status		U	U		I			
Update Encounter Galleries		U	U		I			
Read Template		U	U		I			
Read Galleries								
Read Gallery Content		U	U		I			
Identify		U			I			U
Verify		U			I			U
Credential Services								
Create Credential Request							I	
Read Credential Request							I	
Update Credential Request							I	
Cancel Credential Request							I	
Find Credentials							I	
Read Credential							I	
Suspend Credential							I	
Unsuspend Credential							I	
Revoke Credential							I	
Set Credential Status							I	
Find Credential Profiles							I	
ID Usage								
Verify ID								I
Identify ID								I
Read Attributes								I
Read Attributes set								I

where:

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

2.4 Use Cases - How to Use OSIA

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

2.4.1 Birth Use Case

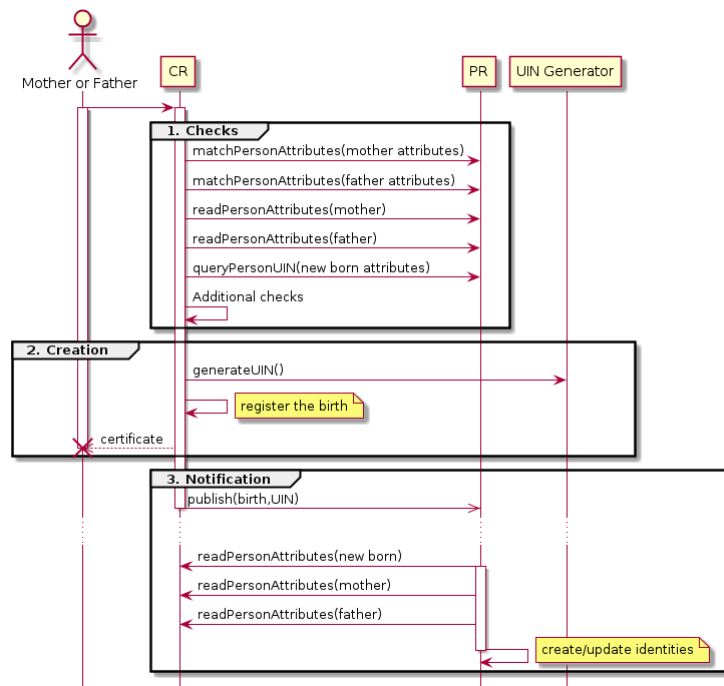


Fig. 2.2: Birth Use Case

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

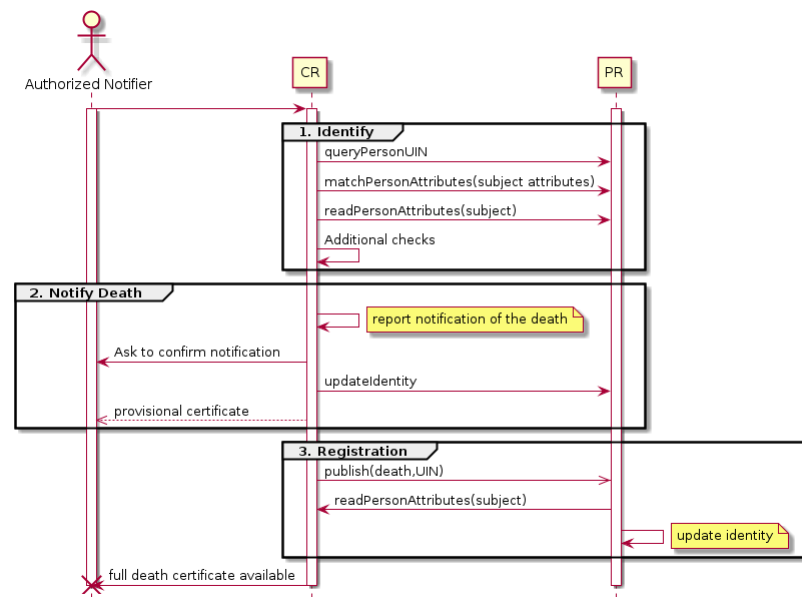


Fig. 2.3: Death Use Case

1. Subject identification checks

When a death notification is submitted by an authorized party, the CR shall run checks against the data available in the PR using:

- `matchPersonAttributes`: to check the exactitude of the subject's attributes as known in the PR
- `readPersonAttributes`: to get missing data about the subject's identity that
- `queryPersonUIN`: to check if the person 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. The CR may implement an internal procedure to create a valid PR record retrospectively.

2. Notification creation

The first step after the identity checks is to notify the life event status to the PR based on an identified record. At this point the death notification is recorded by not finally registered. Most states implement a waiting period. How the CR will process the death notification is specific to each CR implementation - a provisional certificate is possible.

3. Final registration

When the PR finalizes the status of the subject's person record then the CR may publish this information at its discretion. The PR may maintain a list of interested parties who shall be informed of any finalized death status. A final certificate of death including the context of this event is typically issued by the CR to the notifier for distribution.

2.4.3 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.

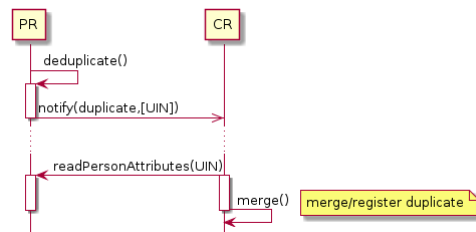


Fig. 2.4: Deduplication Use Case

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

2.4.4 ID Card Request Use Case (1)

An ID card is one type of credential. The procedures surrounding credential issuance may involve several subsystems that contribute to the establishment of the applicant identity and the required data for the type of credential.

This use case assumes a simple starting scenario where the identity is known and can be validated, mostly with data available from a Civil or Population Registry based Identity Provider. These use cases also assume the use of a Credentials Management System (CMS) responsible for the technical personalization and lifecycle management of a credential such as an ID card.

The use case aims to show how a selection of the CMS API calls can support a typical, use case in relation to CMS usage.

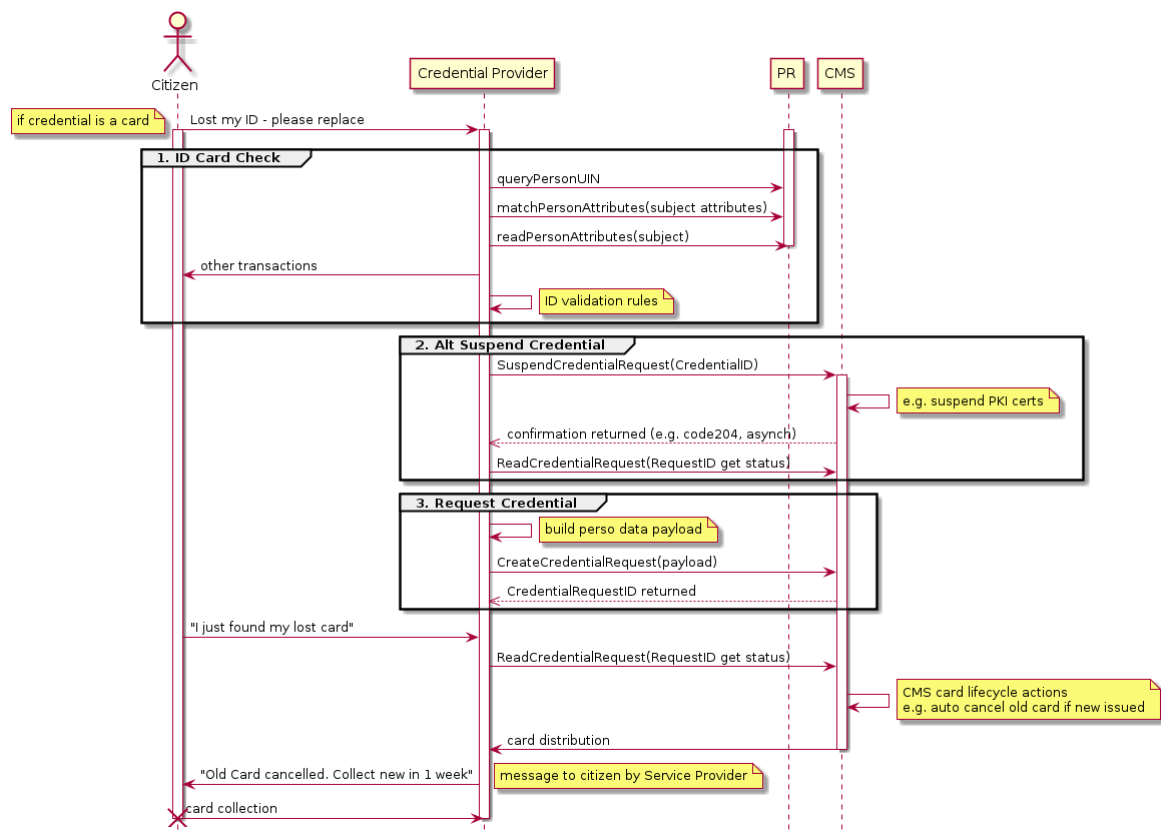


Fig. 2.5: ID Card Request Use Case (1)

1. Identity Checks

The example scenario assumes a credential provider service such as an ID card provider (National ID, Voter, &c). Such a service may access several OSIA API based components to establish an ID check. In this example the Population Register is used. This example case also assumes that the credential provider holds its own register of credentials issued to its subscribers.

2. Suspend Credential

In the example above the citizen has lost a card and requests a replacement. The credential provider service first establishes the legitimacy of the citizen identity and the identity of the lost document within its own register. The next likely step in such a use case is to suspend the current credential. This is done using a CMS API call. The CMS confirms this step with a reference. In some use cases the reported lost credential may be cancelled immediately, but this is typically a decision made by the policy of the credential provider. There is an OSIA API call to both either or both requirements to the CMS.

3. Requesting a New Credential

The credential provider is in this example case responsible for preparing the core document data for the CMS. The CMS itself may further process this data appropriate to the credential type: for example the CMS may be the service that signs this document data electronically. The CMS returns a new request ID to the credential provider service which will enable the provider to query credential production status within the CMS domain.

Such a business process might be interrupted by an new event such as the citizen finding her lost card and wishing to cancel the replacement order, perhaps to avoid a replacement fee. Depending on the status returned by the CMS to the credential provider then the credential provider service will act accordingly in informing the citizen whether this is possible. In this case the citizen's card was already replaced by the CMS so the original card is now cancelled.

The CMS on its side is responsible for maintaining a credential profile which can be accessed by the CR at a later point. This use case stops for CMS when the card is distributed to the CR for collection by the citizen.

2.4.5 ID Card Request Use Case (2)

A second ID Request use case shows how the CMS might expose more decisions to the credential providing service. In this case it is the citizen facing provider that controls the cancellation of the lost document, and this is not automated within the CMS component.

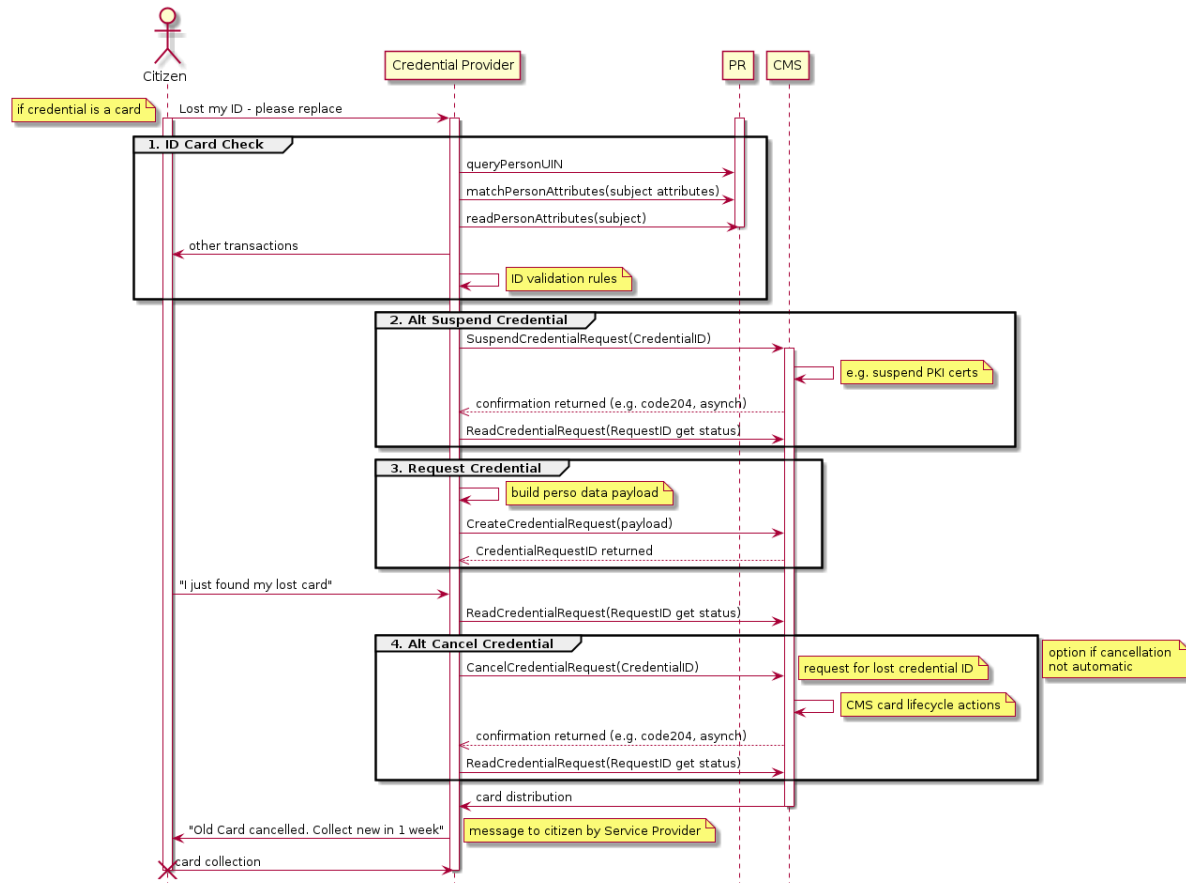


Fig. 2.6: ID Card Request Use Case (2)

This second example shows how APIs may be used to flex the control over functions such as credential lifecycle management. This example first makes use of the API to suspend a credential pending production of a replacement; then a second API call is made to the CMS to instruct cancellation of the lost document.

2.4.6 Credentials Issuance Use Case

This use case describes an example of interaction between the different OSIA components to capture identity data, generate a UIN, process the identity data and issue both a physical and digital credential.

This use case also demonstrates what a middleware could do when connected to multiple OSIA compatible systems. In this example the middleware is acting as an enrollment server, scheduling all the processing when the data collection is finalized.

This use case was implemented for demonstrating OSIA and is presented in this [video](#).

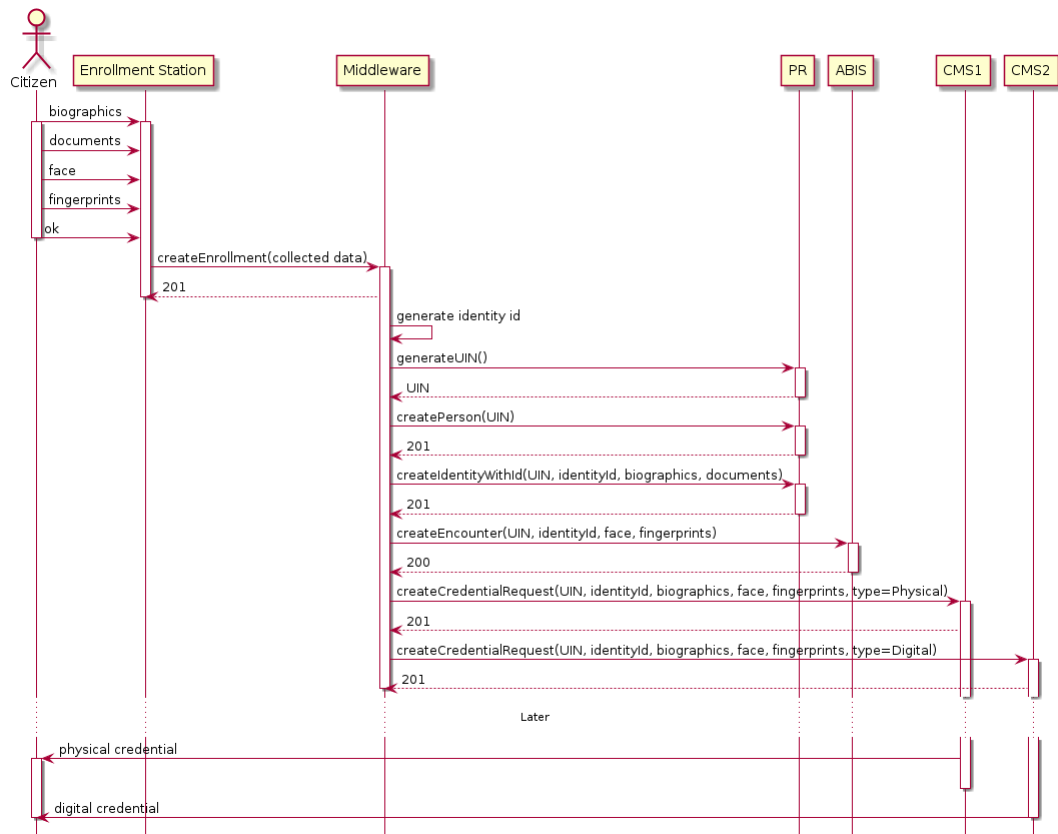


Fig. 2.7: Credentials Issuance Use Case

The main steps are:

1. The citizen interacts with the enrollment station to provide the biographic data, the supporting document images, a portrait and a set of fingerprints.
2. When all the data is collected, the full data set is pushed to the middleware using the OSIA `createEnrollment` service.
3. Backend processing includes:
 - interactions with the population registry to generate a UIN and insert the collected data,
 - interaction with the ABIS to insert the face and fingerprints,
 - interactions with multiple Credential Management System to request the issuance of different types of credentials.

2.4.7 Bank account opening Use Case

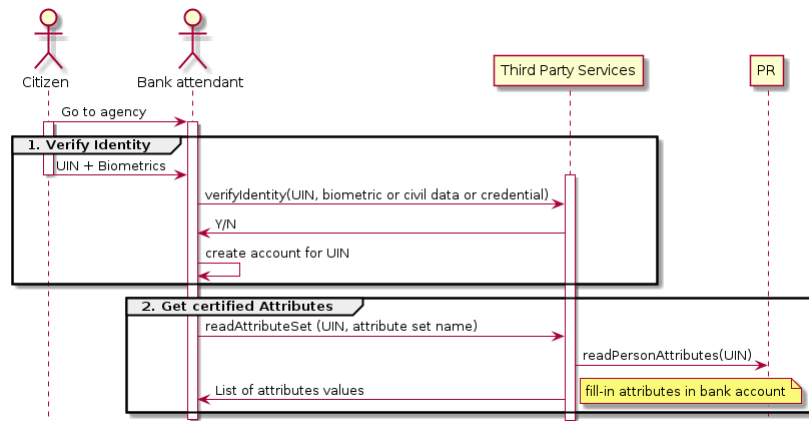


Fig. 2.8: Bank account opening Use Case

2.4.8 Police identity control Use Case

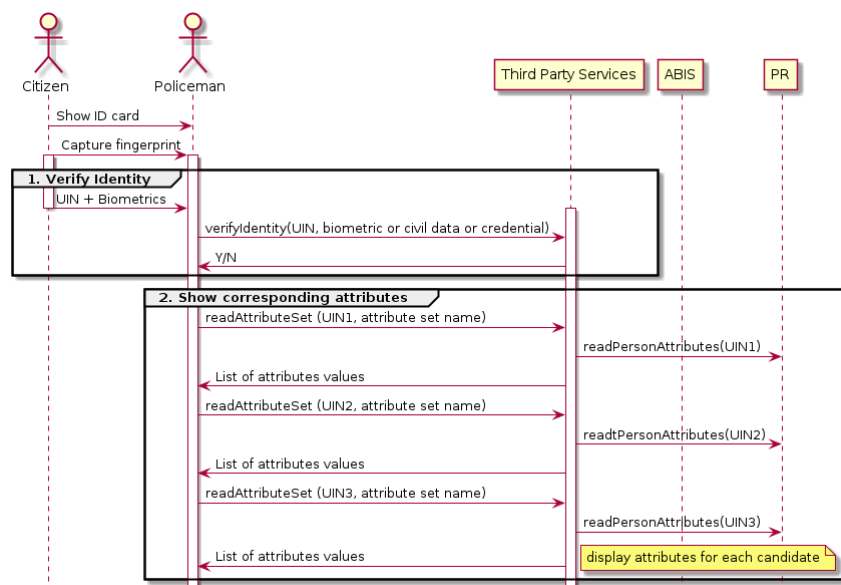


Fig. 2.9: Collaborative identity control

2.4.9 Telco Customer Enrollment with ID document

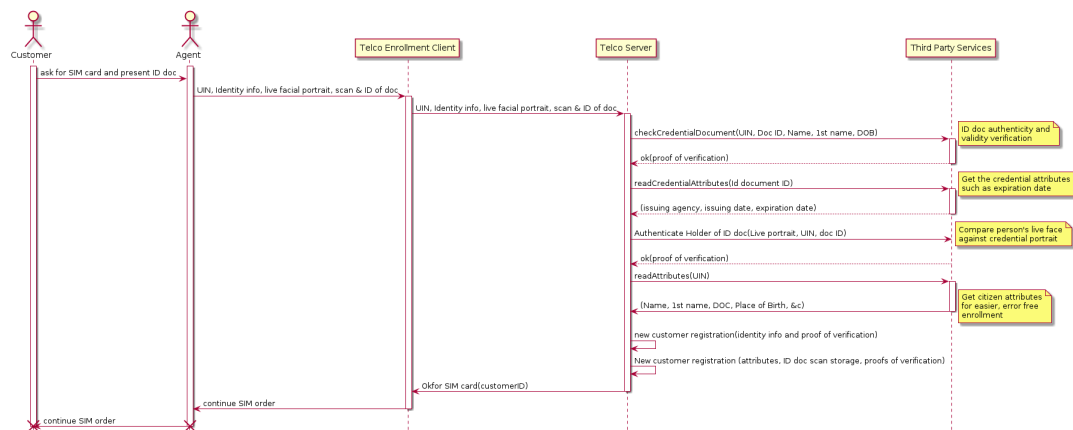


Fig. 2.10: Telco Customer Enrollment with ID document

1. Use case objective

This use case allows a telco operator to check a citizen's ID document and identity.

The use case relies on an IDMS to check the authenticity and validity of the ID document presented by the citizen, then to check that he actually is the holder of the document.

2. Pre-conditions

The citizen is registered in the IDMS and has a UIN.

The citizen has a valid ID document.

The citizen presents as a customer to the agent.

The IDMS should support authentication token generation to protect against misuse of UIN.

3. Use case description

The customer shows his ID document to the Agent. The Agent inputs (possibly by reading an MRZ on the document) the UIN, document ID, name, given name, DOB, and a live facial portrait taken of the citizen.

The telco server calls an IDMS API to check if the information of the ID document is coherent and if the document is still valid.

The telco server calls an IDMS API to get meta data of the document such as the issuing agency, the issuing date, expiration date, etc.

The telco server calls an IDMS API to check if the customer is actually the holder of the document using his live biometric portrait.

The telco server calls an IDMS API to get some reliable data of the customer in order to register him.

4. Result

The citizen is now identified, authenticated and registered in a customer database and becomes eligible to buy a SIM card.

The telco operator can prove regulatory controls have been applied for 'Know Your Customer' compliance.

2.4.10 Telco Customer Enrollment with no ID document

A customer applying for a new network SIM card may not be able to present an ID document as part of her application.

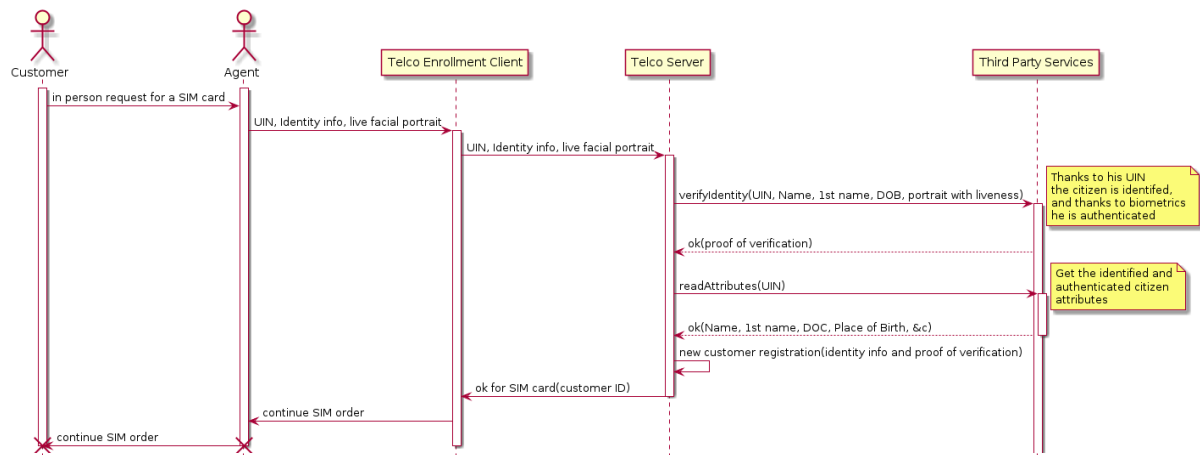


Fig. 2.11: Telco Customer Enrollment with no ID document

1. Use case objective

This use case allows a telco operator to check a citizen's identity and get his attributes relying on IDMS to check that the biometrics of the citizen matches with his UIN.

2. Pre-conditions

The citizen is registered in the IDMS and has a UIN.

The citizen biometrics are registered and associated to his UIN.

The citizen presents as a customer to the agent.

The IDMS should support authentication token generation to protect against misuse of UIN.

3. Use case description

The Agent inputs the citizen's UIN, Name, 1st Name, DOB and takes a live photo portrait of the customer.

The telco server calls an IDMS API to check if the customer is actually the citizen corresponding to the given UIN thanks to his live portrait (face biometric matching).

The telco server calls an IDMS API to get some reliable data of the customer in order to register him.

4. Result

The citizen is now identified, authenticated and registered in a customer database and becomes eligible to buy a SIM card.

The telco operator can prove regulatory controls have been applied for 'Know Your Customer' compliance.

SECURITY & PRIVACY

3.1 Introduction

Insert diagram of security & privacy features

3.2 Virtual UIN

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

3.3 Authorization

Because OSIA is a set of interfaces/API and not a full system, this chapter describes only how to secure those API, through the usage of standard JWT, and not how to generate and protect such tokens, nor how to secure the full system.

Securing the API is one mandatory step on the way to a secure system, but securing a full system includes more than just that: hardware & software components, processes & methodology, audit, etc. that are not in the scope of this document.

3.3.1 Principles

Securing OSIA services is implemented with the following principles:

- Rely on JWT: “JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties” It can be “digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted”. [RFC 7519]
- Tokens of type “Bearer Token” are used. [RFC 6750] The generation and management of those tokens are not in the scope of this document.
- Validating the token is the responsibility of the service implementation, with the help of components not described in this document (PKI, authorization server, etc.)
- The service implementations are responsible for extracting information from the token and give access or not to the service according to the claims contained in the token and the *scope* defined for each service in this document.
- The service implementations are free to change the security scheme used, for instance to use OAuth2 or OpenID Connect, if it fits the full system security policy. **Scopes must not be changed.**
- All HTTP exchanges must be secured with TLS. Mutual authentication is not mandatory.

Note: The added use of peer-to-peer payload encryption - e.g. to protect biometric data - is not in the scope of this document.

Note: OSIA does not define ACL (Access Control List) to protect the access to a subset of the data. This may be added in a future version.

Warning: Bearer tokens are sensitive and subject to security issues if not handled properly. Please refer to [JSON Web Token Best Current Practices](#) for advice on proper implementation.

3.3.2 Rules

All scopes are named according to the following rules:

`application[.resource].action`

where:

- `application` is a key identifying the interface group listed in [Interfaces](#). Examples: `notif`, `pr`, `cr`, `abis`, etc.
- `resource` is a key identifying the resource. Examples: `person`, `encounter`, `identity`, etc.
- `action` is one of:
 - `read`: for read access to the data represented by the *resource* and managed by the *application*.
 - `write`: for creating, updating or deleting the data.
 - or another value, for specific actions such as `match` or `verify` that need to be distinguished from a general purpose read or write for proper segregation.

Scopes should be less than 20 characters when possible to limit the size of the bearer token.

3.3.3 Scopes

The following table is a summary of all scopes defined in OSIA.

Table 3.1: Scopes List

Services	Scope
Notification	
Subscribe	<code>notif.sub.write</code>
List Subscription	<code>notif.sub.read</code>
Unsubscribe	<code>notif.sub.write</code>
Confirm	<code>notif.sub.write</code>
Create Topic	<code>notif.topic.write</code>
List Topics	<code>notif.topic.read</code>
Delete Topic	<code>notif.topic.write</code>
Publish	<code>notif.topic.publish</code>
Notify	N/A
Data Access	
Read Person Attributes	<code>pr.person.read</code> or <code>cr.person.read</code>
Match Person Attributes	<code>pr.person.match</code> or <code>cr.person.match</code>
Verify Person Attributes	<code>pr.person.verify</code> or <code>cr.person.verify</code>
Query Person UIN	<code>pr.person.read</code> or <code>cr.person.read</code>
Query Person List	<code>pr.person.read</code> or <code>cr.person.read</code>

continues on next page

Table 3.1 – continued from previous page

Services	Scope
Read document	pr.document.read or cr.document.read
UIN Management	
Generate UIN	uin.generate
Enrollment Services	
Create Enrollment	enroll.write
Read Enrollment	enroll.read
Update Enrollment	enroll.write
Partial Update Enrollment	enroll.write
Finalize Enrollment	enroll.write
Delete Enrollment	enroll.write
Find Enrollments	enroll.read
Send Buffer	enroll.buf.write
Get Buffer	enroll.buf.read
Population Registry Services	
Find Persons	pr.person.read
Create Person	pr.person.write
Read Person	pr.person.read
Update Person	pr.person.write
Delete Person	pr.person.write
Merge Persons	pr.person.write
Move Identity	pr.identity.write
Create Identity	pr.identity.write
Read Identity	pr.identity.read
Update Identity	pr.identity.write
Partial Update Identity	pr.identity.write
Delete Identity	pr.identity.write
Set Identity Status	pr.identity.write
Define Reference	pr.reference.write
Read Reference	pr.reference.read
Read Galleries	pr.gallery.read
Read Gallery Content	pr.gallery.read
Biometrics	
Create Encounter	abis.encounter.write
Read Encounter	abis.encounter.read
Update Encounter	abis.encounter.write
Delete Encounter	abis.encounter.write
Merge Encounters	abis.encounter.write
Move Encounter	abis.encounter.write
Update Encounter Status	abis.encounter.write
Update Encounter Galleries	abis.encounter.write
Read Template	abis.encounter.read
Read Galleries	abis.gallery.read
Read Gallery content	abis.gallery.read
Identify	abis.identify
Verify	abis.verify
Credential Services	
Create Credential Request	cms.request.write
Read Credential Request	cms.request.read
Update Credential Request	cms.request.write
Cancel Credential Request	cms.request.write
Find Credentials	cms.credential.read
Read Credential	cms.credential.read
Suspend Credential	cms.credential.write

continues on next page

Table 3.1 – continued from previous page

Services	Scope
Unsuspend Credential	cms.credential.write
Revoke Credential	cms.credential.write
Set Credential Status	cms.credential.write
Find Credential Profiles	cms.profile.read
ID Usage (Work in progress)	
Verify ID	id.verify
Identify	id.identify
Read Attributes	id.read
Read Attributes set	id.SET_NAME.read

3.3.4 REST Interface Implementation

The [OpenAPI](#) files included in this document must be changed to:

1. Define the [security scheme](#). This is done with the additional piece of code:

```
components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

2. Apply the security scheme and define the scope (i.e. permission) for each service. Example:

```
paths:
  /yyy:
    get:
      security:
        - BearerAuth: [id.read] # List of scopes
      responses:
        '200':
          description: OK
        '401':
          description: Not authenticated (bad token)
        '403':
          description: Access token does not have the required scope
```

See the different YAML files provided in [Technical Specifications](#).

3.4 Privacy by Design

Privacy by design is a founding principle of the OSIA initiative.

The OSIA API is designed to support the protection of private citizens' Personal Identifiable Information (PII).

The protection of PII data is a central design concern for all identity based systems regardless of where these are based.

PII data does not recognize geographical boundaries; it moves across systems and jurisdictions. Similarly, the OSIA initiative is not geographically limited. OSIA takes its strong reference point from the European Union's GDPR regulation because this is considered by many as a best practice approach. GDPR anticipates the possible adverse consequences from the mobility of PII whether inside or outside the EU.

The General Data Protection Regulation (GDPR) is quite recent. It was introduced across the EU in 2016, before reaching its full legal effect in 2018. It is adopted by all EU governments and carries direct regulatory and legal force for any organization handling Personal Identifiable Information (PII), either in the EU or in connection with EU citizens or residents. Compliance failure in respect of GDPR carries significant financial penalties, reflecting the rights of individuals and groups, as well as the importance of the issue.

GDPR is not the only defined standard, but it is seen as a best practice one. It is exemplary approach for the safeguarding of PII; but, it should also be seen as a safeguard for a system owner/operator's interests. It is a major driver for government leadership in Identity Management is to prevent identity fraud.

3.4.1 Privacy for end-to-end systems

For privacy the bigger goal is to protect PII across the full reach of ID systems. The OSIA API is a fundamental part and principle of the building process, providing definitions of how components are connected.

This is a part of a wider story. An end-to-end solution making use of the OSIA API should address three specific areas of concern for PII.

Correct implementation of the API definition

PII data flows through systems. API based connectivity between functional components is by definition a way of sharing information, which will focus mostly on PII. The OSIA API defines what should happen between application endpoints involving OSIA framework components. It defines content and a minimum acceptable security standard for implementation.

PII safeguards within the components connected by the APIs

The API concept is built around functional components: the sub-systems for Identity Management.

As well as the correct implementation or use of the appropriate API, a component should also meet PII requirements while this is present within the component. Such internal component design and PII behavior is the responsibility of the component supplier.

The customer architect responsible for an API connected solution should therefore ensure that the internal logic of an individual component is itself GDPR compliant. The API concept cannot itself provide any guarantee that components are designed with the same or sufficient internal levels of PII safeguards. What the API can do is to preserve this level of trust and prevent the creation of new vulnerabilities between these components.

The workflow connecting components in an OSIA enabled solution

OSIA provides a model for an open architecture. An end-to-end identity system may use some, or all of the OSIA components. It may use additional components to move data through the system. Wherever the system uses components to move data that are not covered by the OSIA framework definition then these should support end-to-end security with the same objective of GDPR compliance.

3.4.2 PII actors

The GDPR approach provides simple definitions.

- PII is a very wide category of information. It can be a name, a photo, a biometric, an email address, bank details, social media postings, medical data, and even an IP address;
- The PII data belongs to a Data Subject who is a natural person that might identified directly or indirectly using the PII;
- The usage, rules, and means of processing PII are determined by a Data Controller (e.g. the Government agency);
- The data is processed by a Data Processor.

When a government department acts as owner of an ID system then it is a Data Controller. It may also act as the Data Processor if it operates this system 'in house'.

However, in today's commercial world the Data Controller is equally likely to delegate some processing to a data center or to a business service for all or part of the system. In this case these delegated parties are Data Processors, and they also subject to the PII considerations.

Suppliers of the systems purchased and commissioned by Data Controllers, and operated by Data Processors are not directly subject to the regulation.

3.4.3 Data subject rights

A GDPR data subject has several rights that should be reflected throughout the wider ID systems architecture.

The right to be forgotten

A subject may ask for her data to be deleted.

Depending on the purpose and the authority of the system this right may be restricted or blocked, however the deletion of non-essential PII data may be a requirement according to some local laws. The Data Controller should be able to justify why specific items of PII need to be retained against the subject's wishes, and when there is no reason for retention then the automated purging of unnecessary data is generally recommended.

An example impact of this for API usage is where an enrolment client holds enrollee data until receiving a response via the API from the enrollment server to the effect that any client stored data can be deleted. The Data Processor operating the client is responsible to ensure this deletion is systematically applied. Typically this may be done with a configuration in the component product used.

Privacy by design

Systems should be designed to limit data collection, retention and accessibility.

This applies equally to APIs as to the system components themselves. No more data should be passed over an API than is required. A component passing or receiving data should consider how to minimize what new PII it collects, shares, and stores. The Data Controller should know by design what data is held and where; as well as which APIs are sharing what data.

An example of this principle for API usage can be where a credential management system receives PII over an API for credential production, then deletes the PII once the document is produced successfully. The system may limit its retained data to production audit data. A credential management system with a different set of responsibilities defined by the Data Controller may justify the retention of a wider set of PII, which might be replicated elsewhere in the system. A subject might ask to know where this data sits. The Controller should be able to tell the subject, and the Processor able to prove it.

Breach notifications

Supervisory powers vary globally. In the EU organizations have to notify their national supervisory authority in the event of a discovered data breach involving PII. They are given a 72 hour period to do this after becoming aware of the breach. The purpose of this notice period is to allow the organization to determine the nature and the impact of the data breach.

Data subjects have the right to be informed about data breaches involving their personal data.

By following the *Privacy by Design* approach, detection and data exposure can be assessed more accurately and quickly. Data is typically in transit between sub-systems, then at rest or in use within a given sub-system. When correctly implemented the OSIA API concept provides assurance against breaches at the API in-transit level. Combined with the knowledge of what data is stored, and where, this Privacy by Design approach assists in the detection of breaches.

At the time of GDPR's introduction the biggest issuing facing most organizations was not the implementation of new controls, but the discovery of where and what data was in their possession. The made it very difficult to know if data was ever compromised.

Risk and impact assessments

Looking at systems overall an organization has to perform a privacy impact assessment.

This describes what PII is collected, and how this is maintained, protected, and shared. This may be done as part of a wider ISO 27000 process including risk assessment, but this is not mandatory.

Today most providers of components within the OSIA framework will provide such a privacy impact assessment statement for their products, including the GDPR controls in that product.

Taken together with the OSIA API specification then these assessments can be compiled to an overall statement of system PII compliance.

Consent

Systems that deal with identity as their core subject matter may not be legally required to obtain consent for the capture and use of PII data. However, in this service-centric world more and more transactional and contextual data is captured, so this should not be assumed. If this data is to be collected then organizations have to obtain valid and explicit consent from the individuals.

The organizations must also be able to prove that they have gotten consent, not forgetting that in the EU individuals may withdraw their consent.

In the EU additional safeguards apply, where parental consent is required if personal data is to be collected about children under the age of 16.

An API usually indicates that the use or status of data is changing, so it should always be considered. Passing PII over an API requires that the consent covers the scope of this data sharing.

An example of this situation might be where an enrolment system captures biometric data to be loaded to a credential using an API. The Data Controller later decides that the same captured data will be passed via a new API to a biometric matching system. Both the Data Controller and Processor might find that they are processing this data contrary to the principle of consent. If consent matters in this case then the introduction of the new API may alert the user to a change of use. This is not to say that such changes only happen where APIs are concerned, but the OSIA API framework does represent different functions across Identity Management, and therefore indicates that consent may be a relevant consideration.

Data portability

The portability of requirement was conceived for both transparency and commercial reasons.

PII held should be usable by the Data Subject upon request. For privacy it may be held encrypted in the Data Processor system, but must be provided in a structured and commonly useable format to the Data Subject under reasonable terms of access.

An example scenario might be where a Data Subject wishes to have a copy of a child's birth record in a printed format or a format recognized by a third party. The concept of data portability may in some cases be implemented by a report service, or in some cases use an OSIA API to support the retrieval of personal attribute data to meet this demand.

3.4.4 What should OSIA API implementors do to prepare for safe PII?

1. Appoint someone as the organization's own GDPR or PII data expert. Someone who understands the Data Controller business requirements, and knows the technologies likely to be used for data processing.
2. GDPR is a good example of best practice in PII Management, but it is vital to understand the current local regulatory environment. Local existing laws and regulations take precedence unless subject to GDPR, and even then local laws may be stricter.
3. Use the OSIA API specification to understand the security organization of functional systems that might be needed and document an overall assessment of the PII privacy risk. Pay particular attention to sensitive data, and to the aggregation of PII.
4. Ensure that component suppliers understand and support the principles of good PII management, or GDPR. Most suppliers provide a description of how this is enforced in their products or systems. They may even provide a user manual and training for this function.
5. Document the design and lifecycle of data in the end-to-end system. The OSIA API Specification will help with this. It does not provide the full PII story, but it does provide the basis for the parts between components that the customer or its systems integrator will be responsible for.
6. Consider the Data Subject consent requirements, based on the functions that subject data will be subject to.
7. If the role is Data Controller, but not Data Processor then ensure that the organization used for Data Processing can understand and meet the guidelines for PII protection.
8. Remember that good planning and execution are essential, but it might be asked to prove correct operation. Systems logs and audit data should be available. This should include API usage to indicate where data has been transferred.

OSIA VERSIONS & REFERENCING

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

OSIA v. [version] - [interface name] v. [version number]

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

OSIA v. 2.0 - Notification v. 1.0.0

Below is the complete list of available interfaces with related versions:

Table 4.1: OSIA Services Versions

OSIA Release	1.0.0	2.0.0	3.0.0	4.1.0	5.0.0	6.0.0
OSIA Release Date	mar-2019	jun-2019	nov-2019	jul-2020	dec-2020	dec-2021
Notification	.	1.0.0	1.0.0	1.1.0	1.2.0	1.2.0
UIN Management	1.0.0	1.0.0	1.0.0	1.1.0	1.2.0	1.2.0
Data Access	1.0.0	1.0.0	1.0.0	1.1.0	1.3.0	1.3.0
Enrollment Services	.	.	.	1.0.0	1.1.0	1.2.0
Population Registry Services	.	.	1.0.0	1.2.0	1.3.0	1.4.0
Biometrics Services	.	1.0.0	1.1.0	1.3.0	1.4.0	1.5.0
Credential Services	.	.	.	1.0.0	1.1.0	1.2.0
Relying Party Services	1.0.0	1.1.0

INTERFACES

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:

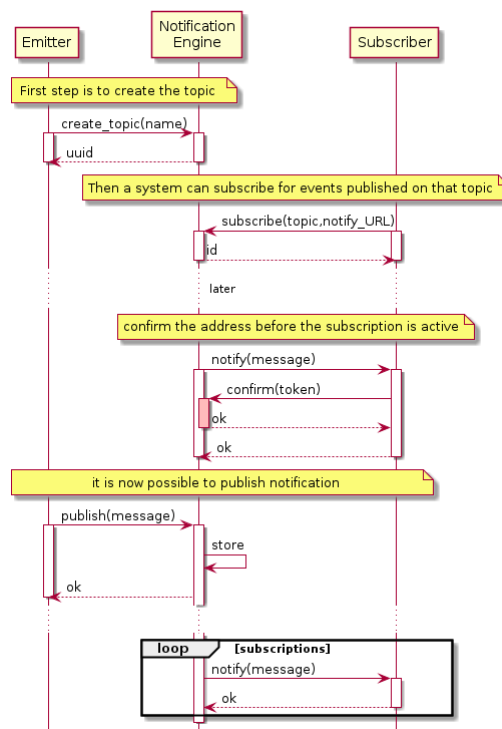


Fig. 5.1: Subscription & Notification Process

5.1.1 Services

For the Subscriber

subscribe(*topic, URL*)

Subscribe a URL to receive notifications sent to one topic

Authorization: notif.sub.write

Parameters

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

Returns a subscription ID

This service is synchronous.

listSubscriptions()

Get all subscriptions

Authorization: notif.sub.read

Parameters **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

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

confirm(*token*)

Used to confirm that the URL used during the subscription is valid

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

For the Publisher

createTopic(*topic*)

Create a new topic. This is required before an event can be sent to it.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

listTopics()

Get the list of all existing topics.

Authorization: notif.topic.read

Returns N/A

This service is synchronous.

deleteTopic(topic)

Delete a topic.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

publish(topic, subject, message)

Notify of a new event all systems that subscribed to this topic

Authorization: notif.topic.publish

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).

For the Receiver

notify(message)

Receive an event published by a publisher. This service needs to be registered through the subscription process.

Parameters **message** (*str*) – The message itself (a string buffer)

Returns N/A

5.1.2 Dictionaries

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

Table 5.1: Event Type

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

5.2 Data Access

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

5.2.1 Services

readPersonAttributes(*UIN*, *names*)

Read person attributes.

Authorization: `pr.person.read` or `cr.person.read`

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.

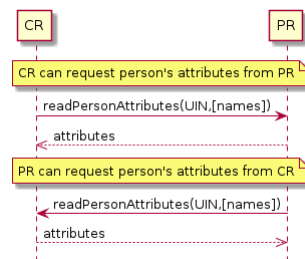


Fig. 5.2: `readPersonAttributes` Sequence Diagram

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: `pr.person.match` or `cr.person.match`

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.

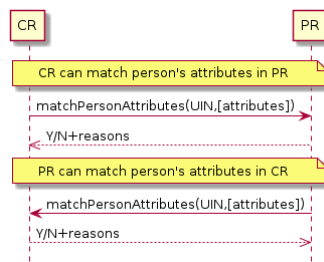


Fig. 5.3: matchPersonAttributes Sequence Diagram

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: `pr.person.verify` or `cr.person.verify`

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.

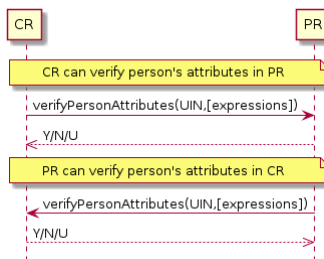


Fig. 5.4: verifyPersonAttributes Sequence Diagram

queryPersonUIN(*attributes, offset, limit*)

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: `pr.person.read` or `cr.person.read`

Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find UIN. Each attribute is described with its name and its value
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

Returns a list of matching UIN

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

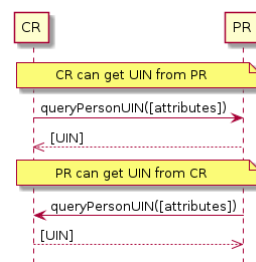


Fig. 5.5: queryPersonUIN Sequence Diagram

queryPersonList(*attributes, names, offset, limit*)

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: pr.person.read or cr.person.read

Parameters

- **attributes** (*list[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
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

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.

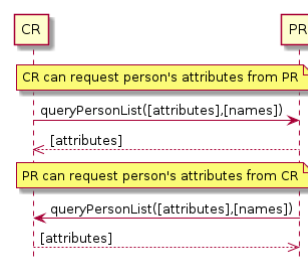


Fig. 5.6: queryPersonList Sequence Diagram

readDocument(*UINs, documentType, format*)

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

Authorization: pr.document.read or cr.document.read

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.

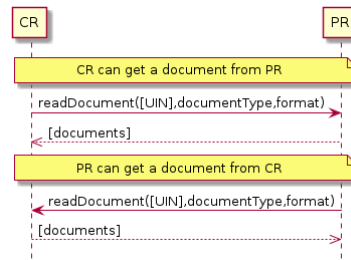


Fig. 5.7: readDocument Sequence Diagram

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*, *transactionID*)

Generate a new UIN.

Authorization: uin.generate

Parameters

- **attributes** (*list*[(*str*, *str*)]) – A list of pair (attribute name, value) that can be used to allocate a new UIN
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

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

This service is synchronous.

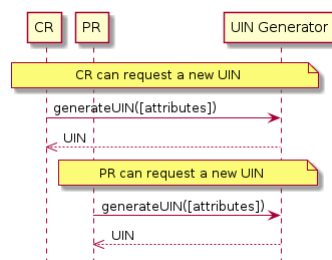


Fig. 5.8: generateUIN Sequence Diagram

5.4 Enrollment Services

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

- When enrollment is done in one step, the CreateEnrollment can contain all the data and an additional flag (finalize) to indicate all data was collected.
- During the process, enrollment structure can be updated. Only the data that changed need to be transferred. Data not included is left unchanged on the server. In the following example, the biographic data is not changed.
- 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 documents

Adding one document or deleting one document implies that:

- The full document list is read (ReadEnrollment)
 - The document list is altered locally to the enrollment client (add or delete)
 - The full document list is sent back using the UpdateEnrollment service
-

5.4.1 Services

createEnrollment(*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Insert a new enrollment.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.
- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

readEnrollment(*enrollmentID, attributes, transactionID*)

Retrieve the attributes of an enrollment.

Authorization: enroll.read

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.

- **attributes** (*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 enrollment data.

updateEnrollment (*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Update an enrollment.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.
- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data, this can be partial data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data, this can be partial data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

partialupdateEnrollment (*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Update part of an enrollment. Not all attributes are mandatory. The payload is defined as per [RFC 7396](#).

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.
- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data, this can be partial data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data, this can be partial data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

finalizeEnrollment (*enrollmentID, transactionID*)

When all the enrollment steps are done, the enrollment client indicates to the enrollment server that all data has been collected and that any further processing can be triggered.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

deleteEnrollment (*enrollmentID, transactionID*)

Deletes the enrollment

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

findEnrollments (*expressions, offset, limit, transactionID*)

Retrieve a list of enrollments which match passed in search criteria.

Authorization: enroll.read

Parameters

- **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
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)
- **transactionID** (*string*) – The client generated transactionID.

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

createBuffer (*enrollmentId, data, digest*)

This service is used to send separately the buffers of the images. Buffers can be sent any time from the enrollment client prior to the create or update.

Authorization: enroll.buf.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **data** (*image*) – The buffer data.
- **transactionID** (*string*) – The client generated transactionID.
- **digest** (*string*) – The digest (hash) of the buffer used by the server to check the integrity of the data received.

Returns a status indicating success or error and in case of success the buffer ID.

readBuffer (*enrollmentId, bufferId*)

This service is used to get the data of a buffer.

Authorization: enroll.buf.read

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **bufferID** (*str*) – The ID of the buffer.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error and in case of success the data of the buffer and a digest.

5.4.2 Attributes

The “attributes” 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 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 can be used for tracing and debugging.

5.4.4 Data Model

Table 5.7: Enrolment Data Model

Type	Description	Example
Enrollment	Set of person data which are captured.	TBD
Document Data	The document data of the enrollment.	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, palm-print, portrait, iris or signature. A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)	fingerprint, portrait, iris, signature
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	TBD
Enrollment Flags	a dictionary (list of names and values) for custom flags.	TBD
Request Data	a dictionary (list of names and values) for data related to the enrollment itself (the operator, the station, the data, etc.).	TBD
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	operatorName, enrollmentDate, etc.
Attributes	a dictionary (list of names and values or <i>range</i> of values) describing the attributes to return. Attributes can apply on biographic data, document data, request data, or enrollment flag data.	TBD
Expressions	Each expression is described with the attribute’s name, the operator (one of <, >, =, >=, <=, !=) and the attribute value	TBD

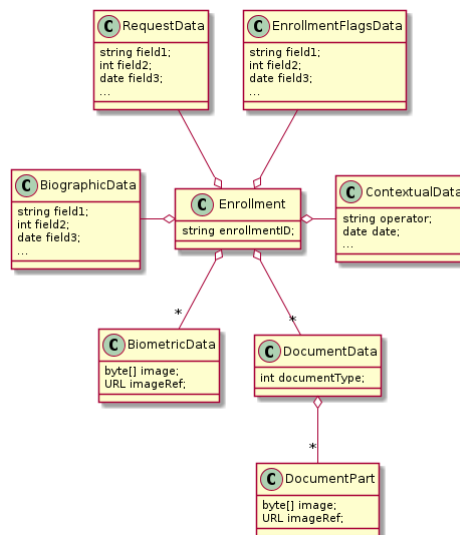


Fig. 5.9: Enrollment Data Model

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

findPersons(*expressions, group, reference, gallery, offset, limit, transactionID*)

Retrieve a list of persons which match passed in search criteria.

Authorization: `pr.person.read`

Parameters

- **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
- **group** (*bool*) – Group the results per person and return only personID
- **reference** (*bool*) – Limit the query to the reference identities
- **gallery** (*string*) – A gallery ID used to limit the search
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)

- **limit** (*int*) – The maximum number of items to return (optional, default to 100)
- **transactionID** (*string*) – The client generated transactionID.

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

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

Create a new person.

Authorization: pr.person.write

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: pr.person.read

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: pr.person.write

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: pr.person.write

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.

mergePerson(*personID1*, *personID2*, *transactionID*)

Merge two person records into a single one. Identity ID are preserved and in case of duplicates an error is returned and no changes are done. The reference identity is not changed.

Authorization: pr.person.write

Parameters

- **personID1** (*str*) – The ID of the person that will receive new identities

- **personID2** (*str*) – The ID of the person that will give its identities. It will be deleted if the move of all identities is successful.
- **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: pr.identity.write

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: pr.identity.read

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: pr.identity.write

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: pr.identity.write

Parameters

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

- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.reference.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.reference.read`

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: `pr.gallery.read`

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, offset, limit*)

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

Authorization: `pr.gallery.read`

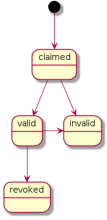
Parameters

- **galleryID** (*str*) – Gallery whose content will be returned.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 1000)

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"> • a status, such as active or inactive, defining the status of the record (the record can be excluded from queries based on this status), • a physical status, such as alive or dead, defining the status of the person, • a set of identities, keeping track of all identity data submitted by the person during the life of the system, • 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. 	N/A
Identity	<p>The attributes describing an identity of a person. An identity has a status such as: claimed (identity not yet validated), valid (the identity is valid), invalid (the identity is confirmed as not valid), revoked (the identity cannot be used any longer).</p> <p>An identity can be updated only in the status claimed. The proposed transitions for the status are represented below. It can be adapted if needed.</p>  <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	firstName, lastName, dateOfBirth, etc.
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	operatorName, enrollmentDate, etc.
Biometric Data	<p>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, palm-print, portrait, iris or signature.</p> <p>A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)</p>	fingerprint, portrait, iris, signature
Document	The document data (images) attached to the identity and used to validate it.	Birth certificate, invoice

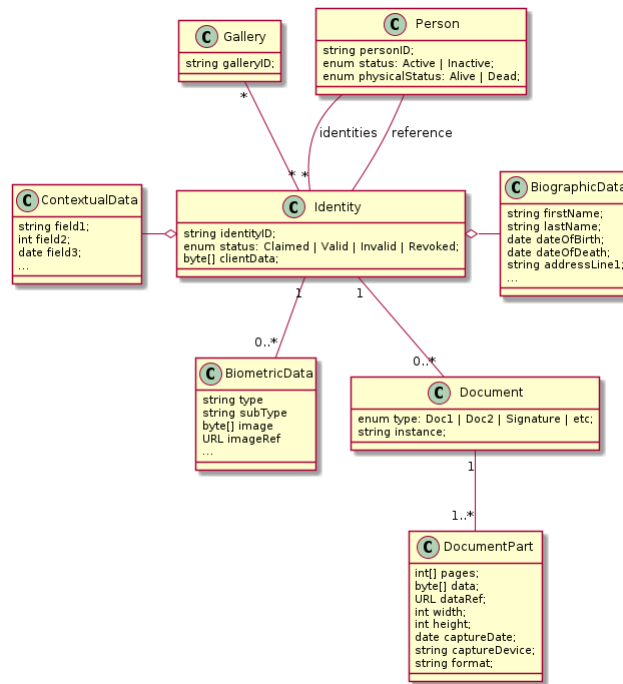


Fig. 5.10: Population Registry Data Model

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, this indicates that the client wants an asynchronous answer, even if the result is immediately available.

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

5.6.1 Services

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

Create a new encounter. No identify is performed.

Authorization: abis.encounter.write

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.

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

Read the data of an encounter.

Authorization: abis.encounter.read

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.

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

Update an encounter.

Authorization: abis.encounter.write

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.

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

Delete an encounter.

Authorization: abis.encounter.write

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.

mergeEncounter(*personID1, personID2, callback, transactionID, options*)

Merge two sets of encounters into a single set. Merging a set of N encounters with a set of M encounters will result in a single set of $N+M$ encounters. Encounter ID are preserved and in case of duplicates an error is returned and no changes are done.

Authorization: abis.encounter.write

Parameters

- **personID1** (*str*) – The ID of the person that will receive new encounters
- **personID2** (*str*) – The ID of the person that will give its encounters
- **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 result will be sent later.

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

Read the generated template.

Authorization: abis.encounter.read

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)
- **templateFormat** (*str*) – the format of the template to return (optional)
- **qualityFormat** (*str*) – the format of the quality to return (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.

updateEncounterStatus(*personID, encounterID, status, transactionID*)

Set an encounter status.

Authorization: abis.encounter.write

Parameters

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

Returns a status indicating success or error.

updateEncounterGalleries(*personID, encounterID, galleries, transactionID*)

Update the galleries of an encounter. This service is used to move one encounter from one gallery to another one without updating the full encounter, which may be resource consuming in a biometric system.

Authorization: abis.encounter.write

Parameters

- **personID** (*str*) – The ID of the person.
- **encounterID** (*str*) – The encounter ID.
- **galleries** (*list[str]*) – The new list of galleries for this encounter.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

Returns a status indicating success or error.

readGalleries(*callback, transactionID, options*)

Read the ID of all the galleries.

Authorization: abis.gallery.read

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, offset, limit, options*)

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

Authorization: abis.gallery.read

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.
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 1000)
- **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: abis.identify

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: abis.verify

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.

identify(*galleryID, filter, personID, encounterID, callback, transactionID, options*)

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

Authorization: abis.verify

Parameters

- **galleryID** (*str*) – Search only in this gallery.
- **filter** (*dict*) – The input data (filters and biometric data)
- **personID** – the person ID
- **encounterID** – the encounter 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: abis.verify

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: abis.verify

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
priority	Priority of the request. Values range from 0 to 9. 0 indicates the lowest priority, 9 indicates the highest priority.
maxNbCand	The maximum number of candidates to return.
threshold	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
algorithm	Specify the type of algorithm to be used.
accuracyLevel	Specify the accuracy expected of the request. This is to support different use cases, when different behavior of the ABIS is expected (response time, accuracy, consolidation/fusion, etc.).

5.6.3 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). An encounter has a status indicating if this encounter is used in the biometric searches. Allowed values are active or inactive .	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, palm-print, portrait, iris or signature. A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)	fingerprint, portrait, iris, signature
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. It can also be extended with proprietary information to better describe the matching result (for instance: rotation needed to align the probe and the candidate)	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

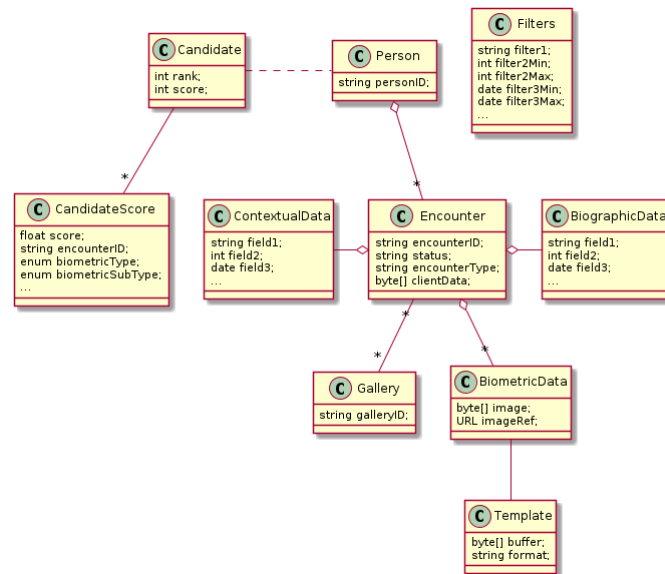


Fig. 5.11: Biometric Data Model

5.7 Credential Services

This interface describes services to manage credentials and credential requests in the context of an identity system.

5.7.1 Services

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

Request issuance of a secure credential.

Authorization: cms.request.write

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, a credential request identifier.

readCredentialRequest(*credentialRequestID*, *attributes*, *transactionID*)

Retrieve the data/status of a credential request.

Authorization: cms.request.read

Parameters

- **credentialRequestID** (*str*) – The ID of the credential request.
- **attributes** (*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.

updateCredentialRequest(*credentialRequestID*, *additionalData*, *transactionID*)

Update the requested issuance of a secure credential.

Authorization: `cms.request.write`

Parameters

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

cancelCredentialRequest (*credentialRequestID, transactionID*)

Cancel the requested issuance of a secure credential.

Authorization: `cms.request.write`

Parameters

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

Returns a status indicating success or error.

findCredentials (*expressions, transactionID*)

Retrieve a list of credentials that match the passed in search criteria.

Authorization: `cms.credential.read`

Parameters

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

Returns a status indicating success or error, in the case of success the list of matching credentials.

readCredential (*credentialID, attributes, 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: `cms.credential.read`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **attributes** (*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, additionalData, transactionID*)

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

Authorization: `cms.credential.write`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for suspension.
- **transactionID** (*string*) – The (optional) client generated transactionID.

Returns a status indicating success or error.

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

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

Authorization: cms.credential.write

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for unsuspension.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

revokeCredential(*credentialID*, *additionalData*, *transactionID*)

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

Authorization: cms.credential.write

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for revocation.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

setCredentialStatus(*credentialID*, *status*, *reason*, *requester*, *comment*, *transactionID*)

Change the status of a credential. This is an extension of the revoke/suspend services, supporting more statuses and transitions.

Authorization: cms.credential.write

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **status** (*string*) – The new status of the credential
- **reason** (*string*) – A text describing the cause of the change of status
- **requester** (*string*) – The client generated transactionID.
- **comment** (*string*) – A free text comment
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

findCredentialProfiles(*expressions*, *transactionID*)

Retrieve a list of credential profiles that match the passed in search criteria

Authorization: cms.profile.read

Parameters

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

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

5.7.2 Attributes

The “attributes” 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. @@ -128,7 +128,7 @@ attributes to retrieve.

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

Some calls may require new 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.

5.7.3 Data Model

Table 5.11: Credential Data Model

Type	Description	Example
Credential	<p>The attributes of the credential itself</p> <p>The proposed transitions for the status are represented below. It can be adapted if needed.</p> <pre> graph TD start(()) --> new(new) new -- issue --> active(active) active -- suspend --> suspended(suspended) active -- suspend --> revoked(revoked) suspended -- unsuspend --> active suspended -- suspend --> revoked </pre>	ID, status, dates, serial number
Biometric Data	<p>Digital representation of biometric characteristics.</p> <p>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, palm-print, portrait, iris or signature.</p> <p>A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)</p>	fingerprint, portrait, iris, signature
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	first name, last name, date of birth, etc.
Request Data	a dictionary (list of names and values) for data related to the request itself.	Type of credential, action to execute, priority

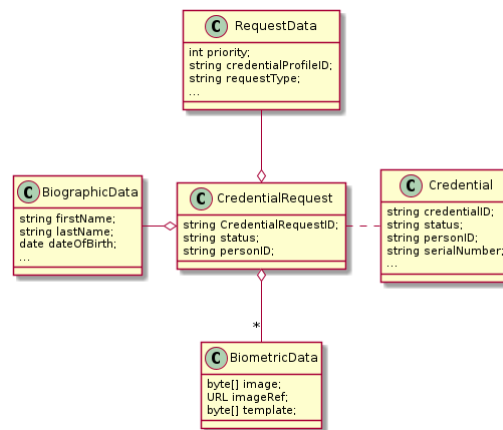


Fig. 5.12: Credential Data Model

5.8 ID Usage

ID Usage consists of a set of services implemented on top of identity systems to favour third parties consumption of identity data. The services can be classified in three sets:

- Relying Party API: submitting citizen ID attributes for validation

The purpose of the Relying Party (RP) API is to extend the use of government-issued identity to registered third party services. The individual will submit their ID attributes to the relying party in order to enroll for, or access, a particular service. The relying party will leverage the RP API to access the identity management system and verify the individual's identity. In this way, external relying parties can quickly and easily verify individuals based on their government issued ID attributes.

Use case applications: telco enrolment

The RP API enables a telco operator to check an individual's identity when applying for a service contract. The telco relies on the government to confirm that the attributes submitted by the individual match against the data held in the database therefore being able to confidently identify the new subscriber. This scenario can be replicated across multiple sectors including banking and finance.

- Digital Credential Management API: delegating digital issuance to third parties

The purpose of the Digital Credential Management (DCM) API is to enable external wallet providers to manage government issued digital credentials distribution, storage and usage.

Use case applications: digital driver license

The DCM API enables individuals to request a digital driver license as a digital credential in their selected wallet to use for online and offline identification. The user initiates a request for digital driver license using the Digital Credential Distribution System (DCDS) frontend, which sends the request to the identity management system. The Credential Management System (CMS) then issues the digital credential by dedicated API endpoint of the DCDS.

- Federation API: user-initiated attributes sharing

The purpose of the federation API is to enable the user to share their attributes with a chosen relying party using well-known internet protocol: OpenID Connect. The relying party benefits from the government's verified attributes.

Use case applications: on-line registration to gambling website

The Federation API enables individuals to log-in with their government credential (log-in/password) and share verified attributes ex. age (above 18) with the relying party.

5.8.1 Relying Party API

verifyIdentity(*Identifier*, *attributeSet*)

Verify an Identity based on an identifier (UIN, token...) and a set of Identity Attributes. Verification is strictly matching all provided identity attributes to compute the global Boolean matching result.

Authorization: *id.verify*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **attributeSet** (*list[str]*) – A set of identity attributes associated to the identifier and to be verified by the system

Returns Y or N

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

identify(*attributeSet*, *outputAttributeSet*)

Identify possibly matching identities against an input set of attributes. Returns an array of predefined datasets as described by *outputAttributeSet*.

Note: This service may be limited to some specific government RPs

Authorization: *id.identify*

Parameters

- **attributeSet** (*list[str]*) – A list of pair (name,value) requested
- **outputAttributeSet** (*list[str]*) – An array of attributes requested

Returns Y or N

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

readAttributes(*Identifier*, *outputAttributeSet*)

Get a list of identity attributes attached to a given identifier.

Authorization: *id.read*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **outputAttributeSet** (*list[str]*) – defining the identity attributes to be provided back to the caller

Returns An array of the requested attributes

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

readAttributeSet(*Identifier*, *AttributeSetName*)

Get a set of identity attributes as defined by *attributeSet*, attached to a given identifier.

Authorization: *id.set.read*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **attributeSetName** (*str*) – The name of predefined attributes set name

Returns An array of the requested attributes

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

5.8.2 Attribute set

When identity attributes are exchanged, they are included in an attribute set, possibly containing groups like biographic data, biometric data, document data, contact data. . . This structure is extensible and may be complemented with other data groups, and each group may contain any number of attribute name / attribute value pairs.

5.8.3 Attribute set name

Attribute sets are by definition structures with variable and optional content, hence it may be useful to pre-agree on a given attribute set content and name between two or more systems in a given project scope.

Any string may be used to define an attribute set name, but in the scope of this specification following names are reserved and predefined:

Name	Description	Data Included
"DEFAULT_SET_01"	Minimum demographic data	First name Last name DoB Place of birth
"DEFAULT_SET_02"	Minimum demographic and portrait	Minimum demographic data + portrait
"DEFAULT_SET_EIDAS"	Set expected to comply with eIDAS pivotal attributes.	TBD

5.8.4 Output Attribute set

To specify what identity attributes are expected in return when performing e.g. an identify request or a read attributes.

COMPONENTS

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

6.1 Enrollment Component

The enrollment component MAY implement the following interfaces:

6.1.1 Enrollment Services

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

- When enrollment is done in one step, the CreateEnrollment can contain all the data and an additional flag (finalize) to indicate all data was collected.
- During the process, enrollment structure can be updated. Only the data that changed need to be transferred. Data not included is left unchanged on the server. In the following example, the biographic data is not changed.
- 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 documents

Adding one document or deleting one document implies that:

- The full document list is read (ReadEnrollment)
 - The document list is altered locally to the enrollment client (add or delete)
 - The full document list is sent back using the UpdateEnrollment service
-

Services

createEnrollment(*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Insert a new enrollment.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.

- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

readEnrollment(*enrollmentID, attributes, transactionID*)

Retrieve the attributes of an enrollment.

Authorization: enroll.read

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **attributes** (*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 enrollment data.

updateEnrollment(*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Update an enrollment.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.
- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data, this can be partial data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data, this can be partial data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

partialupdateEnrollment(*enrollmentID, enrollmentTypeId, enrollmentFlags, requestData, contextualData, biometricData, biographicData, documentData, finalize, transactionID*)

Update part of an enrollment. Not all attributes are mandatory. The payload is defined as per [RFC 7396](#).

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment. If the enrollment already exists for the ID an error is returned.
- **enrollmentTypeId** (*str*) – The enrollment type ID of the enrollment.
- **enrollmentFlags** (*dict*) – The enrollment custom flags.
- **requestData** (*dict*) – The enrollment data related to the enrollment itself.
- **contextualData** (*dict*) – Information about the context of the enrollment
- **biometricData** (*list*) – The enrollment biometric data, this can be partial data.
- **biographicData** (*dict*) – The enrollment biographic data.
- **documentData** (*list*) – The enrollment biometric data, this can be partial data.
- **finalize** (*str*) – Flag to indicate that data was collected.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

finalizeEnrollment (*enrollmentID, transactionID*)

When all the enrollment steps are done, the enrollment client indicates to the enrollment server that all data has been collected and that any further processing can be triggered.

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

deleteEnrollment (*enrollmentID, transactionID*)

Deletes the enrollment

Authorization: enroll.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

findEnrollments (*expressions, offset, limit, transactionID*)

Retrieve a list of enrollments which match passed in search criteria.

Authorization: enroll.read

Parameters

- **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
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)
- **transactionID** (*string*) – The client generated transactionID.

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

createBuffer (*enrollmentId, data, digest*)

This service is used to send separately the buffers of the images. Buffers can be sent any time from the enrollment client prior to the create or update.

Authorization: enroll.buf.write

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **data** (*image*) – The buffer data.
- **transactionID** (*string*) – The client generated transactionID.
- **digest** (*string*) – The digest (hash) of the buffer used by the server to check the integrity of the data received.

Returns a status indicating success or error and in case of success the buffer ID.

readBuffer(*enrollmentId, bufferId*)

This service is used to get the data of a buffer.

Authorization: enroll.buf.read

Parameters

- **enrollmentID** (*str*) – The ID of the enrollment.
- **bufferID** (*str*) – The ID of the buffer.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error and in case of success the data of the buffer and a digest.

Attributes

The “attributes” 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 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 can be used for tracing and debugging.

Data Model

Table 6.1: Enrolment Data Model

Type	Description	Example
Enrollment	Set of person data which are captured.	TBD
Document Data	The document data of the enrollment.	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, palm-print, portrait, iris or signature. A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)	fingerprint, portrait, iris, signature
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	TBD
Enrollment Flags	a dictionary (list of names and values) for custom flags.	TBD
Request Data	a dictionary (list of names and values) for data related to the enrollment itself (the operator, the station, the data, etc.).	TBD
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	operatorName, enrollmentDate, etc.
Attributes	a dictionary (list of names and values or <i>range</i> of values) describing the attributes to return. Attributes can apply on biographic data, document data, request data, or enrollment flag data.	TBD
Expressions	Each expression is described with the attribute's name, the operator (one of <, >, =, >=, <=, !=) and the attribute value	TBD

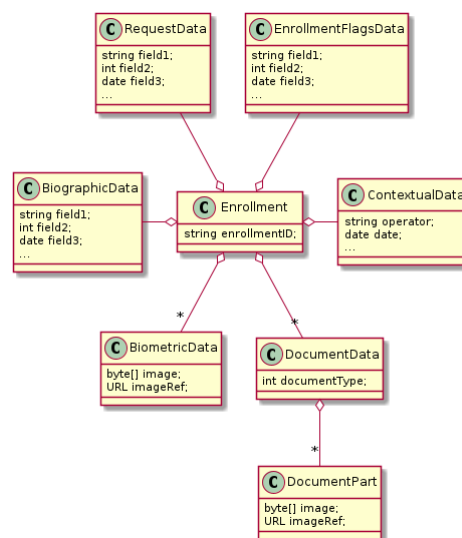


Fig. 6.1: Enrollment Data Model

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:

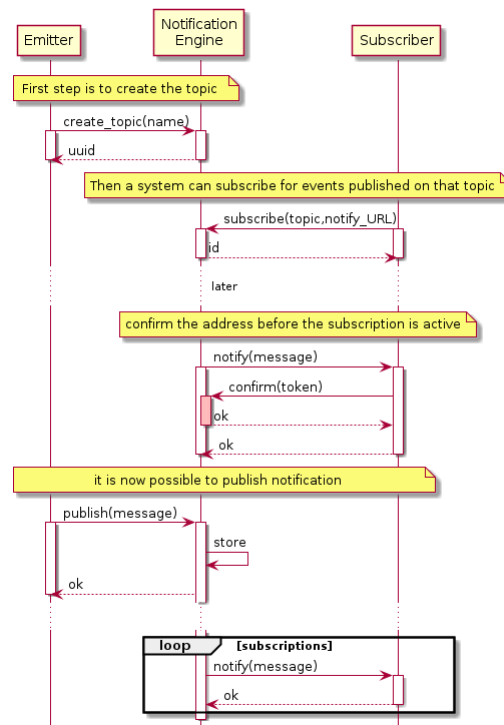


Fig. 6.2: Subscription & Notification Process

Services

For the Subscriber

subscribe(*topic*, *URL*)

Subscribe a URL to receive notifications sent to one topic

Authorization: notif.sub.write

Parameters

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

Returns a subscription ID

This service is synchronous.

listSubscriptions()

Get all subscriptions

Authorization: notif.sub.read

Parameters **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

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

confirm(*token*)

Used to confirm that the URL used during the subscription is valid

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

For the Publisher

createTopic(*topic*)

Create a new topic. This is required before an event can be sent to it.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

listTopics()

Get the list of all existing topics.

Authorization: notif.topic.read

Returns N/A

This service is synchronous.

deleteTopic(*topic*)

Delete a topic.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

publish(*topic, subject, message*)

Notify of a new event all systems that subscribed to this topic

Authorization: notif.topic.publish

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).

For the Receiver

notify(*message*)

Receive an event published by a publisher. This service needs to be registered through the subscription process.

Parameters **message** (*str*) – The message itself (a string buffer)

Returns N/A

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: `pr.person.read` or `cr.person.read`

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.

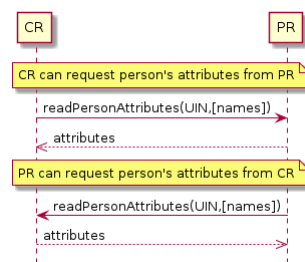


Fig. 6.3: readPersonAttributes Sequence Diagram

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: pr.person.match or cr.person.match

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.

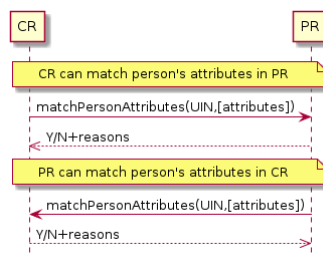


Fig. 6.4: matchPersonAttributes Sequence Diagram

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: pr.person.verify or cr.person.verify

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.

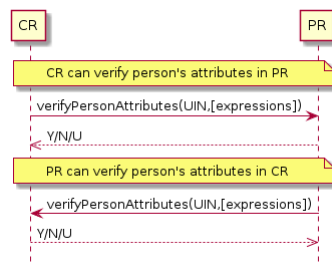


Fig. 6.5: verifyPersonAttributes Sequence Diagram

queryPersonUIN(attributes, offset, limit)

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: pr.person.read or cr.person.read

Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find UIN. Each attribute is described with its name and its value
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

Returns a list of matching UIN

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

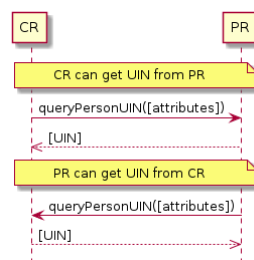


Fig. 6.6: queryPersonUIN Sequence Diagram

queryPersonList(attributes, names, offset, limit)

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: pr.person.read or cr.person.read

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

- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

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.

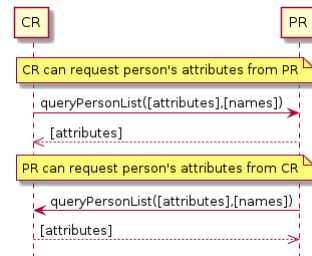


Fig. 6.7: queryPersonList Sequence Diagram

readDocument (*UINs, documentType, format*)

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

Authorization: pr.document.read or cr.document.read

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.

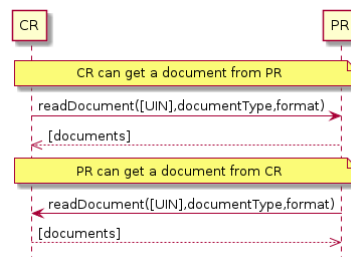


Fig. 6.8: readDocument Sequence Diagram

Dictionaries

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

findPersons(*expressions, group, reference, gallery, offset, limit, transactionID*)

Retrieve a list of persons which match passed in search criteria.

Authorization: `pr.person.read`

Parameters

- **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
- **group** (*bool*) – Group the results per person and return only personID
- **reference** (*bool*) – Limit the query to the reference identities
- **gallery** (*string*) – A gallery ID used to limit the search
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)
- **transactionID** (*string*) – The client generated transactionID.

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

createPerson(*personID, personData, transactionID*)

Create a new person.

Authorization: `pr.person.write`

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: `pr.person.read`

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: pr.person.write

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: pr.person.write

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.

mergePerson(*personID1*, *personID2*, *transactionID*)

Merge two person records into a single one. Identity ID are preserved and in case of duplicates an error is returned and no changes are done. The reference identity is not changed.

Authorization: pr.person.write

Parameters

- **personID1** (*str*) – The ID of the person that will receive new identities
- **personID2** (*str*) – The ID of the person that will give its identities. It will be deleted if the move of all identities is successful.
- **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: pr.identity.write

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: `pr.identity.read`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.identity.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.reference.write`

Parameters

- **personID** (*str*) – The ID of the person.
- **identityID** (*str*) – 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: `pr.reference.read`

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: `pr.gallery.read`

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, offset, limit*)

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

Authorization: `pr.gallery.read`

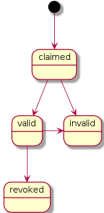
Parameters

- **galleryID** (*str*) – Gallery whose content will be returned.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 1000)

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"> • a status, such as active or inactive, defining the status of the record (the record can be excluded from queries based on this status), • a physical status, such as alive or dead, defining the status of the person, • a set of identities, keeping track of all identity data submitted by the person during the life of the system, • 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. 	N/A
Identity	<p>The attributes describing an identity of a person. An identity has a status such as: claimed (identity not yet validated), valid (the identity is valid), invalid (the identity is confirmed as not valid), revoked (the identity cannot be used any longer).</p> <p>An identity can be updated only in the status claimed. The proposed transitions for the status are represented below. It can be adapted if needed.</p>  <pre> graph TD Start(()) --> claimed[claimed] claimed --> valid[valid] claimed --> invalid[invalid] valid --> revoked[revoked] invalid --> invalid </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	firstName, lastName, dateOfBirth, etc.
Contextual Data	A dictionary (list of names and values) attached to the context of establishing the identity	operatorName, enrollmentDate, etc.
Biometric Data	<p>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, palm-print, portrait, iris or signature.</p> <p>A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)</p>	fingerprint, portrait, iris, signature
Document	The document data (images) attached to the identity and used to validate it.	Birth certificate, invoice

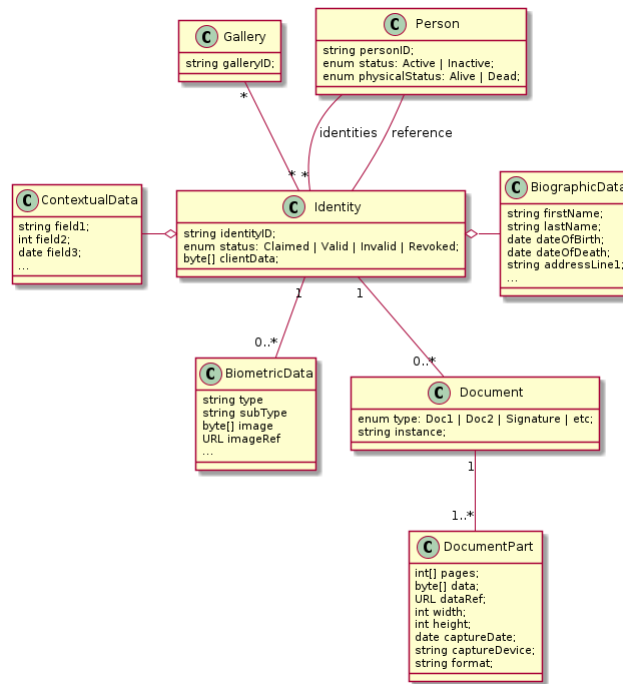


Fig. 6.9: Population Registry Data Model

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:

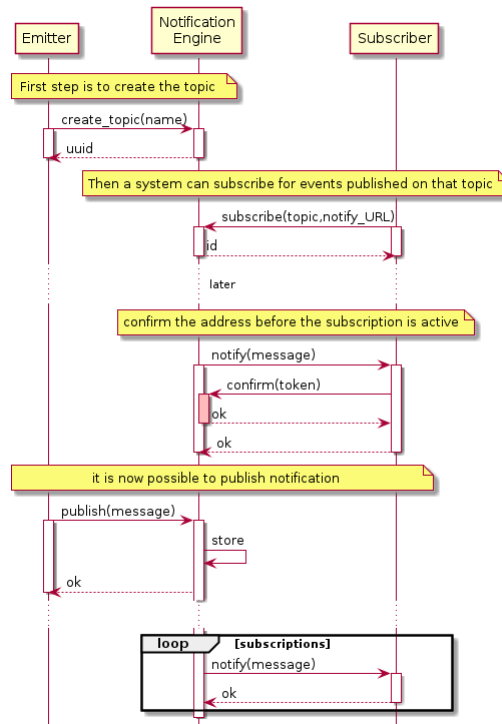


Fig. 6.10: Subscription & Notification Process

Services

For the Subscriber

subscribe(*topic*, *URL*)

Subscribe a URL to receive notifications sent to one topic

Authorization: notif.sub.write

Parameters

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

Returns a subscription ID

This service is synchronous.

listSubscriptions()

Get all subscriptions

Authorization: notif.sub.read

Parameters **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

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

confirm(*token*)

Used to confirm that the URL used during the subscription is valid

Authorization: notif.sub.write

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

Returns bool

This service is synchronous.

For the Publisher

createTopic(*topic*)

Create a new topic. This is required before an event can be sent to it.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

listTopics()

Get the list of all existing topics.

Authorization: notif.topic.read

Returns N/A

This service is synchronous.

deleteTopic(*topic*)

Delete a topic.

Authorization: notif.topic.write

Parameters **topic** (*str*) – Topic

Returns N/A

This service is synchronous.

publish(*topic, subject, message*)

Notify of a new event all systems that subscribed to this topic

Authorization: notif.topic.publish

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).

For the Receiver

notify(*message*)

Receive an event published by a publisher. This service needs to be registered through the subscription process.

Parameters **message** (*str*) – The message itself (a string buffer)

Returns N/A

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: `pr.person.read` or `cr.person.read`

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.

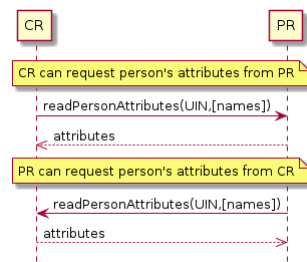


Fig. 6.11: readPersonAttributes Sequence Diagram

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: pr.person.match or cr.person.match

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.

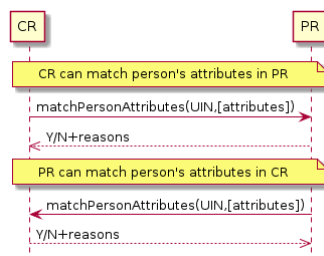


Fig. 6.12: matchPersonAttributes Sequence Diagram

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: pr.person.verify or cr.person.verify

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.

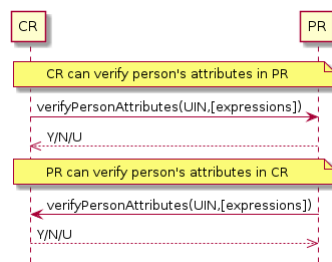


Fig. 6.13: `verifyPersonAttributes` Sequence Diagram

`queryPersonUIN(attributes, offset, limit)`

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: `pr.person.read` or `cr.person.read`

Parameters

- **attributes** (*list[(str, str)]*) – The attributes to be used to find UIN. Each attribute is described with its name and its value
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

Returns a list of matching UIN

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

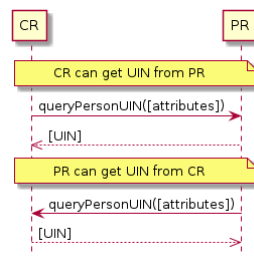


Fig. 6.14: `queryPersonUIN` Sequence Diagram

`queryPersonList(attributes, names, offset, limit)`

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: `pr.person.read` or `cr.person.read`

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
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)

- **limit** (*int*) – The maximum number of items to return (optional, default to 100)

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.

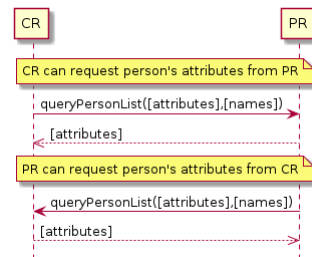


Fig. 6.15: queryPersonList Sequence Diagram

readDocument (*UINs, documentType, format*)

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

Authorization: `pr.document.read` or `cr.document.read`

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.

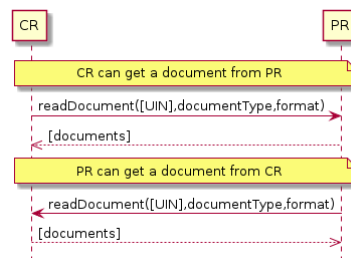


Fig. 6.16: readDocument Sequence Diagram

Dictionaries

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*, *transactionID*)

Generate a new UIN.

Authorization: uin.generate

Parameters

- **attributes** (*list*[(*str*, *str*)]) – A list of pair (attribute name, value) that can be used to allocate a new UIN
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

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

This service is synchronous.

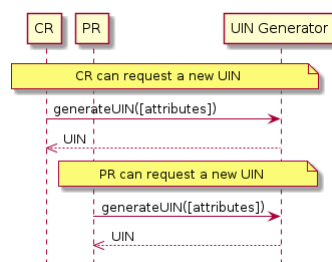


Fig. 6.17: generateUIN Sequence Diagram

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, this indicates that the client wants an asynchronous answer, even if the result is immediately available.

See [Biometrics](#) for the technical details of this interface.

Services

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

Create a new encounter. No identify is performed.

Authorization: abis.encounter.write

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.

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

Read the data of an encounter.

Authorization: abis.encounter.read

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.

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

Update an encounter.

Authorization: abis.encounter.write

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.

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

Delete an encounter.

Authorization: abis.encounter.write

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.

mergeEncounter(*personID1, personID2, callback, transactionID, options*)

Merge two sets of encounters into a single set. Merging a set of N encounters with a set of M encounters will result in a single set of $N+M$ encounters. Encounter ID are preserved and in case of duplicates an error is returned and no changes are done.

Authorization: abis.encounter.write

Parameters

- **personID1** (*str*) – The ID of the person that will receive new encounters
- **personID2** (*str*) – The ID of the person that will give its encounters
- **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 result will be sent later.

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

Read the generated template.

Authorization: abis.encounter.read

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)
- **templateFormat** (*str*) – the format of the template to return (optional)
- **qualityFormat** (*str*) – the format of the quality to return (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.

updateEncounterStatus(*personID, encounterID, status, transactionID*)

Set an encounter status.

Authorization: abis.encounter.write

Parameters

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

Returns a status indicating success or error.

updateEncounterGalleries(*personID, encounterID, galleries, transactionID*)

Update the galleries of an encounter. This service is used to move one encounter from one gallery to another one without updating the full encounter, which maybe resource consuming in a biometric system.

Authorization: abis.encounter.write

Parameters

- **personID** (*str*) – The ID of the person.
- **encounterID** (*str*) – The encounter ID.
- **galleries** (*list[str]*) – The new list of galleries for this encounter.
- **transactionID** (*str*) – A free text used to track the system activities related to the same transaction.

Returns a status indicating success or error.

readGalleries(*callback, transactionID, options*)

Read the ID of all the galleries.

Authorization: abis.gallery.read

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, offset, limit, options*)

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

Authorization: abis.gallery.read

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.
- **offset** (*int*) – The offset of the query (first item of the response) (optional, default to 0)
- **limit** (*int*) – The maximum number of items to return (optional, default to 1000)
- **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: abis.identify

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: abis.verify

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.

identify(*galleryID, filter, personID, encounterID, callback, transactionID, options*)

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

Authorization: abis.verify

Parameters

- **galleryID** (*str*) – Search only in this gallery.
- **filter** (*dict*) – The input data (filters and biometric data)
- **personID** – the person ID
- **encounterID** – the encounter 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: abis.verify

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: abis.verify

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
priority	Priority of the request. Values range from 0 to 9. 0 indicates the lowest priority, 9 indicates the highest priority.
maxNbCand	The maximum number of candidates to return.
threshold	The threshold to apply on the score to filter the candidates during an identification, authentication or verification.
algorithm	Specify the type of algorithm to be used.
accuracyLevel	Specify the accuracy expected of the request. This is to support different use cases, when different behavior of the ABIS is expected (response time, accuracy, consolidation/fusion, etc.).

Data Model

Table 6.16: 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). An encounter has a status indicating if this encounter is used in the biometric searches. Allowed values are active or inactive .	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, palm-print, portrait, iris or signature. A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)	fingerprint, portrait, iris, signature
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. It can also be extended with proprietary information to better describe the matching result (for instance: rotation needed to align the probe and the candidate)	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

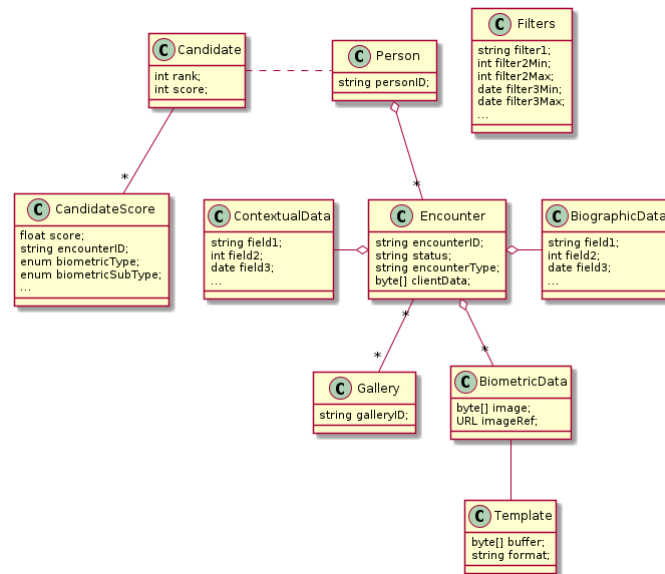


Fig. 6.18: Biometric Data Model

6.6 Credential Management System

The credential management system component MAY implement the following interfaces:

6.6.1 Credential Services

This interface describes services to manage credentials and credential requests in the context of an identity system.

Services

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

Request issuance of a secure credential.

Authorization: cms.request.write

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, a credential request identifier.

readCredentialRequest(*credentialRequestID*, *attributes*, *transactionID*)

Retrieve the data/status of a credential request.

Authorization: cms.request.read

Parameters

- **credentialRequestID** (*str*) – The ID of the credential request.
- **attributes** (*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.

updateCredentialRequest(*credentialRequestID*, *additionalData*, *transactionID*)

Update the requested issuance of a secure credential.

Authorization: `cms.request.write`

Parameters

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

cancelCredentialRequest(*credentialRequestID*, *transactionID*)

Cancel the requested issuance of a secure credential.

Authorization: `cms.request.write`

Parameters

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

Returns a status indicating success or error.

findCredentials(*expressions*, *transactionID*)

Retrieve a list of credentials that match the passed in search criteria.

Authorization: `cms.credential.read`

Parameters

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

Returns a status indicating success or error, in the case of success the list of matching credentials.

readCredential(*credentialID*, *attributes*, *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: `cms.credential.read`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **attributes** (*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*, *additionalData*, *transactionID*)

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

Authorization: `cms.credential.write`

Parameters

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

- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for suspension.
- **transactionID** (*string*) – The (optional) client generated transactionID.

Returns a status indicating success or error.

unsuspendCredential(*credentialID, additionalData, transactionID*)

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

Authorization: `cms.credential.write`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for unsuspension.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

revokeCredential(*credentialID, additionalData, transactionID*)

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

Authorization: `cms.credential.write`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **additionalData** (*dict*) – Additional data relating to the request, e.g. reason for revocation.
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

setCredentialStatus(*credentialID, status, reason, requester, comment, transactionID*)

Change the status of a credential. This is an extension of the revoke/suspend services, supporting more statuses and transitions.

Authorization: `cms.credential.write`

Parameters

- **credentialID** (*str*) – The ID of the credential.
- **status** (*string*) – The new status of the credential
- **reason** (*string*) – A text describing the cause of the change of status
- **requester** (*string*) – The client generated transactionID.
- **comment** (*string*) – A free text comment
- **transactionID** (*string*) – The client generated transactionID.

Returns a status indicating success or error.

findCredentialProfiles(*expressions, transactionID*)

Retrieve a list of credential profiles that match the passed in search criteria

Authorization: `cms.profile.read`

Parameters

- **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

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

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

Attributes

The “attributes” 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. @@ -128,7 +128,7 @@ attributes to retrieve.

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

Some calls may require new 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.

Data Model

Table 6.17: Credential Data Model

Type	Description	Example
Credential	<p>The attributes of the credential itself</p> <p>The proposed transitions for the status are represented below. It can be adapted if needed.</p> <pre> graph TD Start(()) --> new new -- issue --> active active -- suspend --> suspended suspended -- unsuspend --> active active -- revoke --> revoked suspended -- revoke --> revoked </pre>	ID, status, dates, serial number
Biometric Data	<p>Digital representation of biometric characteristics.</p> <p>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, palm-print, portrait, iris or signature.</p> <p>A biometric data can be associated to no image or a partial image if it includes information about the missing items (example: one finger may be amputated on a 4 finger image)</p>	fingerprint, portrait, iris, signature
Biographic Data	a dictionary (list of names and values) giving the biographic data of interest for the biometric services.	first name, last name, date of birth, etc.
Request Data	a dictionary (list of names and values) for data related to the request itself.	Type of credential, action to execute, priority

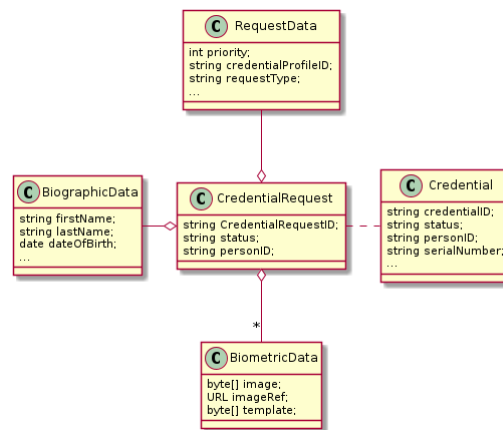


Fig. 6.19: Credential Data Model

6.7 ID Usage

The ID usage component MAY implement the following interfaces:

6.7.1 ID Usage

ID Usage consists of a set of services implemented on top of identity systems to favour third parties consumption of identity data. The services can be classified in three sets:

- Relying Party API: submitting citizen ID attributes for validation

The purpose of the Relying Party (RP) API is to extend the use of government-issued identity to registered third party services. The individual will submit their ID attributes to the relying party in order to enroll for, or access, a particular service. The relying party will leverage the RP API to access the identity management system and verify the individual's identity. In this way, external relying parties can quickly and easily verify individuals based on their government issued ID attributes.

Use case applications: telco enrolment

The RP API enables a telco operator to check an individual's identity when applying for a service contract. The telco relies on the government to confirm that the attributes submitted by the individual match against the data held in the database therefore being able to confidently identify the new subscriber. This scenario can be replicated across multiple sectors including banking and finance.

- Digital Credential Management API: delegating digital issuance to third parties

The purpose of the Digital Credential Management (DCM) API is to enable external wallet providers to manage government issued digital credentials distribution, storage and usage.

Use case applications: digital driver license

The DCM API enables individuals to request a digital driver license as a digital credential in their selected wallet to use for online and offline identification. The user initiates a request for digital driver license using the Digital Credential Distribution System (DCDS) frontend, which sends the request to the identity management system. The Credential Management System (CMS) then issues the digital credential by dedicated API endpoint of the DCDS.

- Federation API: user-initiated attributes sharing

The purpose of the federation API is to enable the user to share their attributes with a chosen relying party using well-known internet protocol: OpenID Connect. The relying party benefits from the government's verified attributes.

Use case applications: on-line registration to gambling website

The Federation API enables individuals to log-in with their government credential (log-in/password) and share verified attributes ex. age (above 18) with the relying party.

Relying Party API

verifyIdentity(*Identifier, attributeSet*)

Verify an Identity based on an identifier (UIN, token...) and a set of Identity Attributes. Verification is strictly matching all provided identity attributes to compute the global Boolean matching result.

Authorization: *id.verify*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **attributeSet** (*list[str]*) – A set of identity attributes associated to the identifier and to be verified by the system

Returns Y or N

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

identify(*attributeSet, outputAttributeSet*)

Identify possibly matching identities against an input set of attributes. Returns an array of predefined datasets as described by outputAttributeSet.

Note: This service may be limited to some specific government RPs

Authorization: *id.identify*

Parameters

- **attributeSet** (*list[str]*) – A list of pair (name,value) requested
- **outputAttributeSet** (*list[str]*) – An array of attributes requested

Returns Y or N

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

readAttributes(*Identifier, outputAttributeSet*)

Get a list of identity attributes attached to a given identifier.

Authorization: *id.read*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **outputAttributeSet** (*list[str]*) – defining the identity attributes to be provided back to the caller

Returns An array of the requested attributes

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

readAttributeSet(*Identifier, AttributeSetName*)

Get a set of identity attributes as defined by attributeSet, attached to a given identifier.

Authorization: *id.set.read*

Parameters

- **Identifier** (*str*) – The person's Identifier
- **attributeSetName** (*str*) – The name of predefined attributes set name

Returns An array of the requested attributes

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

Attribute set

When identity attributes are exchanged, they are included in an attribute set, possibly containing groups like biographic data, biometric data, document data, contact data. . . This structure is extensible and may be complemented with other data groups, and each group may contain any number of attribute name / attribute value pairs.

Attribute set name

Attribute sets are by definition structures with variable and optional content, hence it may be useful to pre-agree on a given attribute set content and name between two or more systems in a given project scope.

Any string may be used to define an attribute set name, but in the scope of this specification following names are reserved and predefined:

Name	Description	Data Included
"DEFAULT_SET_01"	Minimum demographic data	First name Last name DoB Place of birth
"DEFAULT_SET_02"	Minimum demographic and portrait	Minimum demographic data + portrait
"DEFAULT_SET_EIDAS"	Set expected to comply with eIDAS pivotal attributes.	TBD

Output Attribute set

To specify what identity attributes are expected in return when performing e.g. an identify request or a read attributes.

ANNEXES

7.1 Glossary

ABIS Automated Biometric Identification System

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

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.

Gallery Group of persons related by a common purpose, designation, or status. Example: a watch list or a set of persons entitled to a certain benefit.

(Source: ISO-30108-1)

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

Mime Types Mime type definitions are spread across several resources. The mime type definitions should be in compliance with [RFC 6838](#).

Some examples of possible mime type definitions:

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

OSIA Open Standard Identity APIs

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

UIN Unique Identity Number.

7.2 Data Format

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

This is version 1.2.0 of this interface.

Get the OpenAPI file: [notification.yaml](#)

Notification Services

aggedright

- *confirm*
- *createTopic*
- *listTopics*
- *deleteTopic*
- *publish*
- *subscribe*
- *listSubscription*
- *unsubscribe*

Services

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.

Scope required: notif.sub.write

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. Constraints: possible values are `http`, `email`.
- **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.
- **400 Bad Request** – Bad request.
- **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/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Callback: onEvent

POST {\$request.query.address}

Form Parameters

- **body** – The message.

Request Headers

- **message-type** – the type of the message. Constraints: possible values are SubscriptionConfirmation, Notification. (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)

Status Codes

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

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": "2022-01-03T16:43:26.913783"
}
```

Example response:

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

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

GET /v1/subscriptions

Get all subscriptions

Scope required: notif.sub.read

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

Scope required: notif.sub.write

Parameters

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

Status Codes

- 204 No Content – Subscription successfully removed.
- 400 Bad Request – Bad request.
- 404 Not Found – Subscription not found.
- 500 Internal Server Error – Unexpected error.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

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

Scope required: notif.sub.write

Query Parameters

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

Status Codes

- 200 OK – Subscription successfully confirmed.
- 400 Bad Request – 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/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

```

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

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

```

Publisher

POST /v1/topics

Create a topic

Create a new topic. This service is idempotent.

Scope required: notif.topic.write

Query Parameters

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

Status Codes

- 200 OK – Topic was created.
- 400 Bad Request – Bad request.
- 500 Internal Server Error – Unexpected error.

Example response:

```

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

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

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

GET /v1/topics

Get all topics

Scope required: notif.topic.read

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

Scope required: notif.topic.write**Parameters**

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

Status Codes

- **204 No Content** – Topic successfully removed.
- **400 Bad Request** – Bad request.
- **404 Not Found** – Topic not found.
- **500 Internal Server Error** – Unexpected error.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

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.

Scope required: notif.topic.publish

Parameters

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

Query Parameters

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

Form Parameters

- **body** – Message posted.

Status Codes

- 200 OK – Notification published.
- 400 Bad Request – Bad request.
- 500 Internal Server Error – Unexpected error.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Receiver

POST notify_URL

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)

Status Codes

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

Example request (Subscription Confirmation):

```
POST notify_URL HTTP/1.1
Host: example.com
Content-Type: application/json
Message-Type: SubscriptionConfirmation
Subscription-Id: XXX
Message-Id: YYY
Topic-ID: ZZZ
```

(continues on next page)

(continued from previous page)

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

Example request (Event):

```
POST notify_URL HTTP/1.1
Host: example.com
Content-Type: application/json
Message-Type: Notification
Message-Id: YYY
Topic-ID: ZZZ

{"key": "data"}
```

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"> • source: identification of the system emitting the event • uin of the new born • uin1 of the first parent (optional if parent is unknown) • uin2 of the second parent (optional if parent is unknown) <p>Example:</p> <pre>{ "source": "systemX", "uin": "123456789", "uin1": "123456789", "uin2": "234567890" }</pre>

continues on next page

Table 7.1 – continued from previous page

Event Type	Message
death	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin of the dead person <p>Example:</p> <pre>{ "source": "systemX", "uin": "123456789" }</pre>
birthCancellation	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin of the person whose birth declaration is being cancelled <p>Example:</p> <pre>{ "source": "systemX", "uin": "123456789", }</pre>
foetalDeath	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin of the new born <p>Example:</p> <pre>{ "source": "systemX", "uin": "123456789" }</pre>
marriage	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin1 of the first conjoint • uin2 of the second conjoint <p>Example:</p> <pre>{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }</pre>
divorce	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin1 of the first conjoint • uin2 of the second conjoint <p>Example:</p> <pre>{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }</pre>
annulment	<ul style="list-style-type: none"> • source: identification of the system emitting the event • uin1 of the first conjoint • uin2 of the second conjoint <p>Example:</p> <pre>{ "source": "systemX", "uin1": "123456789", "uin2": "234567890" }</pre>

continues on next page

Table 7.1 – continued from previous page

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

continues on next page

Table 7.1 – continued from previous page

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

This is version 1.2.0 of this interface.

Get the OpenAPI file: [uin.yaml](#)

UIN Management

aggregatedright

• *generateUIN*

Services

default

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.

Scope required: uin.generate

Query Parameters

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

Form Parameters

- **body** – A set of attributes for the person.

Status Codes

- 200 OK – UIN is generated.
- 400 Bad Request – Unexpected error.
- 401 Unauthorized – Client must be authenticated.
- 403 Forbidden – Service forbidden.
- 500 Internal Server Error – Unexpected error.

Example request:

```
POST /v1/uin?transactionId=string 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

"1235567890"
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

7.3.3 Data Access

This is version 1.3.0 of this interface.

Get the OpenAPI file: [dataaccess.yaml](#)

Data Access Services

aggedright

- *queryPersonList*
- *readPersonAttributes*

- *matchPersonAttributes*
- *verifyPersonAttributes*
- *readDocument*

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>

Scope required: `pr.person.read` or `cr.person.read`

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.
- **offset** (*integer*) – The offset of the query (first item of the response). Default: 0.
- **limit** (*integer*) – The maximum number of items to return. Default: 100.

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&names=firstName&names=lastName&offset=1&limit=1 HTTP/1.1
Host: example.com
```

Example response:

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

[
  "string"
]
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/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/{uin}

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

Scope required: `pr.person.read` or `cr.person.read`

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.
- 400 Bad Request – Invalid parameter.
- 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/1.1 400 Bad Request
Content-Type: application/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/{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.

Scope required: `pr.person.match` or `cr.person.match`

Parameters

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

Form Parameters

- **body** – A set of attributes for the person.

Status Codes

- **200 OK** – Information about non matching attributes. Returns a list of matching result. An empty list indicates all attributes were matching. .
- **400 Bad Request** – Invalid parameter.
- **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/1.1 400 Bad Request
Content-Type: application/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/{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.

Scope required: `pr.person.verify` or `cr.person.verify`

Parameters

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

Form Parameters

- **body** – A set of expressions on attributes of the person.

Status Codes

- **200 OK** – The expressions are all true (true is returned) or one is false (false is returned).
- **400 Bad Request** – Invalid parameter.
- **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/1.1 400 Bad Request
Content-Type: application/json

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

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>

Scope required: `pr.document.read` or `cr.document.read`

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. . Constraints: possible values are pdf, jpeg, png, TBD. (Required)

Status Codes

- **200 OK** – The document(s) is/are found and returned, as binary data in a MIME multipart structure.
- **400 Bad Request** – Invalid parameter.
- **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?secondaryUin=string&doctype=string&format=pdf HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

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 Enrollment

This is version 1.2.0 of this interface.

Get the OpenAPI file: [enrollment.yaml](#)

Enrollment Services

aggregatedright

- *createEnrollment*
- *readEnrollment*
- *updateEnrollment*
- *partialUpdateEnrollment*
- *deleteEnrollment*
- *finalizeEnrollment*
- *findEnrollments*
- *createBuffer*
- *readBuffer*

Services

Enrollment

POST /v1/enrollments/{enrollmentId}

Create one enrollment

Scope required: enroll.write

Parameters

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

- **finalize** (*boolean*) – Flag to indicate that data was collected (default is false). Object of type boolean.
- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)

Form Parameters

- **body** – Object of type *Enrollment*.

Status Codes

- 204 No Content – Operation successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/enrollments/{enrollmentId}?finalize=True&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "enrollmentType": "string",
  "enrollmentFlags": {
    "timeout": 3600,
    "other": "other",
    "...": "..."
  },
  "requestData": {
    "priority": 1,
    "requestType": "FIRST_ISSUANCE",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    },
    "...": "..."
  },
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
],
"documentData": [
  {
    "documentType": "FORM",
    "documentTypeOther": "string",
    "instance": "string",
    "parts": [
      {
        "pages": [
          1
        ],
        "data": "c3RyaW5n",
        "dataRef": "http://server.com/buffer?id=00003",
        "width": 1,
        "height": 1,
        "mimeType": "string",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string"
      }
    ]
  }
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

GET /v1/enrollments/{enrollmentId}**Read one enrollment****Scope required:** enroll.read**Parameters**

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **attributes** (*array*) – The (optional) set of required attributes to retrieve. If not present all attributes will be returned. Array of string.

Status Codes

- 200 OK – Read successful. Object of type *Enrollment*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/enrollments/{enrollmentId}?transactionId=string&attributes=%5B%27string%27%5D HTTP/1.1
Host: example.com
```

Example response:

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

{
  "enrollmentId": "string",
  "status": "FINALIZED",
  "enrollmentType": "string",
  "enrollmentFlags": {
    "timeout": 3600,
    "other": "other",
    "...": "..."
  },
  "requestData": {
    "priority": 1,
    "requestType": "FIRST_ISSUANCE",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    },
    "...": "..."
  },
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {

```

(continues on next page)

(continued from previous page)

```

        "biometricSubType": "RIGHT_INDEX",
        "presence": "BANDAGED"
    }
],
"metadata": "string",
"comment": "string",
"template": "c3RyaW5n",
"templateRef": "http://dataserver.com/template?id=00014",
"templateFormat": "string",
"quality": 1,
"qualityFormat": "string",
"algorithm": "string",
"vendor": "string"
}
],
"documentData": [
    {
        "documentType": "FORM",
        "documentTypeOther": "string",
        "instance": "string",
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "http://server.com/buffer?id=00003",
                "width": 1,
                "height": 1,
                "mimeType": "string",
                "captureDate": "2019-05-21T12:00:00Z",
                "captureDevice": "string"
            }
        ]
    }
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

PUT /v1/enrollments/{enrollmentId}**Update one enrollment****Scope required:** enroll.write**Parameters**

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

- **finalize** (*boolean*) – Flag to indicate that data was collected (default is false). Object

of type boolean.

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

Form Parameters

- **body** – Object of type *Enrollment*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
PUT /v1/enrollments/{enrollmentId}?finalize=True&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "enrollmentType": "string",
  "enrollmentFlags": {
    "timeout": 3600,
    "other": "other",
    "...": "..."
  },
  "requestData": {
    "priority": 1,
    "requestType": "FIRST_ISSUANCE",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    },
    "...": "..."
  },
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
```

(continues on next page)

(continued from previous page)

```

        {
            "biometricSubType": "RIGHT_INDEX",
            "presence": "BANDAGED"
        }
    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
}
],
"documentData": [
    {
        "documentType": "FORM",
        "documentTypeOther": "string",
        "instance": "string",
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "http://server.com/buffer?id=00003",
                "width": 1,
                "height": 1,
                "mimeType": "string",
                "captureDate": "2019-05-21T12:00:00Z",
                "captureDevice": "string"
            }
        ]
    }
]
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

PATCH /v1/enrollments/{enrollmentId}**Update partially one enrollment**

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

Scope required: enroll.write

Parameters

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

- **finalize** (*boolean*) – Flag to indicate that data was collected (default is false). Object of type *boolean*.
- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)

Form Parameters

- **body** – Object of type *Enrollment*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
PATCH /v1/enrollments/{enrollmentId}?finalize=True&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "enrollmentType": "string",
  "enrollmentFlags": {
    "timeout": 3600,
    "other": "other",
    "...": "..."
  },
  "requestData": {
    "priority": 1,
    "requestType": "FIRST_ISSUANCE",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    },
    "...": "..."
  },
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
```

(continues on next page)

(continued from previous page)

```

    "missing": [
      {
        "biometricSubType": "RIGHT_INDEX",
        "presence": "BANDAGED"
      }
    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
],
"documentData": [
  {
    "documentType": "FORM",
    "documentTypeOther": "string",
    "instance": "string",
    "parts": [
      {
        "pages": [
          1
        ],
        "data": "c3RyaW5n",
        "dataRef": "http://server.com/buffer?id=00003",
        "width": 1,
        "height": 1,
        "mimeType": "string",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string"
      }
    ]
  }
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

DELETE /v1/enrollments/{enrollmentId}**Delete one enrollment****Scope required:** enroll.write**Parameters**

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

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

Status Codes

- 204 No Content – Delete successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

PUT /v1/enrollments/{enrollmentId}/finalize

Finalize one enrollment

Scope required: enroll.write

Parameters

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

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

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

```

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

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

```

POST /v1/enrollments

Retrieve a list of enrollments which match passed in search criteria

Scope required: enroll.read

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **offset** (*integer*) – The offset of the query (first item of the response). Object of type integer. Default: 0.
- **limit** (*integer*) – The maximum number of items to return. Object of type integer. Default: 100.

Form Parameters

- **body** – A set of expressions on attributes of the person. Array of *Expression*.

Status Codes

- 200 OK – Query successful. Array of *Enrollment*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Query not allowed.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

POST /v1/enrollments?transactionId=string&offset=1&limit=1 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

[
  {
    "enrollmentId": "string",
    "status": "FINALIZED",
    "enrollmentType": "string",
    "enrollmentFlags": {
      "timeout": 3600,
      "other": "other",
      "...": "..."
    },
  },

```

(continues on next page)

(continued from previous page)

```

"requestData": {
  "priority": 1,
  "requestType": "FIRST_ISSUANCE",
  "deliveryAddress": {
    "address1": "11 Rue des Rosiers",
    "city": "Libourne",
    "postalCode": "33500",
    "country": "France"
  },
  "...": "..."
},
"contextualData": {
  "enrollmentDate": "2019-01-11",
  "...": "..."
},
"biographicData": {
  "firstName": "John",
  "lastName": "Doo",
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA",
  "...": "..."
},
"biometricData": [
  {
    "biometricType": "FINGER",
    "biometricSubType": "RIGHT_INDEX",
    "instance": "string",
    "image": "c3RyaW5n",
    "imageRef": "http://imageserver.com/image?id=00003",
    "captureDate": "2019-05-21T12:00:00Z",
    "captureDevice": "string",
    "impressionType": "LIVE_SCAN_PLAIN",
    "width": 1,
    "height": 1,
    "bitdepth": 1,
    "mimeType": "string",
    "resolution": 1,
    "compression": "WSQ",
    "missing": [
      {
        "biometricSubType": "RIGHT_INDEX",
        "presence": "BANDAGED"
      }
    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
],
"documentData": [
  {
    "documentType": "FORM",
    "documentTypeOther": "string",
    "instance": "string",
    "parts": [
      {
        "pages": [
          1
        ],
        "data": "c3RyaW5n",
        "dataRef": "http://server.com/buffer?id=00003",
        "width": 1,
        "height": 1,
        "mimeType": "string",

```

(continues on next page)

(continued from previous page)

```

        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string"
      }
    ]
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

Buffer**POST /v1/enrollments/{enrollmentId}/buffer****Create a buffer**

This service is used to send separately the buffers of the images

Scope required: enroll.buf.write

Parameters

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.

Query Parameters

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

Form Parameters

- **body** – The image of the request.

Request Headers

- **Digest** – the buffer digest, as defined per RFC 3230. Object of type string.

Status Codes

- **201 Created** – Operation successful. Object of type string.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Operation not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/enrollments/{enrollmentId}/buffer?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/*
Digest: SHA=thvDyvvhfIqlvFe+A9MYgxAfm1q5=

ABCDEFG...
```

Example request:

```
POST /v1/enrollments/{enrollmentId}/buffer?transactionId=string HTTP/1.1
Host: example.com
Content-Type: image/*
Digest: SHA=thvDyvvhfIqlvFe+A9MYgxAfm1q5=

ABCDEFG...
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "bufferId": "string"
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

GET /v1/enrollments/{enrollmentId}/buffer/{bufferId}**Read a buffer**

This service is used to get the buffer of the images. The content type of the response is the content type used when the buffer was created.

Scope required: enroll.buf.read

Parameters

- **enrollmentId** (*string*) – the id of the enrollment. Object of type string.
- **bufferId** (*string*) – the id of the buffer. Object of type string.

Query Parameters

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

Response Headers

- **Digest** – the buffer digest, as defined per RFC 3230. Object of type string.

Status Codes

- **200 OK** – Read successful.

- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/enrollments/{enrollmentId}/buffer/{bufferId}?transactionId=string HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/*
transactionId: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=
ABCDEFG...
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: image/*
transactionId: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=
ABCDEFG...
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
transactionId: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=

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

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
transactionId: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=

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

Data Model**Error**

Table 7.6: Error

Attribute	Type	Description	Required
code	integer/int32	Error code.	Yes
message	string	Error message.	Yes

Enrollment

Table 7.7: Enrollment

Attribute	Type	Description	Required
enrollmentId	string	Constraints: read only	Yes
status	string	Constraints: possible values are FINALIZED, IN_PROGRESS; read only	Yes
enrollmentType	string	Type of the enrollment (example: citizen, resident, etc.).	
enrollmentFlags	Object of type <i>EnrollmentFlags</i>	The enrollment custom flags (i.e. the properties of the enrollment process). Can be extended.	
requestData	Object of type <i>RequestData</i>	The data describing the request associated to the enrollment (i.e. why the enrollment is done). Can be extended.	
contextualData	Object of type <i>ContextualData</i>		
biographicData	Object of type <i>BiographicData</i>	The set of biographic data.	
biometricData	Array of <i>BiometricData</i>		
documentData	Array of <i>DocumentData</i>		

RequestData

The data describing the request associated to the enrollment (i.e. why the enrollment is done). Can be extended.

Table 7.8: RequestData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "priority": 1,
  "requestType": "FIRST_ISSUANCE",
  "deliveryAddress": {
    "address1": "11 Rue des Rosiers",
    "city": "Libourne",
    "postalCode": "33500",
    "country": "France"
  }
}
```

EnrollmentFlags

The enrollment custom flags (i.e. the properties of the enrollment process). Can be extended.

Table 7.9: EnrollmentFlags

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "timeout": 3600,
  "other": "other"
}
```

ContextualData

Table 7.10: ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "enrollmentDate": "2019-01-11"
}
```

BiographicData

The set of biographic data.

Table 7.11: BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "firstName": "John",
  "lastName": "Doo",
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA"
}
```

BiometricData

Table 7.12: BiometricData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes

continues on next page

Table 7.12 – continued from previous page

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
image	string/byte	Base64-encoded image.	
imageRef	string/uri	URI to an image.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the biometric.	
impressionType	string	Constraints: possible values are LIVE_SCAN_PLAIN, LIVE_SCAN_ROLLED, NONLIVE_SCAN_PLAIN, NONLIVE_SCAN_ROLLED, LATENT_IMPRESSION, LATENT_TRACING, LATENT_PHOTO, LATENT_LIFT, LIVE_SCAN_SWIPE, LIVE_SCAN_VERTICAL_ROLL, LIVE_SCAN_PALM, NONLIVE_SCAN_PALM, LATENT_PALM_IMPRESSION, LATENT_PALM_TRACING, LATENT_PALM_PHOTO, LATENT_PALM_LIFT, LIVE_SCAN_OPTICAL_CONTACTLESS_PLAIN, OTHER, UNKNOWN	
width	integer	the width of the image.	

continues on next page

Table 7.12 – continued from previous page

Attribute	Type	Description	Required
height	integer	the height of the image.	
bitdepth	integer		
mimeType	string	the nature and format of the image. The mime type definitions should be in compliance with RFC 6838.	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of <i>MissingType</i>	Optional properties indicating if a part of the biometric data is missing.	
metadata	string	An optional string used to convey information vendor-specific.	
comment	string	A comment about the biometric data.	
template	string/byte	Base64-encoded template.	
templateRef	string/uri	URI to the template when it is managed in a dedicated data server.	
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table 7.13: MissingType

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

DocumentPart

Table 7.14: DocumentPart

Attribute	Type	Description	Required
pages	Array of integer	The pages included in this part. Can be a single page number, or a list. Constraints: minItems is 1	
data	string/byte	Base64-encoded data of the document.	
dataRef	string/uri	URI to the data.	
width	integer	the width of the image in pixels.	
height	integer	the height of the image in pixels.	
contentType	string	the nature and format of the document. The mime type definitions should be in compliance with RFC 6838.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the document part.	

DocumentData

Table 7.15: DocumentData

Attribute	Type	Description	Required
documentType	string	Type of document. Constraints: possible values are ID_CARD, PASSPORT, INVOICE, BIRTH_CERTIFICATE, FORM, OTHER	Yes
documentTypeOther	string	Details about the type of document when OTHER is used.	
instance	string	Used to separate two distincts documents of the same type (ex: two passports).	
parts	Array of <i>Document-Part</i>	Constraints: minItems is 1	Yes

Expression

Table 7.16: Expression

Attribute	Type	Description	Required
attributeName	string		Yes
operator	string	Constraints: possible values are <, >, =, >=, <=, !=	Yes
value	One of string, integer, number, boolean		Yes

Expressions

Table 7.17: Expressions

Attribute	Type	Description	Required
N/A	Array of <i>Expression</i>		

7.3.5 Population Registry Management

This is version 1.4.0 of this interface.

Get the OpenAPI file: [pr.yaml](#)

Population Registry Services

aggedright

- *findPersons*
- *createPerson*
- *readPerson*
- *updatePerson*
- *deletePerson*
- *mergePerson*
- *readIdentities*
- *createIdentity*
- *createIdentityWithId*
- *readIdentity*
- *updateIdentity*
- *partialUpdateIdentity*
- *deleteIdentity*
- *moveIdentity*
- *setIdentityStatus*
- *defineReference*
- *readReference*
- *readGalleries*
- *readGalleryContent*

Services

Person

POST /v1/persons

Query for persons

Retrieve a list of personId corresponding to the records with one identity matching the criteria.

By default, all identities are used in the search.

Scope required: `pr.person.read`

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **group** (*boolean*) – Group all matching identities of one person and return only the personId. Object of type *boolean*.
- **reference** (*boolean*) – Limit the query to the reference identity. Object of type *boolean*.
- **gallery** (*string*) – Limit the query to the records belonging to this gallery. Object of type *string*.
- **offset** (*integer*) – The offset of the query (first item of the response). Object of type *integer*. Default: 0.
- **limit** (*integer*) – The maximum number of items to return. Object of type *integer*. Default: 100.

Form Parameters

- **body** – A set of expressions on attributes of the person's identity. Array of *Expression*.

Status Codes

- **200 OK** – Query successful. If the group parameter was set the identityId is not included in the response. Object of type *Array*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Query not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/persons?transactionId=string&group=True&reference=True&gallery=string&offset=1&limit=1 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

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

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}

Create one person

Scope required: pr.person.write

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Form Parameters

- **body** – Object of type *Person*.

Status Codes

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

Example request:

```

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

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

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}

Read one person

Scope required: `pr.person.read`

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Status Codes

- **200 OK** – Read successful. Object of type *Person*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Read not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *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": "string",
  "status": "ACTIVE",
  "physicalStatus": "DEAD"
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}

Update one person

Scope required: pr.person.write

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Form Parameters

- **body** – Object of type *Person*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

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

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

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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 identities

Scope required: pr.person.write

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Status Codes

- 204 No Content – Delete successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Delete not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/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/{personIdTarget}/merge/{personIdSource}**Merge two persons**

Merge two person records into a single one. Identity ID are preserved and in case of duplicates an error is returned and no changes are done. If the operation is successful, the person merged is deleted.

Scope required: `pr.person.write`

Parameters

- **personIdTarget** (*string*) – the id of the person receiving new identities. Object of type *string*.
- **personIdSource** (*string*) – the id of the person giving the identities. Object of type *string*.

Query Parameters

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

Status Codes

- 204 No Content – Merge successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Merge not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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

Scope required: pr.identity.read

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Status Codes

- 200 OK – Operation successful. Array of *Identity*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *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",
    "identityType": "string",
    "status": "CLAIMED",
    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "enrollmentDate": "2019-01-11",
      "...": "..."
    },
    "biographicData": {

```

(continues on next page)

(continued from previous page)

```

    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "identityId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ],
  "documentData": [
    {
      "documentType": "FORM",
      "documentTypeOther": "string",
      "instance": "string",
      "parts": [
        {
          "pages": [
            1
          ],
          "data": "c3RyaW5n",
          "dataRef": "http://server.com/buffer?id=00003",
          "width": 1,
          "height": 1,
          "mimeType": "string",
          "captureDate": "2019-05-21T12:00:00Z",
          "captureDevice": "string"
        }
      ]
    }
  ]
}
]

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

```

(continues on next page)

(continued from previous page)

```
{
  "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}/identities

Create one identity and generate its id

Scope required: pr.identity.write

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Form Parameters

- **body** – Object of type *Identity*.

Status Codes

- 200 OK – Insertion successful. Object of type *string*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Insertion not allowed.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

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

{
  "identityType": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",

```

(continues on next page)

(continued from previous page)

```

        "instance": "string",
        "image": "c3RyaW5n",
        "imageRef": "http://imageserver.com/image?id=00003",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",
        "resolution": 1,
        "compression": "WSQ",
        "missing": [
            {
                "biometricSubType": "RIGHT_INDEX",
                "presence": "BANDAGED"
            }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
],
"documentData": [
    {
        "documentType": "FORM",
        "documentTypeOther": "string",
        "instance": "string",
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "http://server.com/buffer?id=00003",
                "width": 1,
                "height": 1,
                "mimeType": "string",
                "captureDate": "2019-05-21T12:00:00Z",
                "captureDevice": "string"
            }
        ]
    }
]
}

```

Example response:

```

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

{
  "identityId": "string"
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}/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.

Scope required: pr.identity.write

Parameters

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

Query Parameters

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

Form Parameters

- **body** – Object of type *Identity*.

Status Codes

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

Example request:

```

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

{
  "identityType": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",

```

(continues on next page)

(continued from previous page)

```

    "imageRef": "http://imageserver.com/image?id=00003",
    "captureDate": "2019-05-21T12:00:00Z",
    "captureDevice": "string",
    "impressionType": "LIVE_SCAN_PLAIN",
    "width": 1,
    "height": 1,
    "bitdepth": 1,
    "mimeType": "string",
    "resolution": 1,
    "compression": "WSQ",
    "missing": [
      {
        "biometricSubType": "RIGHT_INDEX",
        "presence": "BANDAGED"
      }
    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
],
"documentData": [
  {
    "documentType": "FORM",
    "documentTypeOther": "string",
    "instance": "string",
    "parts": [
      {
        "pages": [
          1
        ],
        "data": "c3RyaW5n",
        "dataRef": "http://server.com/buffer?id=00003",
        "width": 1,
        "height": 1,
        "mimeType": "string",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string"
      }
    ]
  }
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}/identities/{identityId}

Read one identity

Scope required: pr.identity.read

Parameters

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

Query Parameters

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

Status Codes

- **200 OK** – Read successful. Object of type *Identity*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Read not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *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",
  "identityType": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "identityId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,

```

(continues on next page)

(continued from previous page)

```

        "compression": "WSQ",
        "missing": [
            {
                "biometricSubType": "RIGHT_INDEX",
                "presence": "BANDAGED"
            }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
],
"documentData": [
    {
        "documentType": "FORM",
        "documentTypeOther": "string",
        "instance": "string",
        "parts": [
            {
                "pages": [
                    1
                ],
                "data": "c3RyaW5n",
                "dataRef": "http://server.com/buffer?id=00003",
                "width": 1,
                "height": 1,
                "mimeType": "string",
                "captureDate": "2019-05-21T12:00:00Z",
                "captureDevice": "string"
            }
        ]
    }
]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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}/identities/{identityId}

Update one identity

Scope required: pr.identity.write

Parameters

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

Query Parameters

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

Form Parameters

- **body** – Object of type *Identity*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

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

```
{
  "identityType": "string",
  "status": "CLAIMED",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
      }
    ],
    "documentData": [
      {
        "documentType": "FORM",
        "documentTypeOther": "string",
        "instance": "string",
        "parts": [
          {
            "pages": [
              1
            ],
            "data": "c3RyaW5n",
            "dataRef": "http://server.com/buffer?id=00003",
            "width": 1,
            "height": 1,
            "mimeType": "string",
            "captureDate": "2019-05-21T12:00:00Z",
            "captureDevice": "string"
          }
        ]
      }
    ]
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "code": 1,
  "message": "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.

Scope required: pr.identity.write

Parameters

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

Query Parameters

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

Form Parameters

- **body** – Object of type *Identity*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *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/1.1 400 Bad Request
Content-Type: application/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}/identities/{identityId}**Delete one identity****Scope required:** pr.identity.write**Parameters**

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

Query Parameters

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

Status Codes

- 204 No Content – Delete successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Delete not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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/{personIdTarget}/move/{personIdSource}/identities/{identityId}

Move one identity

Move one identity from the source person to the target person. Identity ID is preserved and in case of duplicate an error is returned and no changes are done. The source person is not deleted, even if it was the only identity of this person.

Scope required: `pr.identity.write`

Parameters

- **personIdTarget** (*string*) – the id of the person receiving the identity. Object of type *string*.
- **personIdSource** (*string*) – the id of the person giving the identity. Object of type *string*.
- **identityId** (*string*) – the id of the identity. Object of type *string*.

Query Parameters

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

Status Codes

- **204 No Content** – Move successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Move not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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"
  }

```

PUT /v1/persons/{personId}/identities/{identityId}/status

Change the status of an identity

Scope required: pr.identity.write

Parameters

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

Query Parameters

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

Status Codes

- **204 No Content** – Operation successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Operation not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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

Scope required: pr.reference.write

Parameters

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

Query Parameters

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

Status Codes

- 204 No Content – Operation successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/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}/reference**Read the reference**Scope required: `pr.reference.read`**Parameters**

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

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

Status Codes

- 200 OK – Read successful. Object of type *Identity*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *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",
  "identityType": "string",
  "status": "CLAIMED",
  "galleries": [
```

(continues on next page)

(continued from previous page)

```

    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "identityId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ],
  "documentData": [
    {
      "documentType": "FORM",
      "documentTypeOther": "string",
      "instance": "string",
      "parts": [
        {
          "pages": [
            1
          ],
          "data": "c3RyaW5n",
          "dataRef": "http://server.com/buffer?id=00003",
          "width": 1,
          "height": 1,
          "mimeType": "string",
          "captureDate": "2019-05-21T12:00:00Z",
          "captureDevice": "string"
        }
      ]
    }
  ]
}

```


Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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

Scope required: pr.gallery.read

Query Parameters

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

Status Codes

- 200 OK – Operation successful. Array of string.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 500 Internal Server Error – Unexpected error. Object of type *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/1.1 400 Bad Request
Content-Type: application/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

Scope required: pr.gallery.read

Parameters

- **galleryId** (*string*) – the id of the gallery. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **offset** (*integer*) – The offset of the query (first item of the response). Object of type integer. Default: 0.
- **limit** (*integer*) – The maximum number of items to return. Object of type integer. Default: 1000.

Status Codes

- 200 OK – Operation successful. Object of type Array.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

GET /v1/galleries/{galleryId}?transactionId=string&offset=1&limit=1 HTTP/1.1
Host: example.com

```

Example response:

```

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

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

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

Data Model

Error

Table 7.18: Error

Attribute	Type	Description	Required
code	integer/int32	Error code.	Yes
message	string	Error message.	Yes

Person

Person entity.

Table 7.19: Person

Attribute	Type	Description	Required
personId	string	The unique id for this person. Constraints: read only	
status	string	Constraints: possible values are ACTIVE, INACTIVE	Yes
physicalStatus	string	Constraints: possible values are DEAD, ALIVE	Yes

Identity

Table 7.20: Identity

Attribute	Type	Description	Required
identityId	string	Constraints: read only	Yes
identityType	string		Yes
status	string	Constraints: possible values are CLAIMED, VALID, INVALID, REVOKED	Yes
galleries	Array of string	The list of galleries for this object. Constraints: minItems is 1; items must be unique	
clientData	string/byte		
contextualData	Object of type <i>ContextualData</i>		
biographicData	Object of type <i>BiographicData</i>	The set of biographic data.	
biometricData	Array of <i>BiometricData</i>		
documentData	Array of <i>DocumentData</i>		

ContextualData

Table 7.21: ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "enrollmentDate": "2019-01-11"
}
```

BiographicData

The set of biographic data.

Table 7.22: BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "firstName": "John",
  "lastName": "Doo",
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA"
}
```

DocumentData

Table 7.23: DocumentData

Attribute	Type	Description	Required
documentType	string	Type of document. Constraints: possible values are ID_CARD, PASSPORT, INVOICE, BIRTH_CERTIFICATE, FORM, OTHER	Yes
documentTypeOther	string	Details about the type of document when OTHER is used.	
instance	string	Used to separate two distincts documents of the same type (ex: two passports).	
parts	Array of <i>Document-Part</i>	Constraints: minItems is 1	Yes

DocumentPart

Table 7.24: DocumentPart

Attribute	Type	Description	Required
pages	Array of integer	The pages included in this part. Can be a single page number, or a list. Constraints: minItems is 1	
data	string/byte	Base64-encoded data of the document.	
dataRef	string/uri	URI to the data.	
width	integer	the width of the image in pixels.	
height	integer	the height of the image in pixels.	
contentType	string	the nature and format of the document. The mime type definitions should be in compliance with RFC 6838.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the document part.	

BiometricData

Table 7.25: BiometricData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes

continues on next page

Table 7.25 – continued from previous page

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
identityId	string	the id of the identity owner of this biometric. Constraints: read only	
image	string/byte	Base64-encoded image.	
imageRef	string/uri	URI to an image.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the biometric.	

continues on next page

Table 7.25 – continued from previous page

Attribute	Type	Description	Required
impressionType	string	Constraints: possible values are LIVE_SCAN_PLAIN, LIVE_SCAN_ROLLED, NONLIVE_SCAN_PLAIN, NONLIVE_SCAN_ROLLED, LATENT_IMPRESSION, LATENT_TRACING, LATENT_PHOTO, LATENT_LIFT, LIVE_SCAN_SWIPE, LIVE_SCAN_VERTICAL_ROLL, LIVE_SCAN_PALM, NONLIVE_SCAN_PALM, LATENT_PALM_IMPRESSION, LATENT_PALM_TRACING, LATENT_PALM_PHOTO, LATENT_PALM_LIFT, LIVE_SCAN_OPTICAL_CONTACTLESS_PLAIN, OTHER, UNKNOWN	
width	integer	the width of the image.	
height	integer	the height of the image.	
bitdepth	integer		
mimeType	string	the nature and format of the image. The mime type definitions should be in compliance with RFC 6838.	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of <i>MissingType</i>	Optional properties indicating if a part of the biometric data is missing.	
metadata	string	An optional string used to convey information vendor-specific.	
comment	string	A comment about the biometric data.	
template	string/byte	Base64-encoded template.	
templateRef	string/uri	URI to the template when it is managed in a dedicated data server.	
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table 7.26: MissingType

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

Expression

Table 7.27: Expression

Attribute	Type	Description	Required
attributeName	string		Yes
operator	string	Constraints: possible values are <, >, =, >=, <=, !=	Yes
value	One of string, integer, number, boolean		Yes

Expressions

Table 7.28: Expressions

Attribute	Type	Description	Required
N/A	Array of <i>Expression</i>		

7.3.6 Biometrics

This is version 1.5.0 of this interface.

Get the OpenAPI file: [abis.yaml](#)

Biometrics Services

aggedright

- *createEncounterNoIds*
- *createEncounterNoId*
- *readAllEncounters*
- *createEncounter*
- *readEncounter*
- *updateEncounter*
- *deleteEncounter*
- *mergeEncounter*
- *moveEncounter*
- *updateEncounterStatus*
- *updateEncounterGalleries*
- *readTemplate*
- *deleteAll*
- *identify*
- *identifyFromId*
- *identifyFromEncounterId*
- *verifyFromId*
- *verifyFromBio*
- *readGalleries*
- *readGalleryContent*

Services

CRUD

POST /v1/persons

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

Scope required: abis.encounter.write

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **algorithm** (*string*) – Hint about the algorithm to be used. Object of type string.

Form Parameters

- **body** – Object of type *Encounter*.

Status Codes

- 200 OK – Operation successful. Object of type *ExtendablePersonIds*.
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 400 Bad Request – Bad request. Object of type *Error*.

- 403 Forbidden – Operation not allowed.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/persons?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1&
algorithm=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}
```

Example response:

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

{
  "personId": "string",
  "encounterId": "string",
  "others": {
```

(continues on next page)

(continued from previous page)

```

    "...": "...
  }
}

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "...
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Result of the creation. Object of type *ExtendablePersonIds*.

Status Codes

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

Example request:

```

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

{

```

(continues on next page)

(continued from previous page)

```

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

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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.

Scope required: abis.encounter.write

Parameters

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.
- **algorithm** (*string*) – Hint about the algorithm to be used. Object of type *string*.

Form Parameters

- **body** – Object of type *Encounter*.

Status Codes

- **200 OK** – creation successful. Object of type *ExtendablePersonIds*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Creation not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/persons/{personId}/encounters?transactionId=string&callback=http%3A%2F%2Fclient.com
↪%2Fcallback&priority=1&algorithm=string HTTP/1.1
```

Host: example.com

Content-Type: application/json

```
{
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}
```

Example response:

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

```
{
  "personId": "string",
  "encounterId": "string",
  "others": {
    "...": "..."
  }
}
```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Result of the creation. Object of type *ExtendablePersonIds*.

Status Codes

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

Example request:

```

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

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

```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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****Scope required:** abis.encounter.read**Parameters**

- **personId** (*string*) – the id of the person. Object of type *string*.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.

Status Codes

- **200 OK** – Read successful. Array of *Encounter*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Read not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/persons/{personId}/encounters?transactionId=string&callback=http%3A%2F%2Fclient.com
↪%2Fcallback&priority=1 HTTP/1.1
Host: example.com
```

Example response:

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

[
  {
    "encounterId": "string",
    "status": "ACTIVE",
    "encounterType": "string",
```

(continues on next page)

(continued from previous page)

```

    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "enrollmentDate": "2019-01-11",
      "...": "..."
    },
    "biographicData": {
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA",
      "...": "..."
    },
    "biometricData": [
      {
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "encounterId": "string",
        "image": "c3RyaW5n",
        "imageRef": "http://imageserver.com/image?id=00003",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",
        "resolution": 1,
        "compression": "WSQ",
        "missing": [
          {
            "biometricSubType": "RIGHT_INDEX",
            "presence": "BANDAGED"
          }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
      }
    ]
  }
]

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "code": 1,

```

(continues on next page)

(continued from previous page)

```

    "message": "string"
  }

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Encounter data. Array of *Encounter*.

Status Codes

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

Example request:

```

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

```

```

[
  {
    "status": "ACTIVE",
    "encounterType": "string",
    "galleries": [
      "string"
    ],
    "clientData": "c3RyaW5n",
    "contextualData": {
      "enrollmentDate": "2019-01-11",
      "...": "..."
    },
    "biographicData": {
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA",
      "...": "..."
    },
    "biometricData": [
      {
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "image": "c3RyaW5n",
        "imageRef": "http://imageserver.com/image?id=000003",

```

(continues on next page)

(continued from previous page)

```

        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",
        "resolution": 1,
        "compression": "WSQ",
        "missing": [
            {
                "biometricSubType": "RIGHT_INDEX",
                "presence": "BANDAGED"
            }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
}
]

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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/{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.

Scope required: abis.encounter.write

Parameters

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

Query Parameters

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

- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.
- **algorithm** (*string*) – Hint about the algorithm to be used. Object of type *string*.

Form Parameters

- **body** – Object of type *Encounter*.

Status Codes

- **200 OK** – Creation successful. Object of type *ExtendablePersonIds*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Creation not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/persons/{personId}/encounters/{encounterId}?transactionId=string&callback=http%3A%2F
%2Fclient.com%2Fcallback&priority=1&algorithm=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ]
    },
    "metadata": "string",
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
      }
    ]
  }
}

```

Example response:

```

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

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

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type string. (Required)

- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Form Parameters

- **body** – Result of the creation. Object of type *ExtendablePersonIds*.

Status Codes

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

Example request:

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

{
  "personId": "string",
  "encounterId": "string",
  "others": {
    "...": "..."
  }
}
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.encounter.read

Parameters

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

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- 200 OK – Read successful. Object of type *Encounter*.
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/persons/{personId}/encounters/{encounterId}?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1 HTTP/1.1
Host: example.com
```

Example response:

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

{
  "encounterId": "string",
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
```

(continues on next page)

(continued from previous page)

```

        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
      }
    ]
  }
}

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Encounter data. Object of type *Encounter*.

Status Codes

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

Example request:

```

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

{
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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"
}
```

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

Update one encounter

Scope required: abis.encounter.write

Parameters

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

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **algorithm** (*string*) – Hint about the algorithm to be used. Object of type string.

Form Parameters

- **body** – Object of type *Encounter*.

Status Codes

- **200 OK** – Operation successful. Object of type *ExtendablePersonIds*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **204 No Content** – Update successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Update not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
PUT /v1/persons/{personId}/encounters/{encounterId}?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1&algorithm=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "status": "ACTIVE",
  "encounterType": "string",
  "galleries": [
    "string"
  ],
  "clientData": "c3RyaW5n",
  "contextualData": {
    "enrollmentDate": "2019-01-11",
    "...": "..."
  },
  "biographicData": {
    "dateOfBirth": "1985-11-30",
```

(continues on next page)

(continued from previous page)

```

    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}

```

Example response:

```

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

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

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

```

(continues on next page)

(continued from previous page)

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

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Result of the update. Object of type *ExtendablePersonIds*.

Status Codes

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

Example request:

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

{
  "personId": "string",
  "encounterId": "string",
  "others": {
    "...": "..."
  }
}
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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.

Scope required: abis.encounter.write

Parameters

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

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **204 No Content** – Delete successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Delete not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:

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

"OK"
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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/{personIdTarget}/merge/{personIdSource}

Merge two sets of encounters

Merge two sets of encounters into a single set. Merging a set of N encounters with a set of M encounters will result in a single set of $N+M$ encounters. Encounter ID are preserved and in case of duplicates an error is returned and no changes are done.

Scope required: abis.encounter.write

Parameters

- **personIdTarget** (*string*) – the id of the person receiving new encounters. Object of type string.
- **personIdSource** (*string*) – the id of the person giving the encounters. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **204 No Content** – Merge successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Merge not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Callback: mergeResponse

POST \${request.query.callback}
Merge two persons response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:

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

"OK"
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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/{personIdTarget}/move/{personIdSource}/encounters/{encounterId}

Move one encounter

Move one encounter from the source person to the target person. Encounter ID is preserved and in case of duplicate an error is returned and no changes are done.

Scope required: abis.encounter.write

Parameters

- **personIdTarget** (*string*) – the id of the person receiving the encounter. Object of type string.
- **personIdSource** (*string*) – the id of the person giving the encounter. Object of type string.
- **encounterId** (*string*) – the id of the encounter. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- 202 **Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 204 **No Content** – Move successful.
- 400 **Bad Request** – Bad request. Object of type *Error*.
- 403 **Forbidden** – Move not allowed.
- 404 **Not Found** – Unknown record.
- 500 **Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Callback: mergeResponse

POST \${request.query.callback}

Merge two persons response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:


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

"OK"
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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}/status

Update status of an encounter

Scope required: abis.encounter.write

Parameters

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

Query Parameters

- **status** (*string*) – New status of encounter. Object of type *string*. (Required)
- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.

Status Codes

- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **204 No Content** – Status has been updated.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Encounter status update not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Callback: updateEncounterStatusResponse**POST** \${request.query.callback}

Update encounter status response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:

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

"OK"
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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}/galleries

Update the galleries of an encounter

This service is used to move one encounter from one gallery to another one without updating the full encounter, which may be resource consuming in a biometric system.

Scope required: abis.encounter.write

Parameters

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

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.

Form Parameters

- **body** – Array of string.

Status Codes

- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **204 No Content** – Galleries have been updated.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Encounter galleries update not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```

PUT /v1/persons/{personId}/encounters/{encounterId}/galleries?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback HTTP/1.1
Host: example.com
Content-Type: application/json

[
  "VIP",
  "CRIMINAL"
]

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

Callback: updateEncounterGalleriesResponse**POST** \${request.query.callback}

Update encounter galleries response callback

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:

```

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

"OK"

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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**Scope required: `abis.encounter.read`**Parameters**

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

Query Parameters

- **biometricType** (*string*) – the type of biometrics to return. Object of type string. Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN.
- **biometricSubType** (*string*) – the sub-type of biometrics to return. Object of type string. Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE.
- **instance** (*string*) – Used to separate two distincts biometric items of the same type and subtype. Object of type string.
- **templateFormat** (*string*) – the format of the template to return. Object of type string. Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.
- **qualityFormat** (*string*) – the format of the quality to return. Object of type string. Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.
- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- **200 OK** – Operation successful. Array of *BiometricComputedData*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Read not allowed.
- **404 Not Found** – Unknown record or unknown biometrics.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/persons/{personId}/encounters/{encounterId}/templates?biometricType=FINGER&
↳biometricSubType=RIGHT_INDEX&instance=string&templateFormat=string&qualityFormat=string&
↳transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1 HTTP/1.1
Host: example.com
```

Example response:

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

[
  {
    "biometricType": "FINGER",
    "biometricSubType": "RIGHT_INDEX",
    "instance": "string",
    "template": "c3RyaW5n",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
]
```

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Callback: readTemplateResponse**POST \${request.query.callback}****Read biometrics templates response callback****Query Parameters**

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Form Parameters

- **body** – Biometric computed data. Array of *BiometricComputedData*.

Status Codes

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

Example request:

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

[
  {
    "biometricType": "FINGER",
    "biometricSubType": "RIGHT_INDEX",
    "instance": "string",
    "template": "c3RyaW5n",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
]
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.encounter.write

Parameters

- **personId** (*string*) – the id of the person. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.

Status Codes

- 202 **Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 204 **No Content** – Delete successful.
- 400 **Bad Request** – Bad request. Object of type *Error*.
- 403 **Forbidden** – Delete not allowed.
- 404 **Not Found** – Unknown record.
- 500 **Internal Server Error** – Unexpected error. Object of type *Error*.

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

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. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Status Codes

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

Example request:


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

"OK"
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.identify

Parameters

- **galleryId** (*string*) – the id of the gallery. **The special value ‘ALL’ is used when the search is done against all galleries.** Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **maxNbCand** (*integer*) – the maximum number of candidates. Object of type integer.
- **threshold** (*number*) – the algorithm threshold. Object of type number.
- **accuracyLevel** (*string*) – the accuracy level expected for this request. Object of type string.

JSON Parameters

- **filter** (*object*) –
- **biometricData** (*array*) –

Status Codes

- **200 OK** – Request executed. Identification result is returned. Array of *Candidate*.

- 202 Accepted – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Identification not allowed.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/identify/{galleryId}?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&
→priority=1&maxNbCand=1&threshold=1.0&accuracyLevel=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "filter": {
    "dateOfBirthMin": "1980-01-01",
    "dateOfBirthMax": "2019-12-31",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}
```

Example response:

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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 3500,
    "scores": [
      {
        "score": 3500,
        "encounterId": "string",
        "biometricType": "FINGER",

```

(continues on next page)

(continued from previous page)

```

        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
            "...": "..."
        }
    },
    "others": {
        "...": "..."
    }
}
]

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Result of the identification (list of candidate). Array of *Candidate*.

Status Codes

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

Example request:

```

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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 3500,
    "scores": [
      {
        "score": 3500,
        "encounterId": "string",
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
          "...": "..."
        }
      }
    ],
    "others": {
      "...": "..."
    }
  }
]

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.identify

Parameters

- **galleryId** (*string*) – the id of the gallery. **The special value ‘ALL’ is used when the search is done against all galleries.** Object of type string.
- **personId** (*string*) – the id of the person. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.

- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.
- **maxNbCand** (*integer*) – the maximum number of candidates. Object of type *integer*.
- **threshold** (*number*) – the algorithm threshold. Object of type *number*.
- **accuracyLevel** (*string*) – the accuracy level expected for this request. Object of type *string*.

Form Parameters

- **body** – Object of type *Filter*.

Status Codes

- **200 OK** – Request executed. Identification result is returned. Array of *Candidate*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Identification not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/identify/{galleryId}/{personId}?transactionId=string&callback=http%3A%2F%2Fclient.com
→%2Fcallback&priority=1&maxNbCand=1&threshold=1.0&accuracyLevel=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": 3500,
    "scores": [
      {
        "score": 3500,
        "encounterId": "string",
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
          "...": "..."
        }
      }
    ],
    "others": {
      "...": "..."
    }
  }
]
```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – Result of the identification (list of candidate). Array of *Candidate*.

Status Codes

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

Example request:

```

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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 3500,
    "scores": [
      {
        "score": 3500,

```

(continues on next page)

(continued from previous page)

```

        "encounterId": "string",
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
            "...": "..."
        }
    },
    "others": {
        "...": "..."
    }
}
]

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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}/encounters/{encounterId}

Biometric identification based on an existing encounter

Identification based on an existing encounter from one gallery

Scope required: abis.identify

Parameters

- **galleryId** (*string*) – the id of the gallery. **The special value ‘ALL’ is used when the search is done against all galleries.** Object of type string.
- **personId** (*string*) – the id of the person. Object of type string.
- **encounterId** (*string*) – the id of the encounter. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **maxNbCand** (*integer*) – the maximum number of candidates. Object of type integer.
- **threshold** (*number*) – the algorithm threshold. Object of type number.
- **accuracyLevel** (*string*) – the accuracy level expected for this request. Object of type string.

Form Parameters

- **body** – Object of type *Filter*.

Status Codes

- **200 OK** – Request executed. Identification result is returned. Array of *Candidate*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Identification not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/identify/{galleryId}/{personId}/encounters/{encounterId}?transactionId=string&callback=http
→%3A%2F%2Fclient.com%2Fcallback&priority=1&maxNbCand=1&threshold=1.0&accuracyLevel=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": 3500,
    "scores": [
      {
        "score": 3500,
        "encounterId": "string",
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
          "...": "..."
        }
      }
    ],
    "others": {
      "...": "..."
    }
  }
]
```

Example response:

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:


```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Form Parameters

- **body** – Result of the identification (list of candidate). Array of *Candidate*.

Status Codes

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

Example request:

```

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

[
  {
    "personId": "string",
    "rank": 1,
    "score": 3500,
    "scores": [
      {
        "score": 3500,
        "encounterId": "string",
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "others": {
          "...": "..."
        }
      }
    ],
    "others": {
      "...": "..."
    }
  }
]

```

(continues on next page)

(continued from previous page)

```
}
]
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.verify**Parameters**

- **galleryId** (*string*) – the id of the gallery. **The special value ‘ALL’ is used when the verification is done against all galleries.** Object of type string.
- **personId** (*string*) – the id of the person. Object of type string.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **threshold** (*number*) – the algorithm threshold. Object of type number.
- **accuracyLevel** (*string*) – the accuracy level expected for this request. Object of type string.

JSON Parameters

- **biometricData** (*array*) –

Status Codes

- **200 OK** – Verification execution successful. Object of type boolean.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **404 Not Found** – Unknown record.
- **403 Forbidden** – Verification not allowed.

- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/verify/{galleryId}/{personId}?transactionId=string&callback=http%3A%2F%2Fclient.com
->%2Fcallback&priority=1&threshold=1.0&accuracyLevel=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}
```

Example response:

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

{
  "decision": true,
  "scores": [
    {
      "score": 3500,
      "encounterId": "string",
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "others": {
        "...": "..."
      }
    }
  ]
}
```

Example response:

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

(continues on next page)

(continued from previous page)

```
{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
```

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

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. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

JSON Parameters

- **decision** (*boolean*) –
- **scores** (*array*) –

Status Codes

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

Example request:

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

```
{
  "decision": true,
  "scores": [
    {
      "score": 3500,
      "encounterId": "string",
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "instance": "string",
        "others": {
            "...": "..."
        }
    }
]
}

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.verify**Query Parameters**

- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type string/uri.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type integer.
- **threshold** (*number*) – the algorithm threshold. Object of type number.
- **accuracyLevel** (*string*) – the accuracy level expected for this request. Object of type string.

JSON Parameters

- **biometricData1** (*array*) –
- **biometricData2** (*array*) –

Status Codes

- **200 OK** – Verification execution successful. Object of type boolean.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Verification not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/verify?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1&
↳threshold=1.0&accuracyLevel=string HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "biometricData1": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ],
  "biometricData2": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    }
  ]
}

```

Example response:

```

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

{
  "decision": true,
  "scores": [
    {
      "score": 3500,
      "encounterId": "string",
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "others": {
        "...": "..."
      }
    }
  ]
}

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

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 with two sets of data response callback****Query Parameters**

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

- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

JSON Parameters

- **decision** (*boolean*) –
- **scores** (*array*) –

Status Codes

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

Example request:

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

{
  "decision": true,
  "scores": [
    {
      "score": 3500,
      "encounterId": "string",
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "others": {
        "...": "..."
      }
    }
  ]
}
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.gallery.read

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.

Status Codes

- **200 OK** – Operation successful. Array of *string*.
- **202 Accepted** – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Read not allowed.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/galleries?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&priority=1 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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

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. Object of type string. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type string. (Required)

Form Parameters

- **body** – List of gallery IDs. Array of string.

Status Codes

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

Example request:

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

[
  "string"
]
```

Example request:

```
POST ${request.query.callback}?transactionId=string&taskId=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

Scope required: abis.gallery.read

Parameters

- **galleryId** (*string*) – the id of the gallery. Object of type *string*.

Query Parameters

- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)
- **callback** (*string*) – the callback address, where the result will be sent when available. Object of type *string/uri*.
- **priority** (*integer*) – the request priority (0: lowest priority; 9: highest priority). Object of type *integer*.
- **offset** (*integer*) – The offset of the query (first item of the response). Object of type *integer*. Default: 0.
- **limit** (*integer*) – The maximum number of items to return. Object of type *integer*. Default: 1000.

Status Codes

- 200 OK – Operation successful. Array of *PersonIds*.
- 202 Accepted – Request received successfully and correct, result will be returned through the callback. An internal task ID is returned. . Object of type *TaskId*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/galleries/{galleryId}?transactionId=string&callback=http%3A%2F%2Fclient.com%2Fcallback&
→priority=1&offset=1&limit=1 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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```

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

```

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. Object of type *string*. (Required)
- **taskId** (*string*) – The id of the task, used to match this response with the request. Object of type *string*. (Required)

Form Parameters

- **body** – List of encounters IDs. Array of *PersonIds*.

Status Codes

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

Example request:

```

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

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

```

Example request:

```

POST ${request.query.callback}?transactionId=string&taskId=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"
}
```

Data Model

Error

Table 7.29: Error

Attribute	Type	Description	Required
code	integer/int32	Error code.	Yes
message	string	Error message.	Yes

Encounter

Table 7.30: Encounter

Attribute	Type	Description	Required
encounterId	string	Constraints: read only	Yes
status	string	Constraints: possible values are ACTIVE, INACTIVE	Yes
encounterType	string	Type of the encounter.	Yes
galleries	Array of string	The list of galleries for this object. Constraints: minItems is 1; items must be unique	
clientData	string/byte		
contextualData	Object of type <i>ContextualData</i>		
biographicData	Object of type <i>BiographicData</i>	The set of biographic data.	
biometricData	Array of <i>BiometricData</i>		Yes

ContextualData

Table 7.31: ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "enrollmentDate": "2019-01-11"
}
```

BiographicData

The set of biographic data.

Table 7.32: BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA"
}
```

BiometricData

Table 7.33: BiometricData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	

continues on next page

Table 7.33 – continued from previous page

Attribute	Type	Description	Required
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
encounterId	string	the id of the encounter owner of this biometric. Constraints: read only	
image	string/byte	Base64-encoded image.	
imageRef	string/uri	URI to an image.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the biometric.	
impressionType	string	Constraints: possible values are LIVE_SCAN_PLAIN, LIVE_SCAN_ROLLED, NONLIVE_SCAN_PLAIN, NONLIVE_SCAN_ROLLED, LATENT_IMPRESSION, LATENT_TRACING, LATENT_PHOTO, LATENT_LIFT, LIVE_SCAN_SWIPE, LIVE_SCAN_VERTICAL_ROLL, LIVE_SCAN_PALM, NONLIVE_SCAN_PALM, LATENT_PALM_IMPRESSION, LATENT_PALM_TRACING, LATENT_PALM_PHOTO, LATENT_PALM_LIFT, LIVE_SCAN_OPTICAL_CONTACTLESS_PLAIN, OTHER, UNKNOWN	
width	integer	the width of the image.	
height	integer	the height of the image.	
bitdepth	integer		
contentType	string	the nature and format of the image. The mime type definitions should be in compliance with RFC 6838.	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of <i>MissingType</i>	Optional properties indicating if a part of the biometric data is missing.	
metadata	string	An optional string used to convey information vendor-specific.	
comment	string	A comment about the biometric data.	
template	string/byte	Base64-encoded template.	
templateRef	string/uri	URI to the template when it is managed in a dedicated data server.	
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table 7.34: MissingType

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

BiometricComputedData

Table 7.35: BiometricComputedData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes

continues on next page

Table 7.35 – continued from previous page

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
template	string/byte	Base64-encoded template.	Yes
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

Filter

Table 7.36: Filter

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

Candidate

Identification of a candidate result of a biometric search.

This structure can be extended by vendors able to include additional information to the three mandatory properties.

Table 7.37: Candidate

Attribute	Type	Description	Required
personId	string	the identifier of the person.	Yes
rank	integer/int32	the rank of the candidate in relation to other candidates for the same biometric identification operation.	Yes
score	number/float	the score of the candidate in relation to other candidates for the same biometric identification operation.	Yes
scores	Array of <i>ScoreDetail</i>	a list of comparison score(s) and optionally the type and subtype of the relating biometric.	
others.*		Additional properties	

ScoreDetail

Scoring information calculated after a biometric search. It includes at least the score (a float) and optionally the encounterId, type and subtype of the matching biometric item. It can also be extended with proprietary information to better describe the matching result (for instance: rotation needed to align the probe and the candidate)

Table 7.38: ScoreDetail

Attribute	Type	Description	Required
score	number/float	the score.	Yes
encounterId	string		
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	

continues on next page

Table 7.38 – continued from previous page

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
others.*		Additional properties	

PersonIds

Table 7.39: PersonIds

Attribute	Type	Description	Required
personId	string		Yes
encounterId	string		Yes

ExtendablePersonIds

The IDs of a record (personId and encounterId) extendable with additional properties if needed by an implementation.

This is used for the response of insert & update operations, when additional properties (such as: quality evaluation, proof of record, etc.) might be returned by the server.

Table 7.40: ExtendablePersonIds

Attribute	Type	Description	Required
personId	string		Yes
encounterId	string		Yes
others.*		Additional properties	

TaskId

Information about the asynchronous result. Only the taskId is mandatory but the implementation is free to return additional details such as: expected duration, URL to monitor the task, etc.

Table 7.41: TaskId

Attribute	Type	Description	Required
taskId	string		Yes
others.*		Additional properties	

7.3.7 Credential Services

This is version 1.2.0 of this interface.

Get the OpenAPI file: [cms.yaml](#)

Credential Services

aggregatedright

- *createCredentialRequest*
- *readCredentialRequest*
- *updateCredentialRequest*
- *cancelCredentialRequest*
- *findCredentials*
- *readCredential*
- *suspendCredential*
- *unsuspendCredential*
- *revokeCredential*
- *setCredentialStatus*
- *findCredentialProfiles*

Services

Credential Request

POST /v1/credentialRequests/{credentialRequestId}

Create a request for a credential

Scope required: cms.request.write

Parameters

- **credentialRequestId** (*string*) – the id of the credential request. Object of type string.

Query Parameters

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

Form Parameters

- **body** – Object of type *CredentialRequest*.

Status Codes

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

Example request:

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

```
{
  "status": "PENDING",
  "requestData": {
    "priority": 1,
    "credentialProfileId": "ABC",
    "requestType": "FIRST_ISSUANCE",
    "validFromDate": "2020-10-08T18:38:56Z",
    "validToDate": "2025-10-08T18:38:56Z",
    "issuingAuthority": "OSIA",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    }
  },
  "personId": "string",
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ]
    },
    {
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
      }
    ]
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

GET /v1/credentialRequests/{credentialRequestId}**Read a credential request**Scope required: `cms.request.read`**Parameters**

- **credentialRequestId** (*string*) – the id of the credential request. Object of type *string*.

Query Parameters

- **attributes** (*array*) – The (optional) set of attributes to retrieve. Array of *string*.
- **transactionId** (*string*) – The id of the transaction. Object of type *string*. (Required)

Status Codes

- 200 OK – Read successful. Object of type *CredentialRequest*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

GET /v1/credentialRequests/{credentialRequestId}?attributes=%5B%27string%27%5D&transactionId=string_
↪ HTTP/1.1
Host: example.com

```

Example response:

```

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

```

(continues on next page)

(continued from previous page)

```

{
  "credentialRequestId": "string",
  "status": "PENDING",
  "requestData": {
    "priority": 1,
    "credentialProfileId": "ABC",
    "requestType": "FIRST_ISSUANCE",
    "validFromDate": "2020-10-08T18:38:56Z",
    "validToDate": "2025-10-08T18:38:56Z",
    "issuingAuthority": "OSIA",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    }
  },
  "personId": "string",
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ],
  "credentialIds": [
    "string"
  ]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

```

(continues on next page)

(continued from previous page)

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

Example response:

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

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

PUT /v1/credentialRequests/{credentialRequestId}**Update a credential request**Scope required: `cms.request.write`**Parameters**

- **credentialRequestId** (*string*) – the id of the credential request. Object of type *string*.

Query Parameters

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

Form Parameters

- **body** – Object of type *CredentialRequest*.

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
PUT /v1/credentialRequests/{credentialRequestId}?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "status": "PENDING",
  "requestData": {
    "priority": 1,
    "credentialProfileId": "ABC",
    "requestType": "FIRST_ISSUANCE",
    "validFromDate": "2020-10-08T18:38:56Z",
    "validToDate": "2025-10-08T18:38:56Z",
    "issuingAuthority": "OSIA",
    "deliveryAddress": {
      "address1": "11 Rue des Rosiers",
      "city": "Libourne",
      "postalCode": "33500",
      "country": "France"
    }
  },
  "personId": "string",
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
  }
}
```

(continues on next page)

(continued from previous page)

```

    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ]
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

POST /v1/credentialRequests/{credentialRequestId}/cancel
Cancel a credential request

Scope required: cms.request.write

Parameters

- **credentialRequestId** (*string*) – the id of the credential request. Object of type string.

Query Parameters

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

Status Codes

- 204 No Content – Cancel successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Cancel not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

Credential**POST /v1/credentials**

Retrieve a list of credentials that match the given search criteria

Scope required: cms.credential.read

Query Parameters

- **attributes** (*array*) – The (optional) set of required attributes to retrieve. Array of string.
- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)

Form Parameters

- **body** – Array of *Expression*.

Status Codes

- 200 OK – Read successful. Array of *CredentialData*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
POST /v1/credentials?attributes=%5B%27string%27%5D&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
  {
    "attributeName": "string",
    "operator": "<",
    "value": "string"
  }
]
```

Example response:

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

[
  {
    "credentialId": "string",
    "status": "NEW",
    "statusOther": "string",
    "credentialNumber": "string",
    "personId": "string",
    "credentialProfileId": "string",
    "issuedDate": "2022-01-03T16:43:26.913783",
    "expiryDate": "2022-01-03T16:43:26.913783",
    "serialNumber": "string",
    "issuingAuthority": "string",
    "issuingPlace": "string",
    "others": {
      "...": "..."
    }
  }
]
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

GET /v1/credentials/{credentialId}**Read a credential****Scope required:** cms.credential.read**Parameters**

- **credentialId** (*string*) – the id of the credential. Object of type string.

Query Parameters

- **attributes** (*array*) – The (optional) set of required attributes to retrieve. Array of string.
- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)

Status Codes

- 200 OK – Read successful. Object of type *CredentialData*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```
GET /v1/credentials/{credentialId}?attributes=%5B%27string%27%5D&transactionId=string HTTP/1.1
Host: example.com
```

Example response:

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

{
  "credentialId": "string",
  "status": "NEW",
  "statusOther": "string",
  "credentialNumber": "string",
  "personId": "string",
  "credentialProfileId": "string",
  "issuedDate": "2022-01-03T16:43:26.913783",
  "expiryDate": "2022-01-03T16:43:26.913783",
  "serialNumber": "string",
  "issuingAuthority": "string",
  "issuingPlace": "string",
  "others": {
    "...": "..."
  }
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

POST /v1/credentials/{credentialId}/suspend

Suspend a credential

Scope required: cms.credential.write

Parameters

- **credentialId** (*string*) – the id of the credential. Object of type string.

Query Parameters

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

JSON Parameters

- **reason** (*string*) – the reason for suspension

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

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

{
  "reason": "string"
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

POST /v1/credentials/{credentialId}/unsuspend**Unsuspend a credential**

Scope required: cms.credential.write

Parameters

- **credentialId** (*string*) – the id of the credential. Object of type string.

Query Parameters

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

JSON Parameters

- **reason** (*string*) – the reason for unsuspension

Status Codes

- 204 No Content – Update successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Update not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

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

{
  "reason": "string"
}
```

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

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

Example response:

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

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

POST /v1/credentials/{credentialId}/revoke**Revoke a credential**Scope required: `cms.credential.write`**Parameters**

- **credentialId** (*string*) – the id of the credential. Object of type `string`.

Query Parameters

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

JSON Parameters

- **reason** (*string*) – the reason for revocation

Status Codes

- **204 No Content** – Update successful.
- **400 Bad Request** – Bad request. Object of type *Error*.
- **403 Forbidden** – Update not allowed.
- **404 Not Found** – Unknown record.
- **500 Internal Server Error** – Unexpected error. Object of type *Error*.

Example request:

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

{
  "reason": "string"
}
```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

POST /v1/credentials/{credentialId}/status
Change the status of a credential

Scope required: cms.credential.write

Parameters

- **credentialId** (*string*) – the id of the credential. Object of type string.

Query Parameters

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

JSON Parameters

- **status** (*string*) – The new status of the credential
- **reason** (*string*) – The reason for the change of status
- **requester** (*string*) – The ID/name of the entity requesting the change
- **comment** (*string*) – A free comment

Status Codes

- 204 No Content – Operation successful.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

POST /v1/credentials/{credentialId}/status?transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "status": "string",
  "reason": "string",
  "requester": "string",
  "comment": "string"
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

{

```

(continues on next page)

(continued from previous page)

```

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

```

Example response:

```

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

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

```

Credential Profile**POST /v1/credentialProfiles**

Retrieve a list of credential profiles that match the given search criteria

Scope required: `cms.profile.read`**Query Parameters**

- **attributes** (*array*) – The (optional) set of required attributes to retrieve. Array of string.
- **transactionId** (*string*) – The id of the transaction. Object of type string. (Required)

Form Parameters

- **body** – Array of *Expression*.

Status Codes

- 200 OK – Read successful. Array of *CredentialProfile*.
- 400 Bad Request – Bad request. Object of type *Error*.
- 403 Forbidden – Read not allowed.
- 404 Not Found – Unknown record.
- 500 Internal Server Error – Unexpected error. Object of type *Error*.

Example request:

```

POST /v1/credentialProfiles?attributes=%5B%27string%27%5D&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

[
  {
    "attributeName": "string",
    "operator": "<",
    "value": "string"
  }
]

```

Example response:

```

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

[
  {
    "credentialProfileId": "string",
    "name": "string",
    "description": "string",

```

(continues on next page)

(continued from previous page)

```

    "credentialType": "SMARTCARD",
    "defaultLifetime": 1
  }
]

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

Data Model**Error**

Table 7.42: Error

Attribute	Type	Description	Required
code	integer/int32	Error code.	Yes
message	string	Error message.	Yes

Address

Table 7.43: Address

Attribute	Type	Description	Required
address1	string	the first line of the address.	
address2	string	the second line of the address.	
city	string	the city of the address.	
state	string	the state of the address.	
postalCode	string	the postal code of the address.	
country	string	the country of the address.	
others.*		Additional properties	

Example #1:

```

{
  "address1": "11 Rue des Rosiers",
  "address2": "1st floor",
  "city": "Libourne",
  "state": "Gironde",
  "postalCode": "33500",
  "country": "France"
}

```

BiographicData

The set of biographic data.

Table 7.44: BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "firstName": "John",
  "lastName": "Doo",
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA"
}
```

BiometricData

Table 7.45: BiometricData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes

continues on next page

Table 7.45 – continued from previous page

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
encounterId	string	the id of the encounter owner of this biometric.	
image	string/byte	Base64-encoded image.	
imageRef	string/uri	URI to an image.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the biometric.	

continues on next page

Table 7.45 – continued from previous page

Attribute	Type	Description	Required
impressionType	string	Constraints: possible values are LIVE_SCAN_PLAIN, LIVE_SCAN_ROLLED, NONLIVE_SCAN_PLAIN, NONLIVE_SCAN_ROLLED, LATENT_IMPRESSION, LATENT_TRACING, LATENT_PHOTO, LATENT_LIFT, LIVE_SCAN_SWIPE, LIVE_SCAN_VERTICAL_ROLL, LIVE_SCAN_PALM, NONLIVE_SCAN_PALM, LATENT_PALM_IMPRESSION, LATENT_PALM_TRACING, LATENT_PALM_PHOTO, LATENT_PALM_LIFT, LIVE_SCAN_OPTICAL_CONTACTLESS_PLAIN, OTHER, UNKNOWN	
width	integer	the width of the image.	
height	integer	the height of the image.	
bitdepth	integer		
imeType	string	the nature and format of the image. The mime type definitions should be in compliance with RFC 6838.	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of <i>MissingType</i>	Optional properties indicating if a part of the biometric data is missing.	
metadata	string	An optional string used to convey information vendor-specific.	
comment	string	A comment about the biometric data.	
template	string/byte	Base64-encoded template.	
templateRef	string/uri	URI to the template when it is managed in a dedicated data server.	
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table 7.46: MissingType

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

RequestData

The data describing the request itself

Table 7.47: RequestData

Attribute	Type	Description	Required
priority	integer	the request priority (0: lowest priority; 9: highest priority).	Yes
credentialProfileId	string	The id of the credential profile to request.	Yes
requestType	string	The type of request, e.g. first issuance, renewal, etc. Constraints: possible values are FIRST_ISSUANCE, RENEWAL, REPLACEMENT, OTHER	Yes
requestTypeOther	string	Details about the request type when OTHER is selected.	

continues on next page

Table 7.47 – continued from previous page

Attribute	Type	Description	Required
validFromDate	string/date-time	May be used to override the default start date of the requested credential. This must only be later than the current date, not earlier.	
validToDate	string/date-time	May be used to override the default expiry date of the requested credential. This must only be earlier than the default expiry, not later.	
credentialNumber	string	Number to be used for the new credentials created. It can be used for example when requesting a digital credential sharing the same number with a physical credential, or when the number is not created by the issuance system.	
issuingAuthority	string		
deliveryOffice	string	Single code or name identifying the office where the credential has to be delivered.	
deliveryAddress	Object of type <i>Address</i>		
parentCredentialId	string	The ID credential used as a reference, or parent, to build a new one.	
others.*		Additional properties	

Example #1:

```
{
  "priority": 1,
  "credentialProfileId": "ABC",
  "requestType": "FIRST_ISSUANCE",
  "validFromDate": "2020-10-08T18:38:56Z",
  "validToDate": "2025-10-08T18:38:56Z",
  "issuingAuthority": "OSIA",
  "deliveryAddress": {
    "address1": "11 Rue des Rosiers",
    "city": "Libourne",
    "postalCode": "33500",
    "country": "France"
  }
}
```

CredentialRequest

A request for a credential

Table 7.48: CredentialRequest

Attribute	Type	Description	Required
credentialRequestId	string	The unique id of this credential request. Constraints: read only	
status	string	Constraints: possible values are PENDING, ISSUED, CANCELLED, FAILED	
requestData	Object of type <i>RequestData</i>	The data describing the request itself.	Yes
personId	string	The id of the person who is the target of the request.	Yes
biographicData	Object of type <i>BiographicData</i>	The set of biographic data.	Yes

continues on next page

Table 7.48 – continued from previous page

Attribute	Type	Description	Required
biometricData	Array of <i>Biometric-Data</i>		
credentialIds	Array of string	The id of the credentials created for this request. The unique id of the credential. Constraints: read only	

CredentialData

A credential

Table 7.49: CredentialData

Attribute	Type	Description	Required
credentialId	string	The unique id for this credential. Constraints: read only	Yes
status	string	The status of the credential. Constraints: possible values are NEW, ACTIVE, SUSPENDED, REVOKED, OTHER; read only	Yes
statusOther	string	Details about the status when OTHER is used. Constraints: read only	
credentialNumber	string	The number attached to the credential (ex: passport number).	
personId	string	The unique id of the person that the credential request is for.	Yes
credentialProfileId	string	The unique id of the credential profile.	Yes
issuedDate	string/date-time	The date and time that this credential was issued.	
expiryDate	string/date-time	The date and time that this credential expires.	
serialNumber	string	the serial number of the credential.	
issuingAuthority	string	The authority issuing the credential (ex: the Ministry of Interior).	
issuingPlace	string	The place where the credential was issued (ex: Paris).	
others.*		Additional properties	

CredentialProfile

A credential profile

Table 7.50: CredentialProfile

Attribute	Type	Description	Required
credentialProfileId	string	The unique id for this credential profile.	
name	string	The name of the credential profile.	
description	string	The description of the credential profile.	
credentialType	string	The type of credential that this profile will issue. Constraints: possible values are SMARTCARD, VIRTUAL_SMARTCARD, MOBILE, PASSPORT, ID_CARD	
defaultLifetime	integer	The default number of days that this credential will be considered valid for after issuance.	

Expression

Table 7.51: Expression

Attribute	Type	Description	Required
attributeName	string		Yes
operator	string	Constraints: possible values are <, >, =, >=, <=, !=	Yes
value	One of string, integer, number, boolean		Yes

Expressions

Table 7.52: Expressions

Attribute	Type	Description	Required
N/A	Array of <i>Expression</i>		

7.3.8 ID Usage Services

Relying Party Services

This is version 1.1.0 of this interface.

Get the OpenAPI file: [rp.yaml](#)

Relying Party Services

aggregatedright

- *verify*
- *readAttributeSet*

- *readAttributes*
- *identify*
- *readIdentifyResult*

Services

IDUsage

POST /v1/verify/{identifier}

Verify a set of attributes of a person.

Verify an Identity based on an identity identifier (UIN, token...) and a set of Identity Attributes. Verification is strictly matching all provided identity attributes to compute the global Boolean matching result.

Scope required: id.verify

Parameters

- **identifier** (*string*) – person identifier. Object of type string.

Query Parameters

- **identifierType** (*string*) – Type of identifier (default “uin”, “token”, “credential-Number”, ...). Object of type string.
- **verificationProofRequired** (*boolean*) – verification proof required on successful verification (default true). Object of type boolean.

- **transactionId** (*string*) – The client specified id of the transaction. Object of type *string*. (Required)

Form Parameters

- **body** – A set of identity attributes associated to the identity identifier and to be verified by the system. Object of type *AttributeSet*.

Status Codes

- 200 OK – Verification execution successful. Object of type *VerifyResult*.
- 400 Bad Request – Bad Request, Validation Errors, ... Object of type *Error*.
- 401 Unauthorized – Unauthorized.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Identifier not Found.
- 500 Internal Server Error – Internal server error. Object of type *Error*.

Example request:

```
POST /v1/verify/{identifier}?identifierType=token&verificationProofRequired=True&
↳transactionId=string HTTP/1.1
```

```
Host: example.com
```

```
Content-Type: application/json
```

```
{
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ],
      "metadata": "string",
      "comment": "string",
      "template": "c3RyaW5n",
      "templateRef": "http://dataserver.com/template?id=00014",
      "templateFormat": "string",
      "quality": 1,
      "qualityFormat": "string",
      "algorithm": "string",
      "vendor": "string"
    }
  ],
  "credentialData": [
    {
```

(continues on next page)

(continued from previous page)

```

        "credentialNumber": "string",
        "personId": "string",
        "credentialType": "ID_CARD",
        "issuedDate": "2022-01-03T16:43:26.913783",
        "expiryDate": "2022-01-03T16:43:26.913783",
        "serialNumber": "string",
        "issuingAuthority": "string",
        "issuingPlace": "string",
        "others": {
            "...": "..."
        }
    },
    "contactData": {
        "email": "John.Doo@osia.com",
        "phone1": "555666777",
        "phone2": "555888999",
        "...": "..."
    }
}

```

Example response:

```

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

{
  "verificationCode": 1,
  "verificationMessage": "string",
  "verificationProof": "string"
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

GET /v1/attributes/{attributeSetName}/{identifier}**Read a predefined set of a person's attributes.**

Note security role must map the requested attributeSetName, e.g. id.DEFAULT_SET_01.read

Scope required: id.ATTRIBUTESETNAME.read**Parameters**

- **attributeSetName** (*string*) – Predefined attribute set name describing what attributes are to be read. e.g. "DEFAULT_SET_01", "SET_BIOM_01", "EIDAS", ... Object of type string.
- **identifier** (*string*) – person identifier. Object of type string.

Query Parameters

- **identifierType** (*string*) – Type of identifier (default “uin”, “token”, “credential-Number”, ...). Object of type *string*.
- **transactionId** (*string*) – The client specified id of the transaction. Object of type *string*. (Required)

Status Codes

- 200 OK – Operation successful, AttributeSet will contain fields as predefined by the attributeSetName and when value is available. Object of type *AttributeSet*.
- 400 Bad Request – Bad Request, Validation Errors, ... Object of type *Error*.
- 401 Unauthorized – Unauthorized.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Not Found.
- 500 Internal Server Error – Internal server error. Object of type *Error*.

Example request:

```
GET /v1/attributes/{attributeSetName}/{identifier}?identifierType=token&transactionId=string HTTP/1.1
Host: example.com
```

Example response:

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

{
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
      "width": 1,
      "height": 1,
      "bitdepth": 1,
      "mimeType": "string",
      "resolution": 1,
      "compression": "WSQ",
      "missing": [
        {
          "biometricSubType": "RIGHT_INDEX",
          "presence": "BANDAGED"
        }
      ]
    },
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  ]
}
```

(continues on next page)

(continued from previous page)

```

    }
  ],
  "credentialData": [
    {
      "credentialId": "string",
      "status": "NEW",
      "statusOther": "string",
      "credentialNumber": "string",
      "personId": "string",
      "credentialType": "ID_CARD",
      "issuedDate": "2022-01-03T16:43:26.913783",
      "expiryDate": "2022-01-03T16:43:26.913783",
      "serialNumber": "string",
      "issuingAuthority": "string",
      "issuingPlace": "string",
      "others": {
        "...": "..."
      }
    }
  ],
  "contactData": {
    "email": "John.Doo@osia.com",
    "phone1": "555666777",
    "phone2": "555888999",
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/json

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

```

Example response:

```

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

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

```

POST /v1/attributes/{identifier}**Read a variable set of a person's attributes.**

Returns value of attributes listed in the request parameter 'OutputAttributeSet'

Scope required: id.read**Parameters**

- **identifier** (*string*) – person identifier. Object of type string.

Query Parameters

- **identifierType** (*string*) – Type of identifier (default "uin", "token", "credential-Number", ...). Object of type string.
- **transactionId** (*string*) – The client specified id of the transaction. Object of type string. (Required)

Form Parameters

- **body** – A description of expected identity attributes . Object of type *OutputAttributeSet*.

Status Codes

- **200 OK** – Operation successful, AttributeSet will contain fields as defined by parameter outputAttributeSet and when value is available. Object of type *AttributeSet*.
- **400 Bad Request** – Bad Request, Validation Errors, ... Object of type *Error*.
- **401 Unauthorized** – Unauthorized.
- **403 Forbidden** – Operation not allowed.
- **404 Not Found** – Not Found.
- **500 Internal Server Error** – Internal server error. Object of type *Error*.

Example request:

```
POST /v1/attributes/{identifier}?identifierType=token&transactionId=string HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "outputBiographicData": [
    "string"
  ],
  "outputBiometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "biometricDataFields": [
        "string"
      ]
    }
  ],
  "outputCredentialData": [
    {
      "credentialType": "ID_CARD",
      "credentialDataFields": [
        "string"
      ]
    }
  ],
  "outputContactData": [
    "string"
  ]
}
```

Example response:

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

{
  "biographicData": {
    "firstName": "John",
    "lastName": "Doo",
    "dateOfBirth": "1985-11-30",
    "gender": "M",
    "nationality": "FRA",
    "...": "..."
  },
  "biometricData": [
    {
      "biometricType": "FINGER",
      "biometricSubType": "RIGHT_INDEX",
      "instance": "string",
      "encounterId": "string",
      "image": "c3RyaW5n",
      "imageRef": "http://imageserver.com/image?id=00003",
      "captureDate": "2019-05-21T12:00:00Z",
      "captureDevice": "string",
      "impressionType": "LIVE_SCAN_PLAIN",
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",
        "resolution": 1,
        "compression": "WSQ",
        "missing": [
            {
                "biometricSubType": "RIGHT_INDEX",
                "presence": "BANDAGED"
            }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
],
"credentialData": [
    {
        "credentialId": "string",
        "status": "NEW",
        "statusOther": "string",
        "credentialNumber": "string",
        "personId": "string",
        "credentialType": "ID_CARD",
        "issuedDate": "2022-01-03T16:43:26.913783",
        "expiryDate": "2022-01-03T16:43:26.913783",
        "serialNumber": "string",
        "issuingAuthority": "string",
        "issuingPlace": "string",
        "others": {
            "...": "..."
        }
    }
],
"contactData": {
    "email": "John.Doo@osia.com",
    "phone1": "555666777",
    "phone2": "555888999",
    "...": "..."
}
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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

Identify a set of persons matching provided partial attributes

Identify possibly matching identities against an input set of attributes. Returns an array of predefined datasets as described by outputDataSetName. Note this request may be asynchronous or synchronous.

Scope required: id.identify

Query Parameters

- **transactionId** (*string*) – The client specified id of the transaction. Object of type *string*. (Required)

Form Parameters

- **body** – A set of identity attributes to match and an attributeSetName to use as template for returned matching identities. Object of type *IdentifyRequest*.

Status Codes

- 200 OK – Identification request execution successful. Array of *AttributeSet*.
- 202 Accepted – Request received successfully and correct, result will be available later using the task ID returned. Object of type *TaskId*.
- 400 Bad Request – Bad Request, Validation Errors, ... Object of type *Error*.
- 401 Unauthorized – Unauthorized.
- 403 Forbidden – Operation not allowed.
- 404 Not Found – Identifier not Found.
- 500 Internal Server Error – Internal server error. Object of type *Error*.

Example request:

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

{
  "attributeSet": {
    "biographicData": {
      "firstName": "John",
      "lastName": "Doo",
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA",
      "...": "..."
    },
    "biometricData": [
      {
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "image": "c3RyaW5n",
        "imageRef": "http://imageserver.com/image?id=00003",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",
        "resolution": 1,
        "compression": "WSQ",
        "missing": [
          {
            "biometricSubType": "RIGHT_INDEX",
            "presence": "BANDAGED"
          }
        ],
        "metadata": "string",

```

(continues on next page)

(continued from previous page)

```

        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
],
"credentialData": [
    {
        "credentialNumber": "string",
        "personId": "string",
        "credentialType": "ID_CARD",
        "issuedDate": "2022-01-03T16:43:26.913783",
        "expiryDate": "2022-01-03T16:43:26.913783",
        "serialNumber": "string",
        "issuingAuthority": "string",
        "issuingPlace": "string",
        "others": {
            "...": "..."
        }
    }
],
"contactData": {
    "email": "John.Doo@osia.com",
    "phone1": "555666777",
    "phone2": "555888999",
    "...": "..."
}
},
"outputAttributeSetName": "DEFAULT_SET_01"
}

```

Example response:

```

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

[
    {
        "biographicData": {
            "firstName": "John",
            "lastName": "Doo",
            "dateOfBirth": "1985-11-30",
            "gender": "M",
            "nationality": "FRA",
            "...": "..."
        },
        "biometricData": [
            {
                "biometricType": "FINGER",
                "biometricSubType": "RIGHT_INDEX",
                "instance": "string",
                "encounterId": "string",
                "image": "c3RyaW5n",
                "imageRef": "http://imageserver.com/image?id=00003",
                "captureDate": "2019-05-21T12:00:00Z",
                "captureDevice": "string",
                "impressionType": "LIVE_SCAN_PLAIN",
                "width": 1,
                "height": 1,
                "bitdepth": 1,
                "mimeType": "string",
                "resolution": 1,
                "compression": "WSQ",
                "missing": [
                    {
                        "biometricSubType": "RIGHT_INDEX",

```

(continues on next page)

(continued from previous page)

```

        "presence": "BANDAGED"
      }
    ],
    "metadata": "string",
    "comment": "string",
    "template": "c3RyaW5n",
    "templateRef": "http://dataserver.com/template?id=00014",
    "templateFormat": "string",
    "quality": 1,
    "qualityFormat": "string",
    "algorithm": "string",
    "vendor": "string"
  }
],
"credentialData": [
  {
    "credentialId": "string",
    "status": "NEW",
    "statusOther": "string",
    "credentialNumber": "string",
    "personId": "string",
    "credentialType": "ID_CARD",
    "issuedDate": "2022-01-03T16:43:26.913783",
    "expiryDate": "2022-01-03T16:43:26.913783",
    "serialNumber": "string",
    "issuingAuthority": "string",
    "issuingPlace": "string",
    "others": {
      "...": "..."
    }
  }
],
"contactData": {
  "email": "John.Doo@osia.com",
  "phone1": "555666777",
  "phone2": "555888999",
  "...": "..."
}
}
]

```

Example response:

```

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

{
  "taskId": "123e4567-e89b-12d3-a456-426655440000",
  "others": {
    "...": "..."
  }
}

```

Example response:

```

HTTP/1.1 400 Bad Request
Content-Type: application/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"
  }

```

GET /v1/identify/{taskID}**Read the result of a previously sent identify request****Scope required:** id.identify**Parameters**

- **taskID** (*string*) – taskID to get result for. Object of type *string*.

Query Parameters

- **transactionId** (*string*) – The client specified id of the transaction. Object of type *string*. (Required)

Status Codes

- **200 OK** – Operation successful, array of *AttributeSet* is available. Array of *AttributeSet*.
- **204 No Content** – No content, taskID is valid but identify request is still ongoing, retry later.
- **400 Bad Request** – Bad Request, Validation Errors, ... Object of type *Error*.
- **401 Unauthorized** – Unauthorized.
- **403 Forbidden** – Operation not allowed.
- **404 Not Found** – Not Found.
- **500 Internal Server Error** – Internal server error. Object of type *Error*.

Example request:

```

GET /v1/identify/{taskID}?transactionId=string HTTP/1.1
Host: example.com

```

Example response:

```

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

[
  {
    "biographicData": {
      "firstName": "John",
      "lastName": "Doo",
      "dateOfBirth": "1985-11-30",
      "gender": "M",
      "nationality": "FRA",
      "...": "..."
    },
    "biometricData": [
      {
        "biometricType": "FINGER",
        "biometricSubType": "RIGHT_INDEX",
        "instance": "string",
        "encounterId": "string",
        "image": "c3RyaW5n",
        "imageRef": "http://imageserver.com/image?id=00003",
        "captureDate": "2019-05-21T12:00:00Z",
        "captureDevice": "string",
        "impressionType": "LIVE_SCAN_PLAIN",
        "width": 1,
        "height": 1,
        "bitdepth": 1,
        "mimeType": "string",

```

(continues on next page)

(continued from previous page)

```

        "resolution": 1,
        "compression": "WSQ",
        "missing": [
            {
                "biometricSubType": "RIGHT_INDEX",
                "presence": "BANDAGED"
            }
        ],
        "metadata": "string",
        "comment": "string",
        "template": "c3RyaW5n",
        "templateRef": "http://dataserver.com/template?id=00014",
        "templateFormat": "string",
        "quality": 1,
        "qualityFormat": "string",
        "algorithm": "string",
        "vendor": "string"
    }
],
"credentialData": [
    {
        "credentialId": "string",
        "status": "NEW",
        "statusOther": "string",
        "credentialNumber": "string",
        "personId": "string",
        "credentialType": "ID_CARD",
        "issuedDate": "2022-01-03T16:43:26.913783",
        "expiryDate": "2022-01-03T16:43:26.913783",
        "serialNumber": "string",
        "issuingAuthority": "string",
        "issuingPlace": "string",
        "others": {
            "...": "..."
        }
    }
],
"contactData": {
    "email": "John.Doo@osia.com",
    "phone1": "555666777",
    "phone2": "555888999",
    "...": "..."
}
}
]

```

Example response:

HTTP/1.1 400 Bad Request
Content-Type: application/json

```

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

```

Example response:

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

```

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

```

Data Model

AttributeSet

a set of attributes used in verify

Table 7.53: AttributeSet

Attribute	Type	Description	Required
biographicData	Object of type <i>BiographicData</i>	The set of biographic data.	
biometricData	Array of <i>BiometricData</i>		
credentialData	Array of <i>CredentialData</i>		
contactData	Object of type <i>ContactData</i>		

BiographicData

The set of biographic data.

Table 7.54: BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "firstName": "John",
  "lastName": "Doo",
  "dateOfBirth": "1985-11-30",
  "gender": "M",
  "nationality": "FRA"
}
```

ContactData

Table 7.55: ContactData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "email": "John.Doo@osia.com",
  "phone1": "555666777",
  "phone2": "555888999"
}
```

BiometricData

Table 7.56: BiometricData

Attribute	Type	Description	Required
biometricType	string	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	Yes
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distincts biometric items of the same type and subtype.	
encounterId	string	the id of the encounter owner of this biometric. Constraints: read only	
image	string/byte	Base64-encoded image.	
imageRef	string/uri	URI to an image.	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the biometric.	

continues on next page

Table 7.56 – continued from previous page

Attribute	Type	Description	Required
impressionType	string	Constraints: possible values are LIVE_SCAN_PLAIN, LIVE_SCAN_ROLLED, NONLIVE_SCAN_PLAIN, NONLIVE_SCAN_ROLLED, LATENT_IMPRESSION, LATENT_TRACING, LATENT_PHOTO, LATENT_LIFT, LIVE_SCAN_SWIPE, LIVE_SCAN_VERTICAL_ROLL, LIVE_SCAN_PALM, NONLIVE_SCAN_PALM, LATENT_PALM_IMPRESSION, LATENT_PALM_TRACING, LATENT_PALM_PHOTO, LATENT_PALM_LIFT, LIVE_SCAN_OPTICAL_CONTACTLESS_PLAIN, OTHER, UNKNOWN	
width	integer	the width of the image.	
height	integer	the height of the image.	
bitdepth	integer		
mimeType	string	the nature and format of the image. The mime type definitions should be in compliance with RFC 6838.	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of <i>MissingType</i>	Optional properties indicating if a part of the biometric data is missing.	
metadata	string	An optional string used to convey information vendor-specific.	
comment	string	A comment about the biometric data.	
template	string/byte	Base64-encoded template.	
templateRef	string/uri	URI to the template when it is managed in a dedicated data server.	
templateFormat	string	Format of the template. One of ISO_19794_2, ISO_19794_2_NS, ISO_19794_2_CS, ISO_19794_2_2011, ANSI_378_2009 or ANSI_378. Can be extended to include additional proprietary template format.	
quality	integer/int64	Quality, as a number, of the biometric.	
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table 7.57: MissingType

Attribute	Type	Description	Required
biometricSubType	string	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

CredentialData

A credential

Table 7.58: CredentialData

Attribute	Type	Description	Required
credentialId	string	The unique id for this credential. Constraints: read only	
status	string	The status of the credential. Constraints: possible values are NEW, ACTIVE, SUSPENDED, REVOKED, OTHER; read only	
statusOther	string	Details about the status when OTHER is used. Constraints: read only	
credentialNumber	string	The number attached to the credential (ex: passport number).	

continues on next page

Table 7.58 – continued from previous page

Attribute	Type	Description	Required
personId	string	The unique id of the person that the credential request is for.	
credentialType	string	Type of the credential. e.g. “PASSPORT”, “ID_CARD”, ...	
issuedDate	string/date-time	The date and time that this credential was issued.	
expiryDate	string/date-time	The date and time that this credential expires.	
serialNumber	string	the serial number of the credential.	
issuingAuthority	string	The authority issuing the credential (ex: the Ministry of Interior).	
issuingPlace	string	The place where the credential was issued (ex: Paris).	
others.*		Additional properties	

Error

Table 7.59: Error

Attribute	Type	Description	Required
code	integer/int32	Error code.	Yes
message	string	Error message.	Yes

VerifyResult

result of a successful verify request

Table 7.60: VerifyResult

Attribute	Type	Description	Required
verificationCode	integer/int64		Yes
verificationMessage	string		Yes
verificationProof	string		

IdentifyRequest

a set of parameters used in identify

Table 7.61: IdentifyRequest

Attribute	Type	Description	Required
attributeSet	Object of type <i>AttributeSet</i>	a set of attributes used in verify.	Yes
outputAttributeSetName	string	Attribute set name describing what attributes are to be read. e.g. “DEFAULT_SET_01”, “SET_BIOM_01”, “EIDAS”, ...	Yes

OutputAttributeSet

A template describing the expected attributes of a readAttributes request

Table 7.62: OutputAttributeSet

Attribute	Type	Description	Required
outputBiographicData	Array of string	list of BiographicData structure fields to include in the answer.	
outputBiometricData	Array	an array of expected biometric data & fields.	
outputBiometricData biometricType	String	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	
outputBiometricData biometricSubType	String	Constraints: possible values are UNKNOWN, RIGHT_THUMB, RIGHT_INDEX, RIGHT_MIDDLE, RIGHT_RING, RIGHT_LITTLE, LEFT_THUMB, LEFT_INDEX, LEFT_MIDDLE, LEFT_RING, LEFT_LITTLE, PLAIN_RIGHT_FOUR_FINGERS, PLAIN_LEFT_FOUR_FINGERS, PLAIN_THUMBS, UNKNOWN_PALM, RIGHT_FULL_PALM, RIGHT_WRITERS_PALM, LEFT_FULL_PALM, LEFT_WRITERS_PALM, RIGHT_LOWER_PALM, RIGHT_UPPER_PALM, LEFT_LOWER_PALM, LEFT_UPPER_PALM, RIGHT_OTHER, LEFT_OTHER, RIGHT_INTERDIGITAL, RIGHT_THENAR, LEFT_INTERDIGITAL, LEFT_THENAR, LEFT_HYPOTHENAR, RIGHT_INDEX_AND_MIDDLE, RIGHT_MIDDLE_AND_RING, RIGHT_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE, LEFT_MIDDLE_AND_RING, LEFT_RING_AND_LITTLE, RIGHT_INDEX_AND_LEFT_INDEX, RIGHT_INDEX_AND_MIDDLE_AND_RING, RIGHT_MIDDLE_AND_RING_AND_LITTLE, LEFT_INDEX_AND_MIDDLE_AND_RING, LEFT_MIDDLE_AND_RING_AND_LITTLE, EYE_UNDEF, EYE_RIGHT, EYE_LEFT, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
outputBiometricData biometricDataFields	Array of string		
outputCredentialData	Array	an array of expected credential type & fields.	
outputCredentialData credentialType	String	Type of the credential. e.g. "PASSPORT", "ID_CARD", ...	
outputCredentialData credentialDataFields	Array of string		
outputContactData	Array of string	list of ContactData structure fields to include in the answer.	

TaskId

Information about the asynchronous result. Only the taskId is mandatory but the implementation is free to return additional details such as: expected duration, URL to monitor the task, etc.

Table 7.63: TaskId

Attribute	Type	Description	Required
taskId	string		Yes
others.*		Additional properties	

LIST OF TABLES

1.1	Revision History	4
2.1	Components	6
2.2	Interfaces List	8
2.3	Components vs Interfaces Mapping	10
3.1	Scopes List	22
4.1	OSIA Services Versions	29
5.1	Event Type	32
5.2	Person Attributes	36
5.3	Certificate Attributes	36
5.4	Union Attributes	36
5.5	Filiation Attributes	37
5.6	Document Type	37
5.7	Enrolment Data Model	41
5.8	Population Registry Data Model	48
5.9	Biometric Services Options	55
5.10	Biometric Data Model	56
5.11	Credential Data Model	60
6.1	Enrolment Data Model	68
6.2	Event Type	71
6.3	Person Attributes	75
6.4	Certificate Attributes	75
6.5	Union Attributes	75
6.6	Filiation Attributes	75
6.7	Document Type	75
6.8	Population Registry Data Model	81
6.9	Event Type	85
6.10	Person Attributes	89
6.11	Certificate Attributes	89
6.12	Union Attributes	89
6.13	Filiation Attributes	89
6.14	Document Type	89
6.15	Biometric Services Options	96
6.16	Biometric Data Model	97
6.17	Credential Data Model	101
7.1	Event Type & Message	113
7.2	Person Attributes	123
7.3	Matching Error Object	123
7.4	Expression Object	124

7.5	Error Object	124
7.6	Error	138
7.7	Enrollment	139
7.8	RequestData	139
7.9	EnrollmentFlags	139
7.10	ContextualData	140
7.11	BiographicData	140
7.12	BiometricData	140
7.13	MissingType	143
7.14	DocumentPart	143
7.15	DocumentData	144
7.16	Expression	144
7.17	Expressions	144
7.18	Error	167
7.19	Person	167
7.20	Identity	167
7.21	ContextualData	168
7.22	BiographicData	168
7.23	DocumentData	168
7.24	DocumentPart	169
7.25	BiometricData	169
7.26	MissingType	172
7.27	Expression	172
7.28	Expressions	173
7.29	Error	225
7.30	Encounter	225
7.31	ContextualData	225
7.32	BiographicData	226
7.33	BiometricData	226
7.34	MissingType	228
7.35	BiometricComputedData	228
7.36	Filter	230
7.37	Candidate	230
7.38	ScoreDetail	230
7.39	PersonIds	231
7.40	ExtendablePersonIds	232
7.41	TaskId	232
7.42	Error	245
7.43	Address	245
7.44	BiographicData	246
7.45	BiometricData	246
7.46	MissingType	249
7.47	RequestData	249
7.48	CredentialRequest	250
7.49	CredentialData	251
7.50	CredentialProfile	251
7.51	Expression	252
7.52	Expressions	252
7.53	AttributeSet	264
7.54	BiographicData	264
7.55	ContactData	264
7.56	BiometricData	265
7.57	MissingType	267
7.58	CredentialData	267
7.59	Error	268
7.60	VerifyResult	268
7.61	IdentifyRequest	268
7.62	OutputAttributeSet	269

7.63 TaskId 270

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	Death Use Case	13
2.4	Deduplication Use Case	14
2.5	ID Card Request Use Case (1)	15
2.6	ID Card Request Use Case (2)	16
2.7	Credentials Issuance Use Case	17
2.8	Bank account opening Use Case	18
2.9	Collaborative identity control	18
2.10	Telco Customer Enrollment with ID document	19
2.11	Telco Customer Enrollment with no ID document	20
5.1	Subscription & Notification Process	30
5.2	readPersonAttributes Sequence Diagram	33
5.3	matchPersonAttributes Sequence Diagram	34
5.4	verifyPersonAttributes Sequence Diagram	34
5.5	queryPersonUIN Sequence Diagram	35
5.6	queryPersonList Sequence Diagram	35
5.7	readDocument Sequence Diagram	36
5.8	generateUIN Sequence Diagram	37
5.9	Enrollment Data Model	42
5.10	Population Registry Data Model	49
5.11	Biometric Data Model	57
5.12	Credential Data Model	61
6.1	Enrollment Data Model	68
6.2	Subscription & Notification Process	69
6.3	readPersonAttributes Sequence Diagram	72
6.4	matchPersonAttributes Sequence Diagram	72
6.5	verifyPersonAttributes Sequence Diagram	73
6.6	queryPersonUIN Sequence Diagram	73
6.7	queryPersonList Sequence Diagram	74
6.8	readDocument Sequence Diagram	74
6.9	Population Registry Data Model	82
6.10	Subscription & Notification Process	83
6.11	readPersonAttributes Sequence Diagram	86
6.12	matchPersonAttributes Sequence Diagram	86
6.13	verifyPersonAttributes Sequence Diagram	87
6.14	queryPersonUIN Sequence Diagram	87
6.15	queryPersonList Sequence Diagram	88
6.16	readDocument Sequence Diagram	88
6.17	generateUIN Sequence Diagram	90

6.18 Biometric Data Model 98

6.19 Credential Data Model 102

HTTP ROUTING TABLE

`/${request.query.callback}`

POST `${request.query.callback}`, 224

`/notify_URL`

POST `notify_URL`, 112

`/v1`

GET `/v1/attributes/{attributeSetName}/{identifier}`, 254

GET `/v1/credentialRequests/{credentialRequestId}`, 234

GET `/v1/credentials/{credentialId}`, 239

GET `/v1/enrollments/{enrollmentId}`, 126

GET `/v1/enrollments/{enrollmentId}/buffer/{bufferId}`, 137

GET `/v1/galleries`, 165

GET `/v1/galleries/{galleryId}`, 166

GET `/v1/identify/{taskId}`, 262

GET `/v1/persons`, 118

GET `/v1/persons/{personId}`, 147

GET `/v1/persons/{personId}/encounters`, 179

GET `/v1/persons/{personId}/encounters/{encounterId}`, 185

GET `/v1/persons/{personId}/encounters/{encounterId}/templates`, 200

GET `/v1/persons/{personId}/identities`, 150

GET `/v1/persons/{personId}/identities/{identityId}`, 155

GET `/v1/persons/{personId}/reference`, 163

GET `/v1/persons/{uin}`, 119

GET `/v1/persons/{uin}/document`, 122

GET `/v1/subscriptions`, 108

GET `/v1/subscriptions/confirm`, 109

GET `/v1/topics`, 110

POST `/v1/attributes/{identifier}`, 256

POST `/v1/credentialProfiles`, 244

POST `/v1/credentialRequests/{credentialRequestId}`, 232

POST `/v1/credentialRequests/{credentialRequestId}/cancel`, 237

POST `/v1/credentials`, 238

POST `/v1/credentials/{credentialId}/revoke`, 242

POST `/v1/credentials/{credentialId}/status`, 243

POST `/v1/credentials/{credentialId}/suspend`, 240

POST `/v1/credentials/{credentialId}/unsuspend`, 241

POST `/v1/enrollments`, 134

POST `/v1/enrollments/{enrollmentId}`, 124

POST `/v1/enrollments/{enrollmentId}/buffer`, 136

POST `/v1/identify`, 258

POST `/v1/identify/{galleryId}`, 205

POST `/v1/identify/{galleryId}/{personId}`, 208

POST `/v1/identify/{galleryId}/{personId}/encounters/{encounterId}`, 211

POST /v1/persons, 145
 POST /v1/persons/{personIdTarget}/merge/{personIdSource}, 149
 POST /v1/persons/{personIdTarget}/move/{personIdSource}/encounters/{encounterId}, 195
 POST /v1/persons/{personIdTarget}/move/{personIdSource}/identities/{identityId}, 161
 POST /v1/persons/{personId}, 146
 POST /v1/persons/{personId}/encounters, 176
 POST /v1/persons/{personId}/encounters/{encounterId}, 182
 POST /v1/persons/{personId}/identities, 152
 POST /v1/persons/{personId}/identities/{identityId}, 154
 POST /v1/persons/{uin}/match, 120
 POST /v1/persons/{uin}/verify, 121
 POST /v1/subscriptions, 106
 POST /v1/topics, 110
 POST /v1/topics/{uuid}/publish, 111
 POST /v1/uin, 117
 POST /v1/verify, 217
 POST /v1/verify/{galleryId}/{personId}, 214
 POST /v1/verify/{identifier}, 252
 PUT /v1/credentialRequests/{credentialRequestId}, 236
 PUT /v1/enrollments/{enrollmentId}, 128
 PUT /v1/enrollments/{enrollmentId}/finalize, 133
 PUT /v1/persons/{personId}, 148
 PUT /v1/persons/{personId}/encounters/{encounterId}, 189
 PUT /v1/persons/{personId}/encounters/{encounterId}/galleries, 199
 PUT /v1/persons/{personId}/encounters/{encounterId}/status, 197
 PUT /v1/persons/{personId}/identities/{identityId}, 157
 PUT /v1/persons/{personId}/identities/{identityId}/reference, 162
 PUT /v1/persons/{personId}/identities/{identityId}/status, 162
 DELETE /v1/enrollments/{enrollmentId}, 132
 DELETE /v1/persons/{personId}, 148
 DELETE /v1/persons/{personId}/encounters/{encounterId}, 192
 DELETE /v1/persons/{personId}/identities/{identityId}, 160
 DELETE /v1/subscriptions/{uuid}, 108
 DELETE /v1/topics/{uuid}, 111
 PATCH /v1/enrollments/{enrollmentId}, 130
 PATCH /v1/persons/{personId}/identities/{identityId}, 159

/{request.query.address}

POST {request.query.address}, 107

INDEX

A

ABIS, [105](#)

C

CMS, [105](#)

CR, [105](#)

Credential, [105](#)

E

Encounter, [105](#)

F

Functional systems and registries, [105](#)

G

Gallery, [105](#)

H

HTTP Status Codes, [105](#)

M

Mime Types, [105](#)

O

OSIA, [105](#)

P

PR, [106](#)

R

RFC

RFC 6750, [21](#)

RFC 7396, [39](#), [44](#), [65](#), [78](#)

RFC 7519, [21](#)

U

UIN, [106](#)