ITUPublications
Recommendations

International Telecommunication Union
Standardization Sector

Recommendation

ITU-T X.1281 (03/2024)

SERIES X: Data networks, open system communications and security

Cyberspace security – Identity management (IdM) and Authentication

APIs for interoperability of identity management systems

ITU-T X-SERIES RECOMMENDATIONS

Data networks, open system communications and security

PUBLIC DATA NETWORKS	X.1-X.199
OPEN SYSTEMS INTERCONNECTION	X.200-X.299
INTERWORKING BETWEEN NETWORKS	X.300-X.399
MESSAGE HANDLING SYSTEMS	X.400-X.499
DIRECTORY	X.500-X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600-X.699
OSI MANAGEMENT	X.700-X.799
SECURITY	X.800-X.849
OSI APPLICATIONS	X.850-X.899
OPEN DISTRIBUTED PROCESSING	X.900-X.999
INFORMATION AND NETWORK SECURITY	X.1000-X.1099
SECURE APPLICATIONS AND SERVICES (I)	X.1100-X.1199
CYBERSPACE SECURITY	X.1200-X.1299
Cybersecurity	X.1200-X.1229
Countering spam	X.1230-X.1249
Identity management (IdM) and Authentication	**X.1250-X.1299**
SECURE APPLICATIONS AND SERVICES (II)	X.1300-X.1499
CYBERSECURITY INFORMATION EXCHANGE	X.1500-X.1599
CLOUD COMPUTING SECURITY	X.1600-X.1699
QUANTUM COMMUNICATION	X.1700-X.1729
DATA SECURITY	X.1750-X.1799
INTERNATIONAL MOBILE TELECOMMUNICATIONS (IMT) SECURITY	X.1800-X.1839
METAVERSE AND DIGITAL TWIN SECURITY	X.2000-X.2199
SOFTWARE SUPPLY CHAIN SECURITY	X.2150-X.2199
ARTIFICIAL INTELLIGENCE (AI) / MACHINE LEARNING (ML) SECURITY	X.2200-X.2249

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.1281

APIs for interoperability of identity management systems

Summary

Recommendation ITU-T X.1281 describes a set of standardized application program interfaces (APIs) needed to connect the multiple building blocks of an identity management solution.

NOTE – This Recommendation is technically equivalent to the OSIA specification (see [b-OSIA] in the bibliography).

History*

Edition	Recommendation	Approval	Study Group	Unique ID
1.0	ITU-T X.1281	2024-03-01	17	11.1002/1000/15662

Keywords

Authentication, identity management, interoperability.

* To access the Recommendation, type the URL https://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/. Implementers should also be aware that the organization that originated the technically equivalent document listed in the Bibliography may have received notices of intellectual property required for the implementation of this Recommendation.

Table of Contents

Introduction

Identity systems that are built on specific vendor solutions lack interoperability among the main system modules. Lack of vendor and technology interoperability causes difficulties in replacing a building block of an identity management system from vendor A with an equivalent building block from vendor B or when expanding the scope of an existing system by linking to new building blocks. The main technology barrier is the lack of standardized interfaces (APIs). Building blocks are often unable to communicate with each other due to varying interfaces (APIs) and data formats, making it difficult to swap out building blocks or add new ones to the system.

This Recommendation describes a set of standardized application program interfaces (APIs) needed to connect the multiple building blocks of an identity management solution.

Recommendation ITU-T X.1281

APIs for interoperability of identity management systems

1 Scope

This Recommendation specifies a set of standardized interfaces (APIs) needed to connect the multiple building blocks of an identity management system.

NOTE – This Recommendation is technically equivalent to [b-OSIA].

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[IETF RFC 2119] IETF RFC 2119 (1997), *Key words for use in RFCs to Indicate Requirement Levels. Best Current Practice*.

[IETF RFC 3230] IETF RFC 3230 (2002), *Instance Digests in HTTP*.

[IETF RFC 6750] IETF RFC 6750 (2012), *The OAuth 2.0 Authorization Framework: Bearer Token Usage*.

[IETF RFC 6838] IETF RFC 6838 (2013), *Media Type Specifications and Registration Procedures*.

[IETF RFC 7396] IETF RFC 7396 (2014), *JSON Merge Patch*.

[IETF RFC 7519] IETF RFC 7519 (2015), *JSON Web Token (JWT*).

[ISO 8601-1] ISO 8601-1:2019, *Date and time – Representations for information interchange – Part 1: Basic rules*.

[ISO/IEC 19794] ISO/IEC 19794, *Information technology – Biometric data interchange formats – Part 1:2011 – Framework* plus *Part 5:2011 – Face image data* plus *Part 6:2011 – Iris image data*.

3 Definitions

3.1 Terms defined elsewhere

None.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 automated biometric identification system (ABIS) building block: 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.

3.2.2 credential management system (CMS) building block: A system to manage the production, issuance, and lifecycle management of credentials such as identity cards, passports,

driving licenses, digital ID/DTC/driving license, etc. It does not manage the usage of the issued credentials and related user account data (see Identity Provider).

3.2.3 digital credential issuance & distribution system building block: A system in charge of the issuance and delivery of the digital credentials built within the identity databases under the control of the CMS.

3.2.4 enrolment building block: A system to register biographic and/or biometric data of individuals. It is composed of enrolment client and server.

3.2.5 functional systems and registries: Databases managing data including voter rolls, land registry, vehicle registration, passport, residence registry, education, and health.

3.2.6 identity provider building block: A system that creates, maintains, and manages credentials e.g., login/password and provides authentication services to relying on applications within a federation or distributed network. Identity providers offer user authentication as a service.

3.2.7 third party services building block: A system that interfaces with external components that need to leverage identity databases for verification purposes. It provides services to biometrically authenticate, identify, and access identity attributes for use cases such as Know Your Customer (KYC).

3.2.8 unique identity number (UIN) generator building block: A system to generate and manage unique identifiers.

3.2.9 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.

3.2.10 civil registry (CR) building block: The continuous, permanent, compulsory and universal recording of the occurrence and characteristics of vital events pertaining to the population, as provided through decree or regulation is accordance with the legal requirement in each country. Civil registration is carried out primarily for the purpose of establishing the documents provided by the law.

3.2.11 encounter: An 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).

3.2.12 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.

3.2.13 population registry (PR) building block: 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 ad- ministrative 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.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

ABIS Automated Biometric Identification System

APIs Advanced Programming Interfaces

CR	Civil Registry
CMS	Credential Management System
DTC	Digital Travel Credential
ID	Identity
KYC	Know Your Customer
OSIA	Open Standard Identity APIs
PR	Population Registry
UIN	Unique Identity Number

5 Conventions

In the body of this Recommendation, the words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this Recommendation are to be interpreted as described in [IETF RFC 2119].

1) MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2) MUST NOT: This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3) SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4) SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful, but the full implications should be understood, and the case carefully weighed before implementing any behaviour described with this label.
5) MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional.

A vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein, an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Code samples highlighted in blocks appear like this:

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

}
```

6 Introduction

Lack of vendor and technology neutrality causes difficulties in replacing a building block of an identity management system from vendor A with an equivalent building block from vendor B, or expand the scope of an existing system by linking to new building blocks. The main technology barrier is the lack of standardized interfaces (APIs). Building blocks are often unable to communicate with each other due to varying interfaces (APIs) and data formats, making it difficult to swap out building blocks or add new ones to the system.

This Recommendation addresses the vendor lock-in concern by providing a set of standardized interfaces needed to connect the multiple building blocks of an identity management system.

NOTE – The specified APIs in this Recommendation can cover different use cases based on an organization requirement. The implementation of APIs in this Recommendation should adhere to relevant security and compliance standards within an organization based on its security policy. Care should be taken to ensure that the APIs are secured with proper access control to ensure the privacy of user identities. The trust of the data as used in this Recommendation is a function of the data source which means that authoritative sources such as government sources will be highly reliable.

Overview of building blocks

The building blocks identified as part of the identity management system are represented in Figure 1.

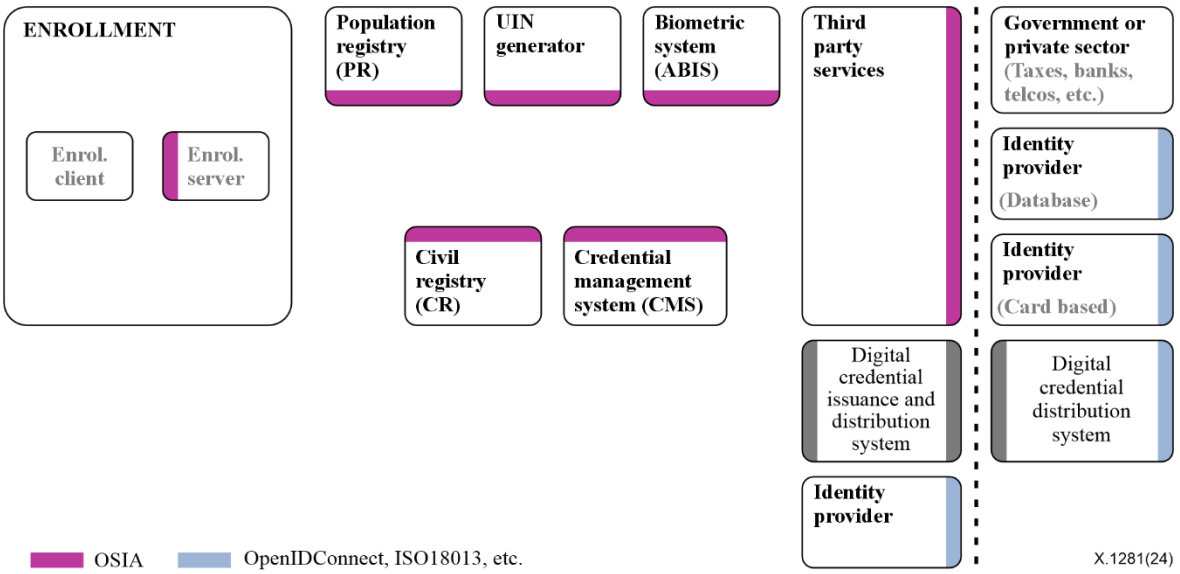


Figure 1 – Building blocks identified as part of the identity management system

NOTE – Unlike all other building blocks, the Identity Provider building block is not expected to implement any standardized interface but rather to consume them.

7 Overview of the interfaces (APIs)

This clause provides an overview of the standardized 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:

  – All persons recorded in a registry have a UIN that is considered a key to access the person's data for all records. Please note that the UIN does not have to be the same throughout all registries.

  – The registries (civil, population, or other) are considered as centralized systems that are connected. If one registry is architectured in a decentralized way, one of its component must be centralized, connected to the network, and in charge of the exchanges with the other registries.

  – 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.

  - 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.
  - 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.

Table 1 describes in detail the interfaces and associated services.

Table 1 – 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 receivers 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

Table 1 – Interfaces List

Services	Description
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
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

Table 1 – Interfaces List

Services	Description
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

8 Functional specifications of the interfaces (APIs)

This clause provides the functional specifications of the interfaces and related services.

8.1 Notification

The subscription and notification process is managed by a middleware and is described in Figure 2.

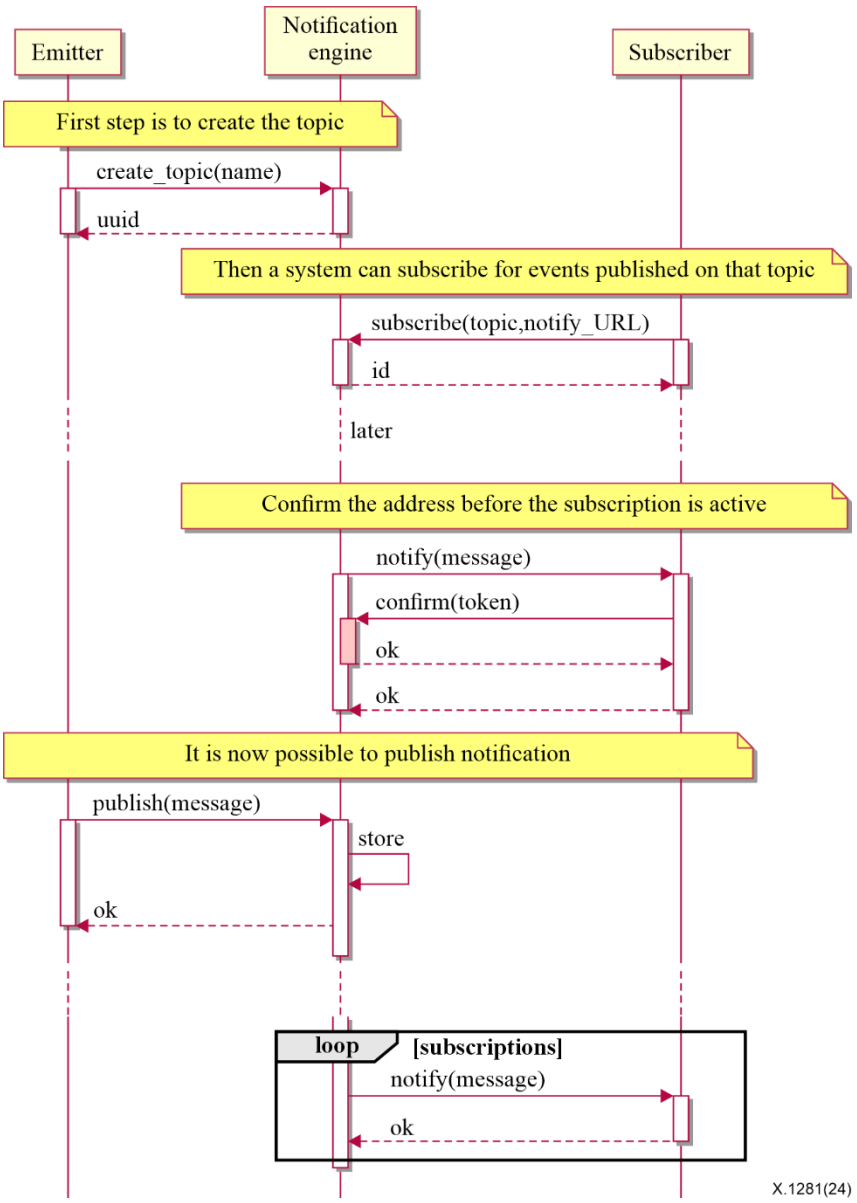


Figure 2 – Subscription and notification process

8.1.1 Services

8.1.1.1 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.

8.1.1.2 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).

8.1.1.3 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

8.1.2 Dictionary

As an example, Table 2 contains a list of events that each building block might handle.

Table 2 – Event type

Event type	Emitted by CR	Emitted by PR
Live birth	✓	
Death	✓	
Fœtal Death	✓	
Marriage	✓	
Divorce	✓	
Annulment	✓	
Separation, judicial	✓	
Adoption	✓	
Legitimation	✓	

Table 2 – Event type

Event type	Emitted by CR	Emitted by PR
Recognition	✓	
Change of name	✓	
Change of gender	✓	
New person		✓
Duplicate person	✓	✓

8.2 Data Access

8.2.1 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 population registry (PR).

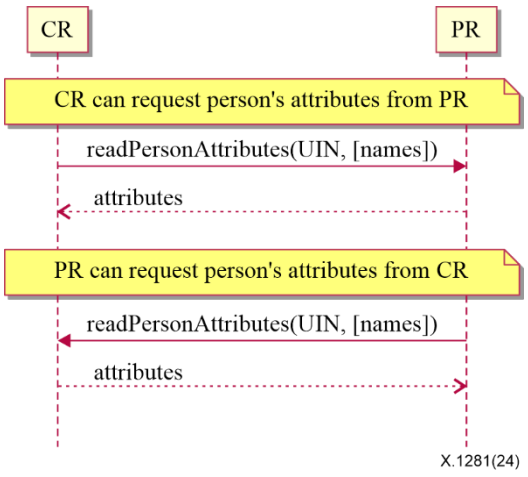


Figure 3 – readPersonAttributes sequence diagram

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.

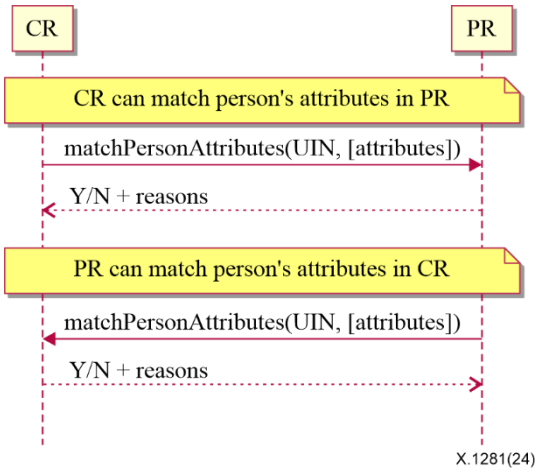


Figure 4 – matchPersonAttributes sequence diagram

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

- IN (*str*) – The person's UIN
- expressions (*list[(str,str,str)]*) – The expressions to evaluate. Each expres- sion 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.

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

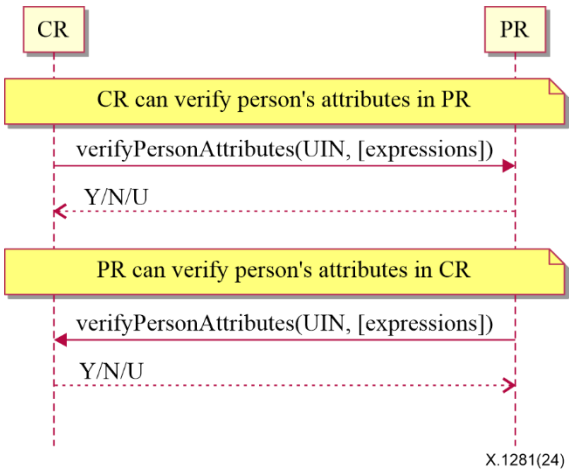


Figure 5 – verifyPersonAttributes sequence diagram

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 at- tribute 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

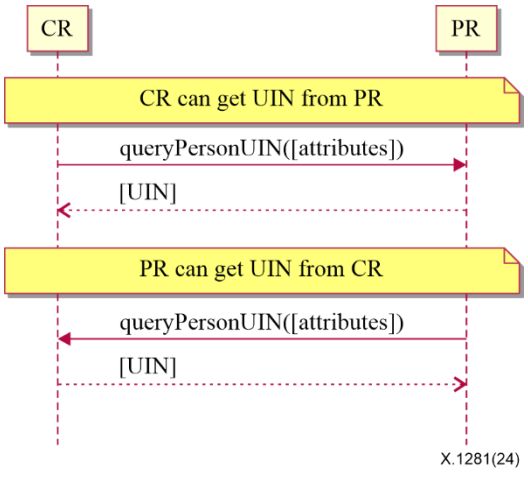


Figure 6 – queryPersonUIN sequence diagram

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

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.

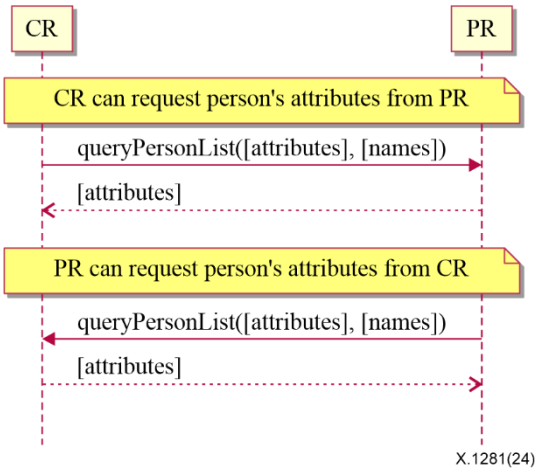


Figure 7 – queryPersonList sequence diagram

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.

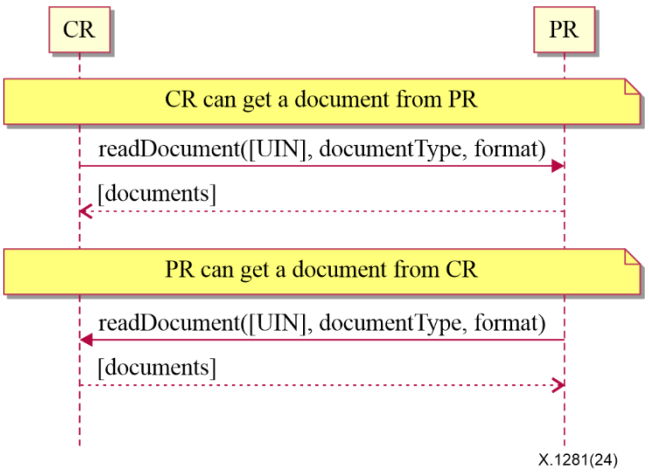


Figure 8 – readDocument sequence diagram

8.2.2 Dictionaries

As an example, Tables 3 to 7 list the attributes/documents that each component might handle.

Table 3 – Person attributes

Attribute Name	In CR	In PR
UIN	✓	✓
first name	✓	✓
last name	✓	✓
spouse name	✓	✓
date of birth	✓	✓
place of birth	✓	✓
gender	✓	✓
date of death	✓	✓
place of death	✓	
reason of death	✓	
status		✓

Table 4 – Certificate attributes

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

Table 5 – Union attributes

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

Table 6 – Filiation attributes

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

Table 7 – Document type

Document Type
birth certificate
death certificate
marriage certificate

8.3 UIN Management

8.3.1 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.

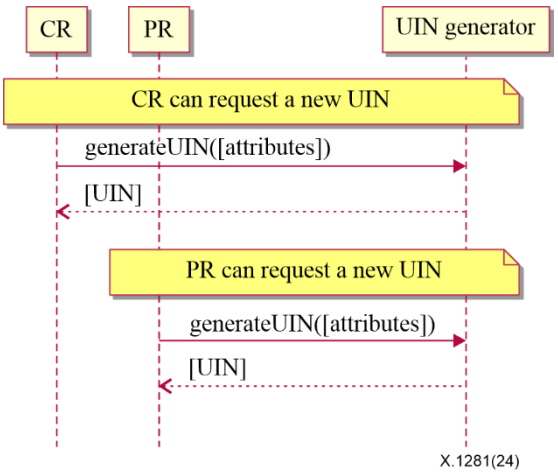


Figure 9 – generateUIN sequence diagram

8.4 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/IEC 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

---

8.4.1 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 [IETF 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 expres- sion 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 in- tegrity 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.

8.4.2 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.

8.4.3 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.

8.4.4 Data Model

Type	Description	Example(s)
Enrollment	The full set of data which are captured for one purpose.	N/A
Document Data	The documents used as an element of proof for part of the enrollment data.	Birth certificate, invoice.
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 sup- ports 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 the identity	firstName, lastName, dateOfBirth, etc.
Enrollment Flags	a dictionary (list of names and values) for custom flags controlling the enrollment process.	maximum time allowed to finish the enrollment, etc.
Request Data	a dictionary (list of names and values) for data related to the process initiated by the enrollment.	type of request, priority of execution, type of cre- dential to produce, etc.
Contextual Data	A dictionary (list of names and values) for data related to the enrollment itself	operatorName, enrollmentDate, etc.
Attributes	Generic name for any information collected during an enrollment. Attributes can apply on biographic data, document data, request data, or enrollment flag data.	firstName, lastName, enrollmentDate, etc.
Expressions	An expression combines an attribute's name, an operator (one of <, >, =, >=, <=, !=) and a value. It is used in search services.	firstName=John

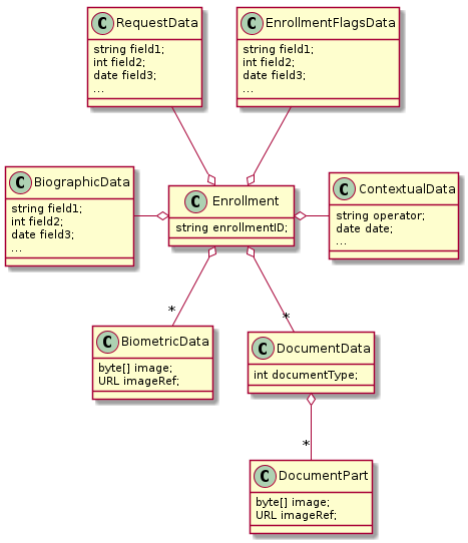


Figure 10 – Enrolment Data Model

8.5 Population Registry Services

This interface describes services to manage a registry of the population in the context of an identity management 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.

8.5.1 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 expres- sion 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 [IETF 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.

8.5.2 Data Model

Table 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.

Table 8 – Population Registry Data Model

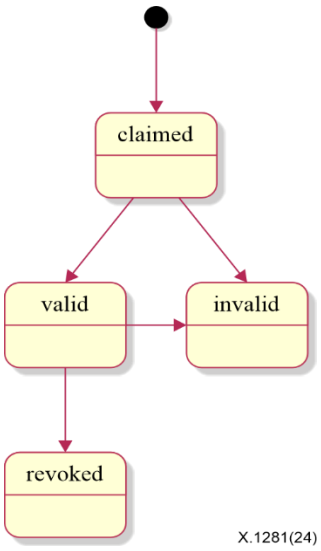
Type	Description	Example
Person	Person who is known to an identity assurance system. A person record has: • a status, such as active or inactive, defining the status of the record (the record can be excluded from queries based on this status), • a physical status, such as alive or dead, defining the 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	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). 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.  The attributes are separated into two categories: the biographic data and the contextual data.	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.

Table 8 – Population Registry Data Model

Type	Description	Example
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/IEC 19794]. [ISO/IEC 19794] allows multiple encoding and sup- ports 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
Document	The document data (images) attached to the identity and used to validate it.	Birth certificate, invoice

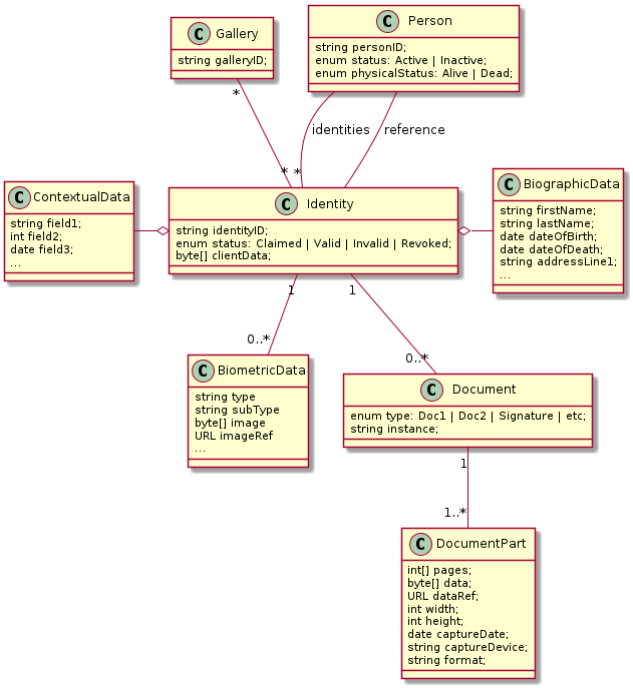


Figure 11 – Population Registry Data Model

8.6 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/IEC 19794].

About Synchronous and Asynchronous processes

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.

8.6.1 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 en-

counters 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.

moveEncounter(*personID1, personID2, encounterID, callback, transactionID, options*)

Move one single encounter from one person to another person. Encounter ID is 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 the encounter
- personID2 (*str*) – The ID of the person that will give one encounter
- encounterID (*str*) – the ID of the encounter in personID2 that will be moved in personID1.
- 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.

8.6.2 Options

Table 9 – Biometric Services Option

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.).

8.6.3 Data Model

Table 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.	VIP, Wanted, etc.
Person	Person who is known to an identity assurance system.	N/A

Table 10 – Biometric Data Model

Type	Description	Example
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). An encounter has a status indicating if this encounter is used in the biometric searches. Allowed values are active or inactive.	N/A
Biographic Data	A dictionary (list of names and values) giving the biographic data of interest for the biometric services. This should be as limited as possible.	gender, region, yearOfBirth
Filters	A dictionary (list of names and values or *range* of values) describing the filters during a search. Filters can apply on biographic data, contextual data, or encounter type.	gender, yearOfBirthMin, yearOfBirthMax
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 sup- ports 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	personId
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)	3000
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

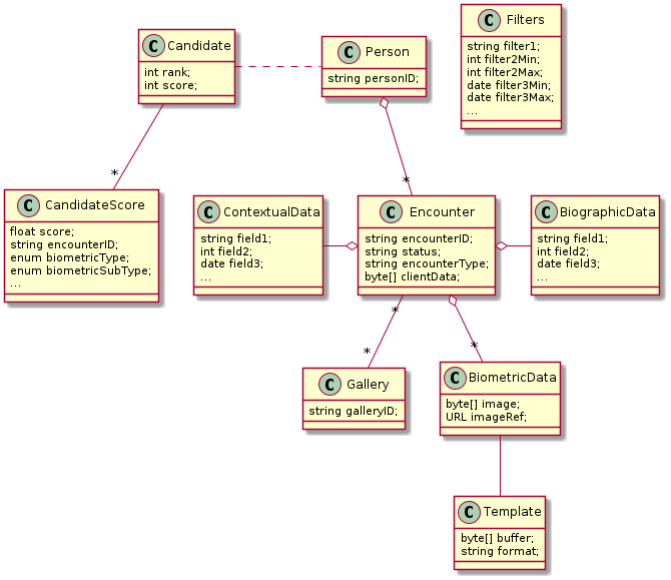


Figure 12 – Biometric Data Model

8.7 Credential Services

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

8.7.1 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 expres- sion 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 profils that match the passed in search criteria

Authorization: cms.profile.read

Parameters

- expressions (*list[(str,str,str)]*) – The expressions to evaluate. Each expres- sion 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.

8.7.2 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.

8.7.3 Data Model

Table 11 – Credential Data Model

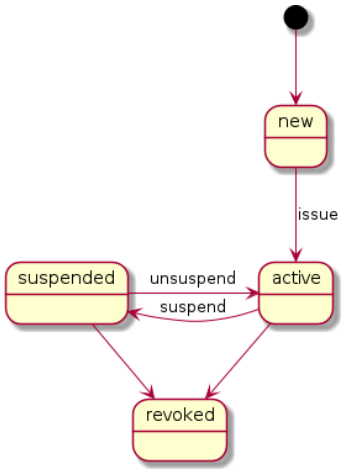
Type	Description	Example
Credential	The attributes of the credential itself The proposed transitions for the status are represented below. It can be adapted if needed. 	ID, status, dates, serial number

Table 11 – Credential Data Model

Type	Description	Example
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.	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

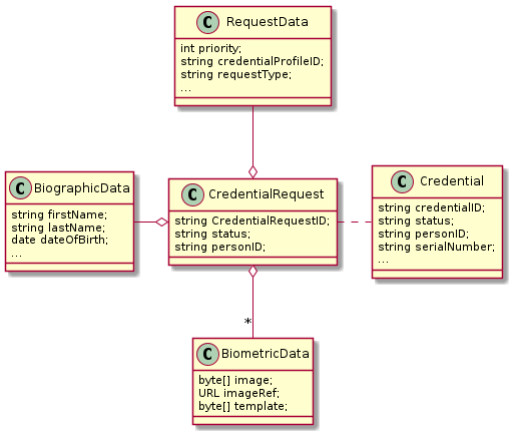


Figure 13 – Credential Data Model

8.8 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 enrollment

  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 (out of scope for the current recommendation)

  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.

8.8.1 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.

8.8.1.1 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 etc. 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.

8.8.1.2 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:

Table 12 – Attributes Set name

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 eI-DAS pivotal attributes[1].	Mandatory attributes: • First name • Last name • DoB • Identifier Optional attributes: • Birth name • Place of birth • Gender • Current address

8.8.1.3 Output Attribute set

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

9 Authorization

This clause describes how to secure the APIs through the usage of standard JWT but not how to generate and protect such tokens, nor how to secure a complete identity management system.

9.1 Principles

The following principles are implemented to secure the APIs:

- 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" [IETF RFC 7519].
- Tokens of type "Bearer Token" are used. [IETF RFC 6750] The generation and management of those tokens are not in the scope of this Recommendation.
- Validating the token is the responsibility of the service implementation, with the help of components not described in this Recommendation (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 Recommendation.
- 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 Recommendation.

NOTE – The following recommendation does not define ACL (Access Control List) to protect the access to a subset of the data.

Warning: Bearer tokens are sensitive and subject to security issues if not handled properly. Please refer to [b-JSON Web Token] for advice on proper implementation.

9.2 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.

9.3 Scopes

The Table 13 gives a summary of all scopes defined in the Recommendation.

Table 13 – Scopes List

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

Table 13 – Scopes List

Services	Scope
Query Person List	pr.person.read or cr.person.read
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

Table 13 – Scopes List

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

Annex A

Technical Specifications of the interfaces (APIs)

(This annex forms an integral part of this Recommendation.)

A.1 Notification

This is version 1.2.0 of this interface.

Notification Services

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

A.1.1 Services

Subscriber

POST /v1/subscriptions

Subscribe to a topic

Subscribes a client to receive event notifications.

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": "2022-12-16T14:51:39.008263"

}
```

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
```

```
{
    "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
```

Example response:

```
HTTP/1.1 500 Internal Server Error

Content-Type: application/json

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

A.1.2 Notification message

This clause 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 A.1 – Event Type and Message

Event type	Message

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

death	• source: identification of the system emitting the event
	• uin of the dead person
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

birthCancellation	• source: identification of the system emitting the event
	• uin of the person whose birth declaration is being cancelled
	Example:
	{
	"source": "systemX",
	"uin": "123456789",
	}

foetalDeath	• source: identification of the system emitting the event
	• uin of the new born
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

marriage	• source: identification of the system emitting the event
	• uin1 of the first conjoint
	• uin2 of the second conjoint
	Example:
	{
	"source": "systemX",
	"uin1": "123456789",
	"uin2": "234567890"
	}

divorce	• source: identification of the system emitting the event
	• uin1 of the first conjoint
	• uin2 of the second conjoint
	Example:
	{
	"source": "systemX",
	"uin1": "123456789",
	"uin2": "234567890"
	}

Table A.1 – Event Type and Message

Event type	Message

annulment	• source: identification of the system emitting the event
	• uin1 of the first conjoint
	• uin2 of the second conjoint
	Example:
	{
	"source": "systemX",
	"uin1": "123456789",
	"uin2": "234567890"
	}

separation	• source: identification of the system emitting the event
	• uin1 of the first conjoint
	• uin2 of the second conjoint
	Example:
	{
	"source": "systemX",
	"uin1": "123456789",
	"uin2": "234567890"
	}

adoption	• source: identification of the system emitting the event
	• uin of the child
	• uin1 of the first parent
	• uin2 of the second parent (optional)
	Example:
	{
	"source": "systemX",
	"uin": "123456789",
	"uin1": "234567890"
	}

legitimation	• source: identification of the system emitting the event
	• uin of the child
	• uin1 of the first parent
	• uin2 of the second parent (optional)
	Example:
	{
	"source": "systemX",
	"uin": "987654321",
	"uin1": "123456789",
	"uin2": "234567890"
	}

recognition	• source: identification of the system emitting the event
	• uin of the child
	• uin1 of the first parent
	• uin2 of the second parent (optional)
	Example:
	{
	"source": "systemX",
	"uin": "123456789",
	"uin2": "234567890"
	}

Table A.1 – Event Type and Message

Event type	Message

changeOfName	• source: identification of the system emitting the event
	• uin of the person
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

changeOfGender	• source: identification of the system emitting the event
	• uin of the person
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

updatePerson	• source: identification of the system emitting the event
	• uin of the person
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

newPerson	• source: identification of the system emitting the event
	• uin of the person
	Example:
	{
	"source": "systemX", "uin": "123456789"
	}

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

Note: Anonymized notification of events will be treated separately.

A.2 UIN management

This is version 1.2.0 of this interface.

UIN Management

- *generateUIN*

A.2.1 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"

}
```

A.3 Data access

This is version 1.3.0 of this interface.

Data Access Services

- *queryPersonList*
- *readPersonAttributes*
- *matchPersonAttributes*
- *verifyPersonAttributes*
- *readDocument*

A.3.1 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. (Re- quired)

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 avail- able at this format, it must be converted. TBD: one format for certificate data. . Con- straints: 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"
}
```

A.3.2 Data model

Person Attributes

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

Table A.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 A.3 – Matching Error Object

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

Expression

Table A.4 – Expression Object

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

Error

Table A.5 – Error Object

Attribute	Type	Description
code	32 bits integer	Error code
message	String	Error message

A.4 Enrollment

This is version 1.2.1 of this interface.

Enrollment Services

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

A.4.1 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.
- 409 Conflict – Creation not allowed, enrollmentId already exists.
- 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", "...": "..."
    },
```

(continues on next page)

(continued from previous page)


```
"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,
```

(continued from previous page)

```
                    "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",
```

(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"
}
```

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",
        "...": "..."
    },
```

(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"
    }
        ],
```


(continues on next page)

(continued from previous page)


```
    "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 [IETF RFC 7396].

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",
        "...": "..."
},
```

(continues on next page)

(continued from previous page)


```
    "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",
                    "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", "...":
            "..."
        },
            "requestData":
                { "priority"
                : 1,
                "requestType": "FIRST_ISSUANCE",
                "deliveryAddress": {
                    "address1": "11 Rue des
                    Rosiers", "city": "Libourne",
                    "postalCode":
                    "33500", "country":
                    "France"
                },
                "...": "..."
            },
            "contextualData":
                { "enrollmentDate":
                "2019-01-11", "...":
                "..."
            },
```

(continues on next page)

(continued from previous page)


```
"biographicDat
    a":
    {  "firstNam
    e": "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"

    }

],
```


(continues on next page)

(continued from previous page)


```
            "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"
}
```


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 [IETF 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=thvDyvhfIqlvFe+A9MYgxAfm1q5=
```

Example request:

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

Example response:

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

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Digest: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=
{
    "code":          1,
    "message": "string"
}
```

Example response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
Digest: SHA=thvDyvhfIqlvFe+A9MYgxAfm1q5=
{
    "code":          1,
    "message": "string"
}
```

A.4.2 Data Model

Error

Table A.6 – Error

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

Enrollment

Table A.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 *EnrollmentFlags*	The enrollment custom flags (i.e., the properties of the enrollment process). Can be extended.	
requestData	Object of type *RequestData*	The data describing the request associated to the enrollment (i.e., why the enrollment is done). Can be extended.	
contextualData	Object of type *ContextualData*		

Table A.7 – Enrollment

Attribute	Type	Description	Required
biographicData	Object of type *BiographicData*	The set of biographic data.	
biometricData	Array of *BiometricData*		
documentData	Array of *DocumentData*		

RequestData

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

Table A.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 A.9 – EnrollmentFlags

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

ContextualData

Table A.10 – ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

BiographicData

The set of biographic data.

Table A.11 – BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

BiometricData

Table A.12 – BiometricData

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

Table A.12 – BiometricData

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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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.	

Table A.12 – BiometricData

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 [IETF RFC 6838].	
resolution	integer	the image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of *MissingType*	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 A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

DocumentPart

Table A.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.	

Table A.14 – DocumentPart

Attribute	Type	Description	Required
mimeType	string	The nature and format of the document. The mime type definitions should be in compliance with [IETF RFC 6838].	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the document part.	

DocumentData

Table A.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 distinct documents of the same type (ex: two passports).	
parts	Array of *DocumentPart*	Constraints: minItems is 1	Yes

Expression

Table A.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 A.17 – Expressions

Attribute	Type	Description	Required
N/A	Array of *Expression*		

A.5 Population Registry Management

This is version 1.4.1 of this interface.

Population Registry Services

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

A.5.1 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.
- 409 Conflict – Creation not allowed, personId already exists.
- 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 409 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.
- 409 Conflict – Creation not allowed, conflict of identityId.
- 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":   {
            "firstName":
            "John",
            "lastName":        "Doo",
            "dateOfBirth":  "1985-11-
            30",   "gender":       "M",
            "nationality":  "FRA",
            "...": "..."
        },
```

(continues on next page)

(continued from previous page)


```
        "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"

}
```

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",
        "...": "..."

    },
```

(continues on next page)

(continued from previous page)


```
"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,
```


(continues on next page)

(continued from previous page)


```
                    "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.
- 409 Conflict – Creation not allowed, identityId already exists.
- 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",
            "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/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",
```

(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"

}
```

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"

    ],
```

(continues on next page)

(continued from previous page)


```
"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,
        "qualityFormat":    "string",
        "algorithm":        "string",
        "vendor": "string"

    }

],

"documentData":  [

    {
        "documentType":               "FORM",
        "documentTypeOther":          "string",
        "instance": "string",

        "parts": [

            {

                "pages": [

                    1

                ],
```


(continues on next page)

(continued from previous page)


```
                    "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 [IETF RFC 7396].

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 409 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.
- 409 Conflict – Operation not allowed, conflict of identityId.
- 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/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": [

        "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"

                }

            ],
```

(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"

}
```

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"
}
```

A.5.2 Data Model

Error

Table A.18 – Error

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

Person

Person entity.

Table A.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 A.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 *ContextualData*		
biographicData	Object of type *BiographicData*	The set of biographic data.	
biometricData	Array of *BiometricData*		
documentData	Array of *DocumentData*		

ContextualData

Table A.21 – ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

BiographicData

The set of biographic data.

Table A.22 – BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

DocumentData

Table A.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 distinct documents of the same type (ex: two passports).	
parts	Array of *DocumentPart*	Constraints: minItems is 1	Yes

DocumentPart

Table A.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.	
mimeType	string	the nature and format of the document. The mime type definitions should be in compliance with [IETF RFC 6838].	
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to capture the document part.	

BiometricData

Table A.25 – BiometricData

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

Table A.25 – BiometricData

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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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.	

Table A.25 – BiometricData

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 [IETF RFC 6838].	
resolution	integer	The image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of *MissingType*	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 in clude additional proprietary quality format.	
algorithm	string		
vendor	string		

MissingType

Table A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

Expression

Table A.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 A.28 – Expressions

Attribute	Type	Description	Required
N/A	Array of *Expression*		

A.6 Biometrics

This is version 1.5.1 of this interface.

Biometrics Services

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

A.6.1 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",
        "...": "..."
    },
```

(continues on next page)

(continued from previous page)


```
    "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",
    "ecounterId": "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 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
    "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",
        "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": [
```

(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"

            }

        ]

    }

]
```


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: 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=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"

            }

        ]

    }

]
```

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 409 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 – Operation not allowed.
- 409 Conflict – Creation not allowed, encounterId already exists.
- 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",
```

(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"

        }
    ]
}
```


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
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "encounterId":      "string",
    "status":          "ACTIVE",
    "encounterType":    "string",
    "galleries": [
        "string"
    ],
```

(continues on next page)

(continued from previous page)


```
    "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,
    "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
{
    "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",
        "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: 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 409 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 – Operation not allowed.
- 409 Conflict – Merge not allowed, conflict of encounterId.
- 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 409 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 – Operation not allowed.
- 409 Conflict – Move not allowed, conflict with the encounterId.
- 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": {
        "...": "..."
    }
}
```

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 maybe 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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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 unkown 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.
- 404 Not Found – Unknown gallery.
- 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",
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
[
    {
        "personId": "string",
        "rank": 1,
        "score": 3500,
```

(continues on next page)

(continued from previous page)


```
    "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: 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.
- 404 Not Found – Unknown person or gallery.
- 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,
                "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.
- 404 Not Found – Unknown person, gallery or encounter.

- 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": {
            "...": "..."
        }
    }
]
```

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 person or gallery.
- 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

{
    "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",
            "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"

        }

    ]

}
```

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

A.6.2 Data Model

Error

Table A.29 – Error

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

Encounter

Table A.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 *ContextualData*		
biographicData	Object of type *BiographicData*	The set of biographic data.	
biometricData	Array of *BiometricData*		Yes

ContextualData

Table A.31 – ContextualData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

BiographicData

The set of biographic data.

Table A.32 – BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

```
{
  "dateOfBirth": "1985-11-30",
  "gender": "M", "nationality":
  "FRA"

}
```

BiometricData

Table A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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	

Table A.33 – BiometricData

Attribute	Type	Description	Required
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		
mimeType	string	the nature and format of the image. The mime type definitions should be in compliance with [IETF RFC 6838].	
resolution	integer	The image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of *MissingType*	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 A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

BiometricComputedData

Table A.35 – BiometricComputedData

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

Table A.35 – BiometricComputedData

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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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 A.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 A.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 *ScoreDetail*	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 optionnally the encounterId, type and subtype of the matching bometric 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 A.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	

Table A.38 – ScoreDetail

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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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 A.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 A.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 A.41 – TaskId

Attribute	Type	Description	Required
taskId	string		Yes
others.*		Additional properties	

A.7 Credential Services

This is version 1.2.1 of this interface.

Credential Services

- *createCredentialRequest*
- *readCredentialRequest*
- *updateCredentialRequest*
- *cancelCredentialRequest*
- *findCredentials*
- *readCredential*
- *suspendCredential*
- *unsuspendCredential*
- *revokeCredential*
- *setCredentialStatus*
- *findCredentialProfiles*

A.7.1 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.
- 409 Conflict – Creation not allowed, credentialRequestId already exists.
- 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",
        "...": "..."
    },
```

(continues on next page)

(continued from previous page)


```
    "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"

}
```


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
{
    "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

{
    "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",
        "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"
        }
    ]
}
```


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":
        "2022-12-16T14:51:39.008263", "expiryDate":
        "2022-12-16T14:51:39.008263",
        "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":
    "2022-12-16T14:51:39.008263", "expiryDate":
    "2022-12-16T14:51:39.008263",
    "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
{
    "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",
        "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"

}
```

A.7.2 Data Model

Error

Table A.42 – Error

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

Address

Table A.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.	

Table A.43 – Address

Attribute	Type	Description	Required
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 A.44 – BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:


```
{

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

}
```


BiometricData

Table A.45 – BiometricData

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

Table A.45 – BiometricData

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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
instance	string	Used to separate two distinct 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.	

Table A.45 – BiometricData

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 [IETF RFC 6838].	
resolution	integer	The image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of *MissingType*	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 A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

RequestData

The data describing the request itself.

Table A.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

Table A.47 – RequestData

Attribute	Type	Description	Required
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.	
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 is- suance system.	
issuingAuthority	string		
deliveryOffice	string	Single code or name identifying the office where the credential has to be delivered.	
deliveryAddress	Object of type *Address*		
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 A.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 *RequestData*	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 *BiographicData*	The set of biographic data.	Yes
biometricData	Array of *BiometricData*		
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 A.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:	

Table A.49 – CredentialData

Attribute	Type	Description	Required
		the Ministry of Interior).	
issuingPlace	string	The place where the credential was issued (ex: Paris).	
others.*		Additional properties	

CredentialProfile

A credential profile.

Table A.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 A.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 A.52 – Expressions

Attribute	Type	Description	Required
N/A	Array of *Expression*		

A.8 ID Usage Services

A.8.1 Relying Party Services

This is version 1.1.1 of this interface.

Relying Party Services

- *verify*
- *readAttributeSet*
- *readAttributes*
- *identify*
- *readIdentifyResult*

A.8.1.1 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", "credentialNumber", . . .). 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"
        }
    ],
```

(continues on next page)

(continued from previous page)


```
    "credentialData": [
        {
            "credentialNumber":  "string",
            "personId":  "string",
            "credentialType":  "ID_CARD",
            "issuedDate":  "2022-12-16T14:51:39.008263",
            "expiryDate": "2022-12-16T14:51:39.008263",
            "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 at- tributes 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",
```

(continues on next page)

(continued from previous page)


```
            "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": [
        {
            "credentialId":          "string",
            "status":                   "NEW",
            "statusOther":          "string",
            "credentialNumber":     "string",
            "personId":             "string",
            "credentialType":  "ID_CARD",
            "issuedDate":             "2022-12-16T14:51:39.008263",
            "expiryDate": "2022-12-16T14:51:39.008263",
            "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", "credentialNumber", . . .). 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",
```

(continues on next page)

(continued from previous page)


```
            "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": [

        {
            "credentialId":          "string",
            "status":                   "NEW",
            "statusOther":          "string",
            "credentialNumber":      "string",
            "personId":             "string",
            "credentialType":  "ID_CARD",

            "issuedDate":              "2022-12-16T14:51:39.008263",
            "expiryDate": "2022-12-16T14:51:39.008263",

            "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",
            "...": "..."
            },
```

(continues on next page)

```
        "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": [
            {
                "credentialNumber":        "string",
                "personId":                "string",
                "credentialType":  "ID_CARD",
                "issuedDate":         "2022-12-16T14:51:39.008263",
                "expiryDate": "2022-12-16T14:51:39.008263",
                "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",
                        "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": "2022-12-16T14:51:39.008263",
```

(continues on next page)

(continued from previous page)


```
                "expiryDate": "2022-12-16T14:51:39.008263",
                "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
{
    "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",
                "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"
            }
        ],
        "credentialData": [
            {
                "credentialId":          "string",
                "status":                  "NEW",
                "statusOther":          "string",
                "credentialNumber":     "string",
                "personId":             "string",
                "credentialType":  "ID_CARD",
                "issuedDate":        "2022-12-16T14:51:39.008263",
                "expiryDate": "2022-12-16T14:51:39.008263",
                "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"
}
```

A.8.1.2 Data Model

AttributeSet

A set of attributes used in verify.

Table A.53 – AttributeSet

Attribute	Type	Description	Required
biographicData	Object of type *BiographicData*	The set of biographic data.	
biometricData	Array of *BiometricData*		
credentialData	Array of *CredentialData*		
contactData	Object of type *ContactData*		

BiographicData

The set of biographic data.

Table A.54 – BiographicData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:

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

ContactData

Table A.55 – ContactData

Attribute	Type	Description	Required
*		Additional properties	

Example #1:


```
{
  "email":          "John.Doo@osia.com",
  "phone1": "555666777",
  "phone2": "555888999"
```


BiometricData

Table A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, 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.	

Table A.56 – BiometricData

Attribute	Type	Description	Required
captureDate	string/date-time		
captureDevice	string	A string identifying the device used to cap ture 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_PL AIN, 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 [IETF RFC 6838].	
resolution	integer	The image resolution (in DPI).	
compression	string	Constraints: possible values are NONE, WSQ, JPEG, JPEG2000, PNG	
missing	Array of *MissingType*	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.	

Table A.56 – BiometricData

Attribute	Type	Description	Required
qualityFormat	string	Format of the quality. One of ISO_19794, NFIQ, or NFIQ2. Can be extended to include additional proprietary quality formats.	
algorithm	string		
vendor	string		

MissingType

Table A.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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
presence	string	Constraints: possible values are BANDAGED, AMPUTATED, DAMAGED	

CredentialData

A credential.

Table A.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).	
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 A.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 A.60 – VerifyResult

Attribute	Type	Description	Required
verificationCode	integer/int64		Yes
verificationMessage	string		Yes
verificationProof	string		

IdentifyRequest

A set of parameters used in identify.

Table A.61 – IdentifyRequest

Attribute	Type	Description	Required
attributeSet	Object of type *AttributeSet*	A set of attributes used in verify.	Yes
outputAttributeSetN	asmtreing	Attribute set name describing what attributes are to be read. e.g., "DE-FAULT_SET_01", "SET_BIOM_01", "EI- DAS", . . .	Yes

OutputAttributeSet

A template describing the expected attributes of a readAttributes request.

Table A.62 – OutputAttributeSet

Attribute	Type	Description	Required
outputBiographicDat	aArray of string	List of BiographicData structure fields to include in the answer.	
outputBiometricData	Array	An array of expected biometric data & fields.	
outputBiometricData biometricType	[s]tr.ing	Constraints: possible values are FACE, FINGER, IRIS, SIGNATURE, UNKNOWN	

Table A.62 – OutputAttributeSet

Attribute	Type	Description	Required
outputBiometricData biometricSubType	[s]tr.ing	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, RIGHT_HYPOTHENAR, 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, EYE_BOTH, PORTRAIT, LEFT_PROFILE, RIGHT_PROFILE	
outputBiometricData biometricDataFields	[A]r.ray of string		
outputCredentialDat	aArray	an array of expected credential type & fields.	
outputCredentialDat credentialType	as[tr]i.ng	Type of the credential. e.g., "PASSPORT", "ID_CARD", …	
outputCredentialDat credentialDataField	aA[r]r.ay of string s		
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 A.63 – TaskId

Attribute	Type	Description	Required
taskId	string		Yes
others.*		Additional properties	

A.8.1.3 Data Format

The following data formats are used in OSIA:

Table A.64 – OSIA Data Formats

Data	Format	Description
Service Description	OpenAPI 3.0.x	All OSIA interfaces are described using OpenAPI 3.0.x format [b-OpenAPI] OpenAPI can use json or YAML file, YAML is preferred for its readability.
Service Payload	json	json has builtin format for string, integer, floating number, boolean, as well as list and dictionary. json is widely used and supported by many different lan- guages and frameworks, making it ideal to favor inter- operability.
Date & Time	iso8601	[ISO 8601] defines the format for date and date and time. It supports local time as well as UTC time. [ISO 8601-2], dated 2019, can be used to represent dates with un- known part.
Biometric Images	JPEG, JPEG2000, PNG, WSQ, ISO19794	Images can be transfered as a URL or can be embedded in the json. When embedded they are encoded in base64. It is highly recommended to use a widely used format such as JPEG, PNG, or WSQ [b-WSQ] for fingerprints. ISO19794 is also recommended when possible.
Documents	JPEG, PNG, PDF	Documents captured during enrollment can use the JPEG format for monopage document. PDF is also widely recognized and very convenient to transfer both images and text in multipage documents.

Appendix I

(This appendix does not form an integral part of this Recommendation.)

There is a version for each interface. Each interface can be referenced as follows:

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

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

v. 2.0 – Notification v. 1.0.0

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

Table I.1 – APIs Version

Interfaces	Version
Notification	1.2.0
UIN Management	1.2.0
Data Access	1.3.0
Enrollment Services	1.2.1
Population Registry Services	1.4.1
Biometrics Services	1.5.1
Credential Services	1.2.1
Relying Party Services	1.1.1

Bibliography

[b-JSON Web Token] IETF RFC 8725, *JSON Web Token Best Practice*. https://datatracker.ietf.org/doc/html/rfc8725

[b-OpenAPI] OpenAPI Specification v3.0.3 (2020), https://spec.openapis.org/oas/v3.0.3

[b-OSIA] OSIA Specification, https://osia.readthedocs.io/en/stable/

[b-WSQ] NIST; FBI; *Los Alamos National Laboratory. Wavelet Scalar Quantization algorithm 2: PDF 2.0*, https://www.nist.gov/system/files/documents/2020/09/03/11-wsq_bradley_brislawn_standard_ieee_iscs_-_19940530.pdf

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks, open system communications and security**
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Published in Switzerland
Geneva, 2024