# Open Source API

*Release 1.0a*

**Jun 14, 2019**

# Contents


**1 Introduction** **2**

**2 Revision History** **3**

**3 Definitions** **4**

**4 Functional Specifications** **5**
- 4.1 Hypothesis . . . 5
- 4.2 Concepts . . . 5
- 4.3 Interface . . . 5
  - 4.3.1 Functions/Sevices . . . 5
  - 4.3.2 Dictionaries . . . 9
- 4.4 Use Cases . . . 11
  - 4.4.1 Birth Use Case . . . 11
  - 4.4.2 Death Use Case . . . 12
  - 4.4.3 Fœtal Death Use Case . . . 12
  - 4.4.4 Marriage Use Case . . . 12
  - 4.4.5 Divorce Use Case . . . 12
  - 4.4.6 Annulment Use Case . . . 12
  - 4.4.7 Separation Use Case . . . 12
  - 4.4.8 Adoption Use Case . . . 12
  - 4.4.9 Legitimation Use Case . . . 12
  - 4.4.10 Recognition Use Case . . . 12
  - 4.4.11 Change of Name/Gender Use Case . . . 12
  - 4.4.12 Transcription Use Case . . . 12
  - 4.4.13 Change of Nationality Use Case . . . 12
  - 4.4.14 Deduplication . . . 13

**5 Technical Specifications** **14**
- 5.1 Services . . . 14
  - 5.1.1 UIN Management . . . 14
  - 5.1.2 Data Access . . . 15
  - 5.1.3 Notifications . . . 19
- 5.2 Data Model . . . 19
  - 5.2.1 Person Attributes . . . 19
  - 5.2.2 Notification Message . . . 19
  - 5.2.3 Matching Error . . . 23
  - 5.2.4 Expression . . . 23
  - 5.2.5 Error . . . 24

**HTTP Routing Table** **27**

Index 28

**Version 1.0a0**

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

CHAPTER 1

# Introduction

To be completed

CHAPTER 2

# Revision History

| Version | Date | Notes |
|---|---|---|
| 1.0 | 2018-12-xx | Release of version 1.0 |
| 1.0a0 | 2018-07-31 | First alpha version |

CHAPTER 3

# Definitions

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

**CI** Civil Identity. The system in charge of the recording of selected information pertaining to each member of the resident population of a country. To be completed

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

Some examples of possible mime type definitions:

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

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

**UIN** Unique Identity Number.

To be completed

CHAPTER 4

# Functional Specifications

## 4.1 Hypothesis

The design of this interface is based on the following assumptions:

1. All persons recorded in CR have a *UIN*. The UIN can be used as a key to access person data for all records.
2. All persons recorded in CI have a UIN. The UIN can be used as a key to access person data for all records.
3. The CR and CI are both considered as centralized systems that are connected. If CR is architectured in a decentralized way, and it is often the case, one of its component must be centralized, connected to the network, and in charge of the exchanges with CI.
4. Since all instances of CR and CI are customized for each business needs, dictionaries must be explicitly defined to describe the attributes, the event types, and the document types. See *Dictionaries* for the mandatory elements of those dictionaries.
5. The relationship parent/child is not mandatory in CI. A CI implementation may manage this relationship or may ignore it and rely on CR to manage it.
6. All persons are stored in CI. There is no record in CR that is not also in CI.
7. The interface does not expose biometric services. Usage of biometrics is optional and is described in other standards already defined.

## 4.2 Concepts

To be completed

## 4.3 Interface

### 4.3.1 Functions/Sevices

This chapter describes in pseudo code the services defined between CR and CI. There three categories of services:

- UIN management. This service can be implemented by CI, by CR or by another system. We will consider it is provided by a system called *UIN Generator*.

- Notifications. When data is changed, a notification is sent and received by systems that registered for this type of events. For instance, CI can register for the events *birth* emitted by CR.
- Data access. A set of services to access data.

### UIN Management

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

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

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

This service is synchronous.

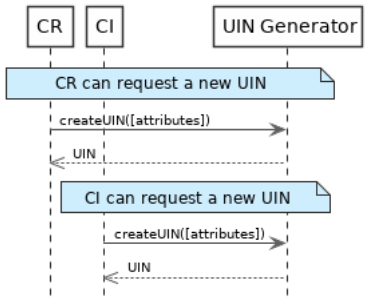


Fig. 1: `createUIN` Sequence Diagram

### Notifications

**notify**(*type*, *UIN*)
Notify of a new event all systems that subscribed to this event

**Parameters**

- **type** (*str*) – Event type
- **UIN** (*list[str]*) – Records affected by the event

**Returns** N/A

This service is asynchronous.

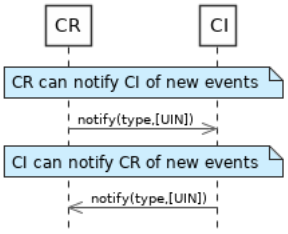


Fig. 2: `notify` Sequence Diagram

**Note:** Notifications are possible after the receiver has subscribed to an event.

### Data Access

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

**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 CI.

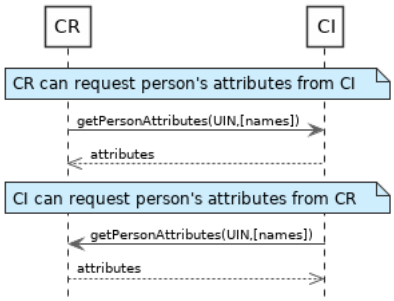


Fig. 3: getPersonAttributes Sequence Diagram

---

**matchPersonAttributes** (*UIN*, *attributes*)
Match person attributes. This service is used to check the value of attributes without exposing private data

**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 CI.

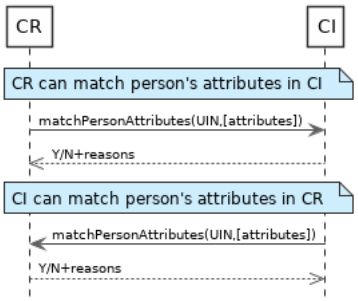


Fig. 4: matchPersonAttributes Sequence Diagram

---

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

**Parameters**

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

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

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

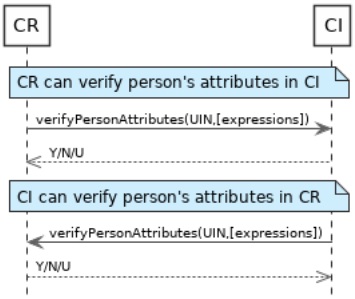


Fig. 5: `verifyPersonAttributes` Sequence Diagram

---

**getPersonUIN** (*attributes*)

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

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

**Returns** a list of matching UIN

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

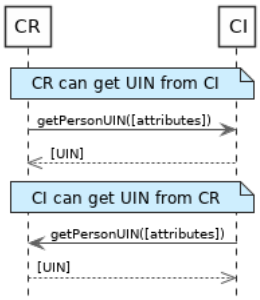


Fig. 6: `getPersonUIN` Sequence Diagram

---

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

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

**Parameters**

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

**Returns** The list of the requested documents

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

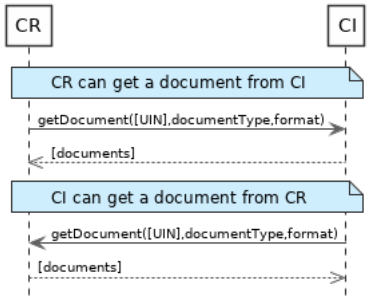


Fig. 7: `getDocument` Sequence Diagram

### 4.3.2 Dictionaries

#### Attributes

Table 1: Person Attributes

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

Table 2: Certificate Attributes

| Attribute Name | In CR | In CI | Description |
|---|---|---|---|
| officer name | ✓ | | |
| number | ✓ | | |
| date | ✓ | | |
| place | ✓ | | |
| type | ✓ | | |

Table 3: Union Attributes

| Attribute Name | In CR | In CI | Description |
|---|---|---|---|
| date of union | ✓ | | |
| place of union | ✓ | | |
| conjoint1 UIN | ✓ | | |
| conjoint2 UIN | ✓ | | |
| date of divorce | ✓ | | |

Table 4: Filiation Attributes

| Attribute Name | In CR | In CI | Description |
|---|---|---|---|
| parent1 UIN | ✓ | | |
| parent2 UIN | ✓ | | |

### Events

Table 5: Event Type

| Event Type | Emitted by CR | Emitted by CI |
|---|---|---|
| Live birth | ✓ | |
| Death | ✓ | |
| Birth cancellation | ✓ | |
| Fœtal Death | ✓ | |
| Marriage | ✓ | |
| Divorce | ✓ | |
| Annulment | ✓ | |
| Separation, judicial | ✓ | |
| Adoption | ✓ | |
| Legitimation | ✓ | |
| Recognition | ✓ | |
| Change of name | ✓ | |
| Change of gender | ✓ | |
| Person update | ✓ | ✓ |
| New person | | ✓ |
| Duplicate person | | ✓ |

### Documents

Table 6: Document Type

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

## 4.4 Use Cases

### 4.4.1 Birth Use Case

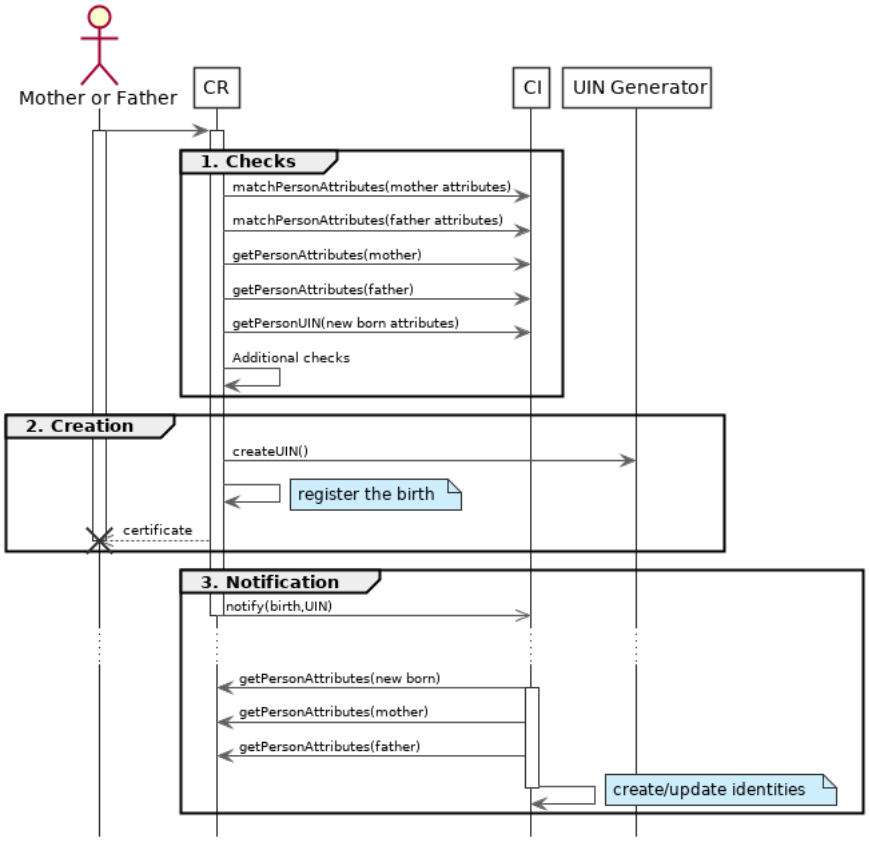


Fig. 8: Birth Use Case

1. Checks

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

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

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

2. Creation

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

3. Notification

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

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

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

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

#### 4.4.2 Death Use Case

To be completed

#### 4.4.3 Fœtal Death Use Case

To be completed

#### 4.4.4 Marriage Use Case

To be completed

#### 4.4.5 Divorce Use Case

To be completed

#### 4.4.6 Annulment Use Case

To be completed

#### 4.4.7 Separation Use Case

To be completed

#### 4.4.8 Adoption Use Case

To be completed

#### 4.4.9 Legitimation Use Case

To be completed

#### 4.4.10 Recognition Use Case

To be completed

#### 4.4.11 Change of Name/Gender Use Case

To be completed

#### 4.4.12 Transcription Use Case

To be completed

#### 4.4.13 Change of Nationality Use Case

(To be confirmed)

### 4.4.14 Deduplication

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

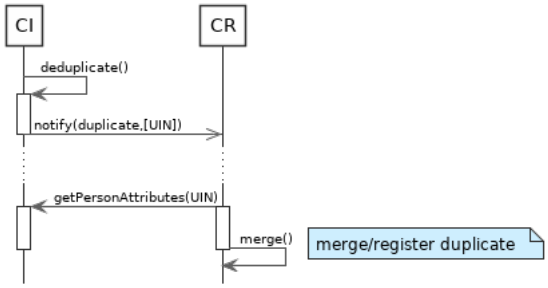


Fig. 9: Deduplication Use Case

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

CHAPTER 5

# Technical Specifications

## 5.1 Services

### 5.1.1 UIN Management

**POST /v1/uin**

Request the generation of a new UIN.

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

**Status Codes**

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

**Example request:**

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

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

**Example response:**

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

1235567890
```

**Example response:**

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

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

### 5.1.2 Data Access

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

**Query Parameters**

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

**Status Codes**

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

**Example request:**

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

**Example response:**

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

[
    "1235567890"
]
```

**Example response:**

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

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

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

**Parameters**

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

**Query Parameters**

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

**Status Codes**

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

**Example request:**

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

**Example response:**

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

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

**Example response:**

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

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

**`POST /v1/persons/{uin}/match`**

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

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

**Parameters**

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

**Status Codes**

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

**Example request:**

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

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

**Example response:**

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

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

**Example response:**

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

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

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

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

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

**Parameters**

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

**Status Codes**

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

**Example request:**

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

[
    {
        "attributeName": "firstName",
        "operator": "=",
```

(continues on next page)

(continued from previous page)

```
        "value": "John"
    },
    {
        "attributeName": "dateOfBirth",
        "operator": "<",
        "value": "1990-12-31"
    }
]
```

**Example response:**

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

true
```

**Example response:**

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

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

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

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

**Parameters**

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

**Query Parameters**

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

**Status Codes**

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

**Example request:**

```
GET /v1/persons/{uin}/document?doctype=marriage&secondaryUin=234567890&
↪format=pdf HTTP/1.1
Host: example.com
```

**Example response:**

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

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

### 5.1.3 Notifications

**Attention:** The interface to subscribe, receive, and notify events is not described here. It has not been decided if it is worth defining a neutral interface abstracting the broker and making the CR & CI provider independent from the broker selected by the integrator, or if it is better to use the native interface of the broker.

The first solution means an abstraction of the broker must be implemented, adding possible source of bugs or failures.

The second solution means the CR or CI cannot be simply replaced by a CR or CI from another vendor without some adaptation to use the interface of the broker.

To all reviewers: please comment and propose on this topic.

## 5.2 Data Model

### 5.2.1 Person Attributes

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

Table 1: 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 |

### 5.2.2 Notification Message

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

Table 2: Event Type & Message

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

Continued on next page

Table 2 – continued from previous page

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

Continued on next page

Table 2 – continued from previous page

| Event Type | Message |
|---|---|
| `legitimation` | • `source`: identification of the system emitting the event<br>• `uin` of the child<br>• `uin1` of the first parent<br>• `uin2` of the second parent (optional)<br>Example:<br>`{ "source": "systemX", "uin": "987654321", "uin1": "123456789", "uin2": "234567890" }` |
| `recognition` | • `source`: identification of the system emitting the event<br>• `uin` of the child<br>• `uin1` of the first parent<br>• `uin2` of the second parent (optional)<br>Example:<br>`{ "source": "systemX", "uin": "123456789", "uin2": "234567890" }` |
| `changeOfName` | • `source`: identification of the system emitting the event<br>• `uin` of the person<br>Example:<br>`{ "source": "systemX", "uin": "123456789" }` |
| `changeOfGender` | • `source`: identification of the system emitting the event<br>• `uin` of the person<br>Example:<br>`{ "source": "systemX", "uin": "123456789" }` |
| `updatePerson` | • `source`: identification of the system emitting the event<br>• `uin` of the person<br>Example:<br>`{ "source": "systemX", "uin": "123456789" }` |

Continued on next page

Table 2 – continued from previous page

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

**Note:** Anonymized notification of events will be treated separately.

**Attention:** Should the UIN be mandatory? What happens when a person has no UIN?

### 5.2.3 Matching Error

A list of:

Table 3: Matching Error Object

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

### 5.2.4 Expression

Table 4: Expression Object

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

### 5.2.5 Error

Table 5: Error Object

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

# List of Tables


1 Person Attributes . . . . . 9
2 Certificate Attributes . . . . . 9
3 Union Attributes . . . . . 9
4 Filiation Attributes . . . . . 10
5 Event Type . . . . . 10
6 Document Type . . . . . 10

1 Person Attributes . . . . . 19
2 Event Type & Message . . . . . 20
3 Matching Error Object . . . . . 23
4 Expression Object . . . . . 23
5 Error Object . . . . . 24

# List of Figures


1 `createUIN` Sequence Diagram . . . . . 6
2 `notify` Sequence Diagram . . . . . 6
3 `getPersonAttributes` Sequence Diagram . . . . . 7
4 `matchPersonAttributes` Sequence Diagram . . . . . 7
5 `verifyPersonAttributes` Sequence Diagram . . . . . 8
6 `getPersonUIN` Sequence Diagram . . . . . 8
7 `getDocument` Sequence Diagram . . . . . 9
8 Birth Use Case . . . . . 11
9 Deduplication Use Case . . . . . 13

# HTTP Routing Table

## /v1

GET /v1/persons, 15
GET /v1/persons/{uin}, 15
GET /v1/persons/{uin}/document, 18
POST /v1/persons/{uin}/match, 16
POST /v1/persons/{uin}/verify, 17
POST /v1/uin, 14

# Index


## C

CI, **4**
CR, **4**
createUIN() (*built-in function*), 6

## G

getDocument() (*built-in function*), 8
getPersonAttributes() (*built-in function*), 7
getPersonUIN() (*built-in function*), 8

## H

HTTP Status Codes, **4**

## M

matchPersonAttributes() (*built-in function*), 7
Mime Types, **4**

## N

notify() (*built-in function*), 6

## U

UIN, **4**

## V

verifyPersonAttributes() (*built-in function*), 7