## Introduction

The CPSV-AP API implemented for Netherlands converts the output of SRU API into CPSV-AP data model.

In order to perform such conversion a mapping analysis was done to understand which data could be extracted from the SRU API (insert real link of the api and check in the code).

After that, the CSPV-AP API has been implemented starting for the analysis of the input request and the output response, which determined the API contract.

## The SRU API

The SRU API is an XML based API which can be reached at the below URL:

<http://zoekdienst.overheid.nl/SRUServices/SRUServices.asmx/Search>

The SRU API has the below parameters:

Table 1: Parameters accepted by the SRU API

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| version | 1.2 |
| operation | searchRetrieve |
| x-connection | sc |
| recordSchema | sc4.0 |
| startRecord | 1 |
| maximumRecords | 10 |
| query | authority="Gelderland" |

For example the user can get the first 10 records on the authority Gelderland with the below URL:

[http://zoekdienst.overheid.nl/SRUServices/SRUServices.asmx/Search?version=1.2&operation=searchRetrieve&x-connection=sc&recordSchema=sc4.0&startRecord=1&maximumRecords=10&query=authority="Gelderland](http://zoekdienst.overheid.nl/SRUServices/SRUServices.asmx/Search?version=1.2&operation=searchRetrieve&x-connection=sc&recordSchema=sc4.0&startRecord=1&maximumRecords=10&query=authority=%22Gelderland)"

Only the last 3 parameters (startRecord, maximumRecords, query) have been considered to be dynamic, and therefore passed through the CPSV-AP API e.g:

[http://localhost:8080/swagger-cxf-server-1.0.0/api/PublicServices?startRecord=1&maximumRecords=10&query=authority="Gelderland](http://localhost:8080/swagger-cxf-server-1.0.0/api/PublicServices?startRecord=1&maximumRecords=10&query=authority=%22Gelderland)"

## The CPSV-AP Mapping

The CPSV-AP mapping with the SRU API is based on the multiple XML schemas, which define the structure of the output of the SRU API, mainly:

* Owms.xsd (5 properties can be mapped to CSPV-AP)
* Sc.xsd (2 properties can be mapped to CSPV-AP)
* Gzd.xsd (2 properties can be mapped to CPSV-AP)

The outcome of the mapping analysis shows that, a mapping towards CPSV-AP is possible, however the result might not be fully compliant with CPSV-AP since 2 properties (authority and productHTML) have cardinalities more relaxed than CPSV-AP,

The reader can find more details in the Mapping spreadsheet

## API Architecture

### Input request and Output response

The CSPV-AP API implemented is a REST API which has 3 input parameters:

* startRecord
* maximumRecords
* query

Which are the same input for the SRU API, see The SRU API for more details, and 1 output:

* PublicServiceDataset

The CSPV-API passes the 3 input parameters to the SRU API and adds the other parameters, it gets back the SRU Response and generates a PublicServiceDataset.

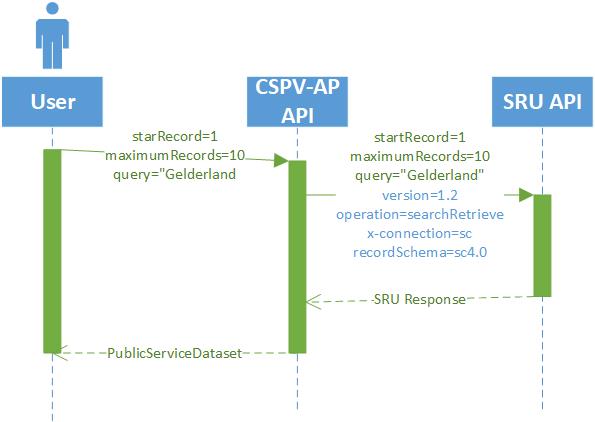


Figure 1: Sequence diagram of interaction between the CPSV-AP API and SRU API

The PublicServiceDataset will contain a list of PublicService which in turn will have the following properties/relations (based on the aforementioned The CPSV-AP Mapping):

* Identifier
* Title
* Language
* Spatial
* hasCompetentAuthority
* type
* description

While the PublicOrganization connected will have its identifier and the Spatial property.



Figure 2: JSON-LD representation of the output of the CSPV-AP API via JSON-LD playground

In the example above, the reader can find a JSON-LD output of the CPSV-AP API, presenting at the top the PublicServiceDataset including, via @context property, the CPSV-AP JSON-LD context that it reused to express linked data properties. Further, the example shows an example of PublicService with its properties

Displaying the JSON-LD as graph it would show:

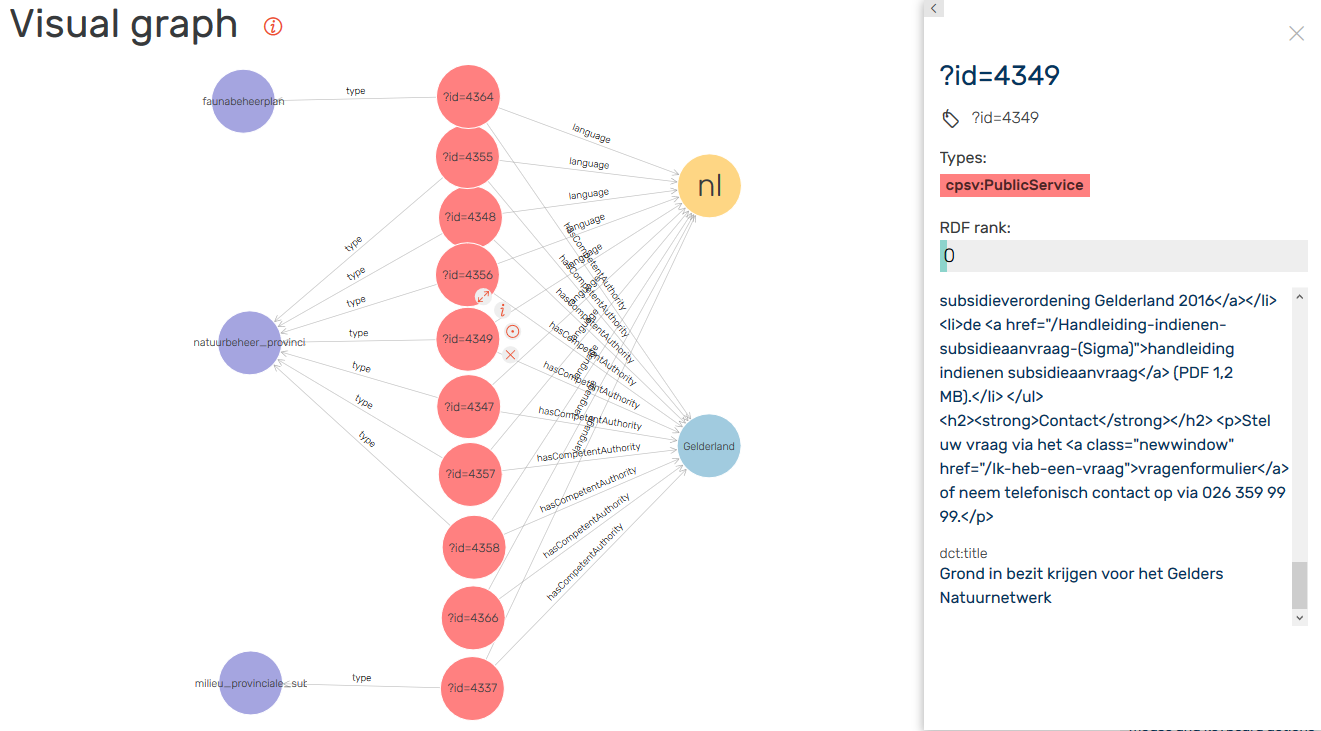


Figure 3: The output of the CPSV-API displayed as graph

In the above image, we can notice that all of the public services have the same competent authority (Gelderland) as requested to the API by the query.

### The API Contract

The API Contract has been created accordingly to the OpenAPI 2.0 (Swagger) standard using the online editor at: <https://editor.swagger.io/> which provides syntax highlighting in case of typos and displays the possible representation:

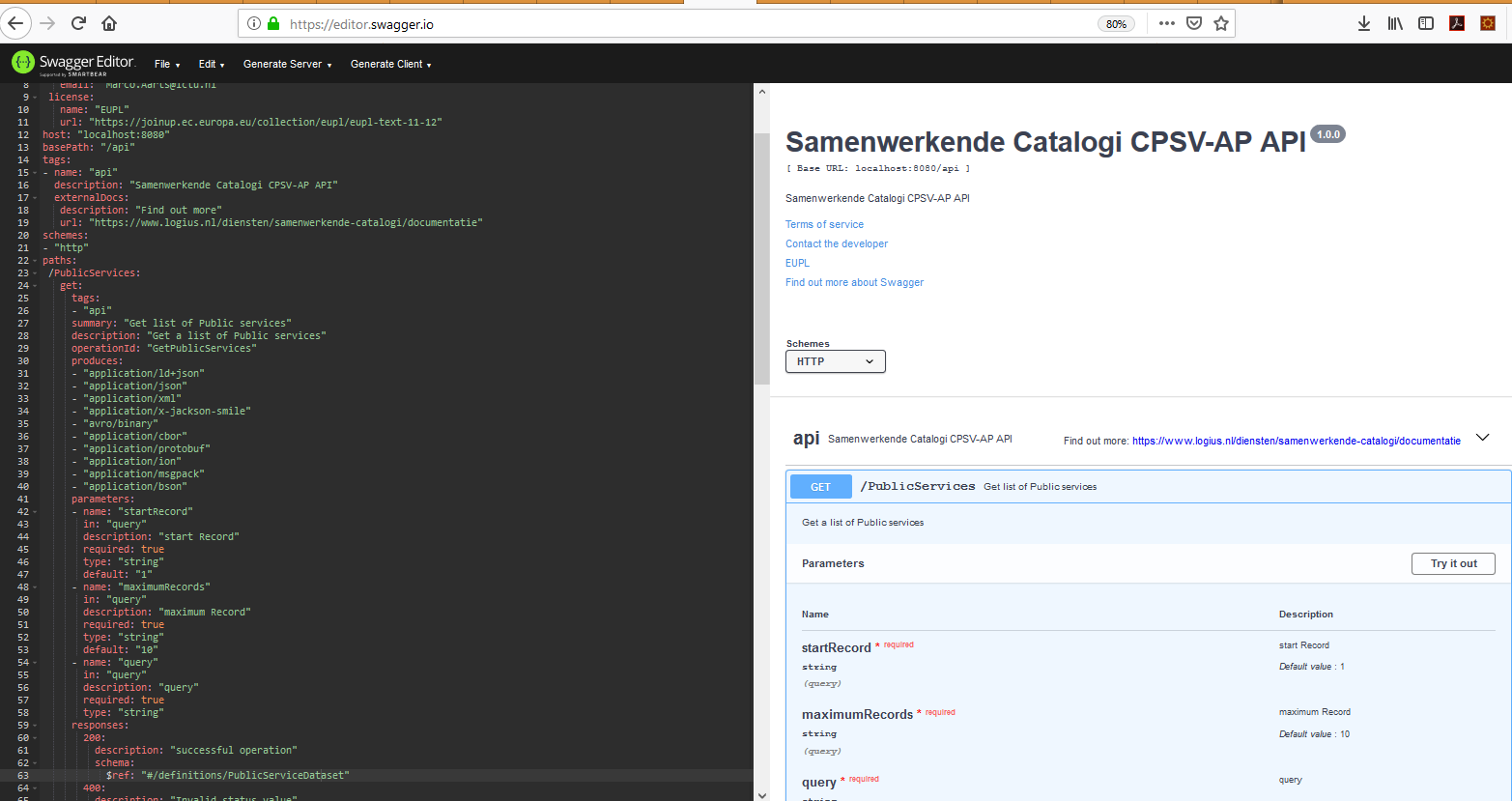


Figure 4: The online Swagger editor used to create the API contract

#### Formats

The CPSV-API supports content negotiation in order to provide different output formats via the HTTP Accept header by using the related mime types:

Table 2: Supported formats of the CPSV-AP API

|  |  |  |
| --- | --- | --- |
| **Format** | **Type** | **Mime type** |
| JSON-LD | Text | application/ld+json |
| XML | Text | application/xml |
| JSON | Text | application/json |
| SMILE | Binary | application/x-jackson-smile |
| AVRO | Binary | avro/binary |
| CBOR | Binary | application/cbor |
| Protobuf | Binary | application/protobuf |
| ION | Binary | application/ion |
| MSGPACK | Binary | application/msgpack |
| BSON | Binary | application/bson |

The API provides the output in JSON-LD format by default, so there is no need to add the HTTP Accept header for this format.

As it is possible to see, there are several supported binary formats, useful to improve the performance of the API and, accordingly to the tests conducted on the query, Avro and Protobuf format result the most performant in terms of output length:

Table 3: Binary format comparison

|  |  |  |
| --- | --- | --- |
| **Format** | **Byte length** | **GZIP compression** |
| AVRO | 38600 | 8527 |
| PROTOBUF | 38665 | 8573 |
| SMILE | 38788 | 8553 |
| ION | 38871 | 8721 |
| MSGPACK | 39471 | 8602 |
| CBOR | 39546 | 8622 |
| BSON | 40114 | 8748 |

## Component architecture

INSERT HERE A COMPONENT ARCHITECTURE

Describe:

1. Applicationcontext.xml
2. Binary Providers (GenericReport)
3. SRU Object – Response
4. JSON-LD Annotations

## Mapping spreadsheet

Attached below the mapping spreadsheet use to map CPSV-AP with the SRU data model

