CPSV-AP API for netherlands

Making the SRU API compliant with CPSV-AP

Table of Contents

[1. Introduction 2](#_Toc9502624)

[2. The SRU API 2](#_Toc9502625)

[3. The CPSV-AP Mapping 2](#_Toc9502626)

[4. API Architecture 3](#_Toc9502627)

[4.1 Input request and Output response 3](#_Toc9502628)

[4.2 The API Contract 5](#_Toc9502629)

[4.3 Performance 6](#_Toc9502630)

[4.4 Component architecture 7](#_Toc9502631)

[5. Development 8](#_Toc9502632)

[5.1 Source code structure 8](#_Toc9502633)

[5.2 Requirements 8](#_Toc9502634)

[5.3 Compiling and Deploying 8](#_Toc9502635)

[6. Mapping spreadsheet 8](#_Toc9502636)

## Introduction

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

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 samenwerkende catalogi[[1]](#footnote-1) is provided as an API which is based on the OWMS[[2]](#footnote-2), a standard adopted at the national level in the Netherlands. OWMS is described as an XML Schema.

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 input parameters:

Table 1: Parameters accepted by the SRU API

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

For example, users 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 three 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 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 version 2.2.1 since two properties (authority and productHTML) have cardinalities more relaxed than the CPSV-AP.

The reader can find more details in the Mapping spreadsheet.

## API Architecture

### 4.1 Input request and Output response

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

* startRecord
* maximumRecords
* query

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

* PublicServiceDataset

The CSPV-API passes the three input parameters to the SRU API and adds the other parameters. It then gets back the SRU Response and generates a PublicServiceDataset.

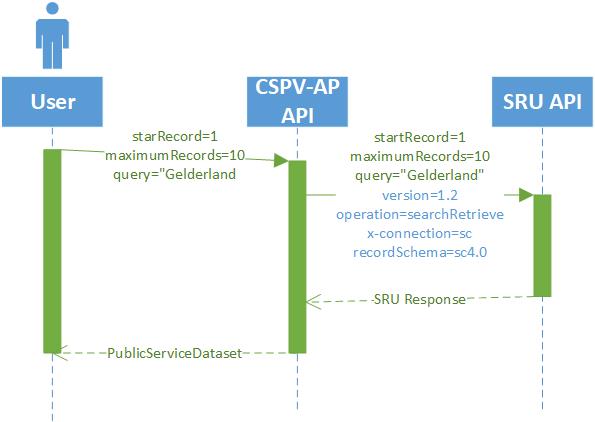


Figure 1: Sequence diagram of interaction between the CPSV-AP API and SRU API

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

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

While the PublicOrganization connected has 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, that displays, at the top, the PublicServiceDataset including, via the *@context* property, the CPSV-AP JSON-LD context which is reused to express linked data properties. Furthermore, the above example shows a PublicService with its properties (type, language, etc.).

Displaying the JSON-LD output as a graph, gives the following:

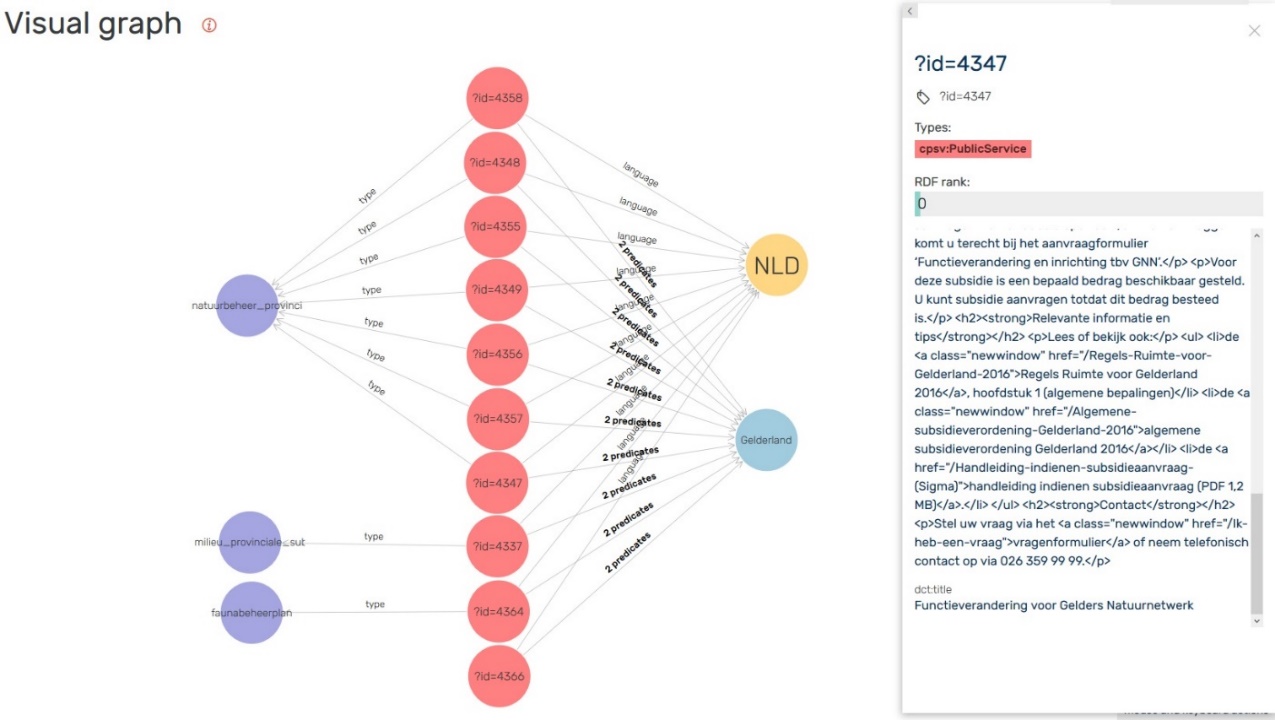


Figure 3: The output of the CPSV-API displayed as graph via GraphDB triple store

As can be seen in the above image, all public services have the same competent authority (Gelderland) as requested to the API via the query parameter.

### 4.2 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:

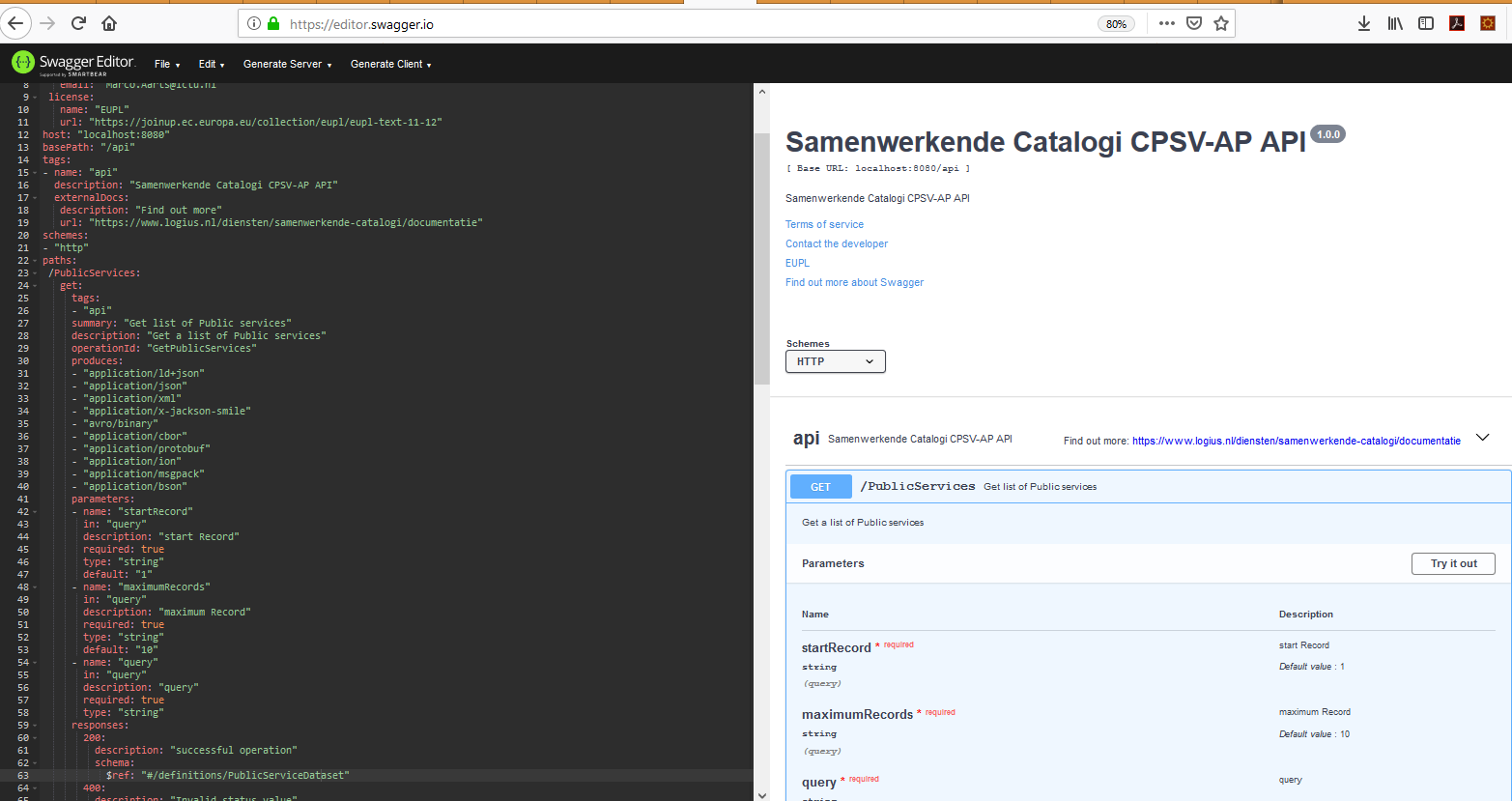


Figure 4: The online Swagger editor used to create the API contract

#### 4.2.1 Protocol

The only supported protocol is HTTP, leaving security aside from the scope of the project.

#### 4.2.2 Formats

As described in the contract, 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.

#### 4.2.3 Data models

The below data models are described within the API Contract which are those defined in the Input request and Output response section:

* PublicServiceDataset, as container of PublicService
* PublicService
* PublicOrganization
* Concept (Public Service type)
* Location (Public Service and Public Organization spatial)
* Language

### 4.3 Performance

As the user can choose to increase the number of public services to be provided the CPSV-API need to be performant.

Performance of the API can be given by:

* the output format chosen via the query;
* the caching mechanism established in the API;
* the zipping mechanism established in the API.

#### 4.3.1 Binary formats

As explained in the Formats section, there are several supported binary formats, which are useful to improve the performance of the API. According to the tests conducted on the API, Avro and Protobuf formats are 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 |

#### 4.3.2 Caching

Caching is enabled by means of the annotation in the API code (ApiApiServiceImpl.java) with a default of 10 minutes (600 seconds): @CacheControl("max-age=600").

As a result, the response of the CPSV-AP API includes the HTTP Header Cache-Control set to 600 and the response code changed to 304.

#### 4.3.3 Zipping

Zipping is enabled by default in the API, by leveraging two configuration properties:

1. threshold, set to 0 bytes, which means everything (bigger than 0) will be compressed;
2. Force, to force the zipping.

Such configuration can be found in the applicationContext.xml as seen in the Component architecture. As a result, the response of the CPSV-AP API includes the HTTP Header Content-encoding set to gzip.

#### 4.3.4 Etag

ETags are usually used for two reasons: caching and conditional requests. The implementation is a shallow one – the application calculates the ETag based on the response, which will save bandwidth but not server performance.

The ETag value can be thought of as a hash computed out of the bytes of the Response body. Because the service likely uses a cryptographic hash function, even the smallest modification of the body will drastically change the output and thus the value of the ETag.

The CPSV-AP API supports the ETag by declaring a filter within the web.xml file. Such filter generates an ETag value based on the content of the response. This ETag is compared to the If-None-Match header of the request. If these headers are equal, the response content is not sent, but rather a 304 "Not Modified" status instead.

### 4.4 Component architecture

The below image shows the component diagram.

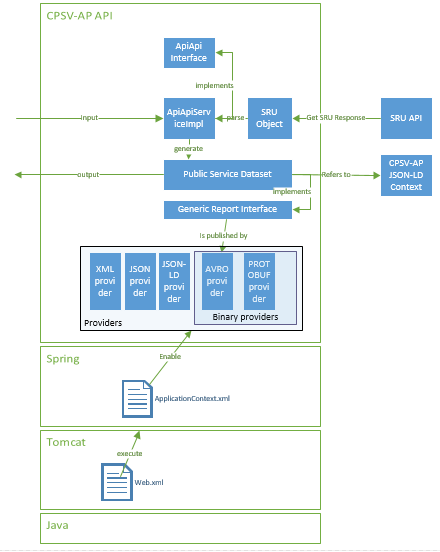


Figure 5: Component Diagram

The CPSV-AP API is based on the Apache CXF Framework and deployed on Apache Tomcat. In order to be deployed, a web.xml file, which specifies the base URL of the API and some filters (e.g. Etag), is required.

The CXF Framework is based on the Spring Framework. As such, to run, the API is required to have an ApplicationContext.xml which contains the list of providers (e.g. Avro, Protobuf, etc.). The applicationContext.xml is linked by the web.xml file and allows to generate dynamically the swagger json of the API (<http://localhost:8080/swagger-cxf-server-1.0.0/api/swagger.json>).

As explained in section “4.1 Input request and Output response”, the CPSV-AP API passes input parameters to the SRU API, which returns an SRU Response. The input parameters are described inside the ApiApi Interface (which is also responsible of formatting the output of the API). The ApiApi Interface is implemented by the ApiApiServiceImpl. The SRU Response is translated into one or more SRU Objects, which represent descriptions of various public services.

The CPSV-AP API transforms the SRU Object into a PublicService Dataset. Such object contains java annotations to refer to the CPSV-AP JSON-LD context which is included when the user requires a JSON-LD output.

The PublicService Dataset implements the GenericReport interface which is used by the binary providers.

## Development

### 5.1 Source code structure

In the below table, the reader can find the main folders and files included in the API.

Table 4: Source code structure

|  |  |
| --- | --- |
| **File / Folder** | **Description** |
| pom.xml | File needed by Maven to compile and package the API. |
| .swagger-codegen-ignore | File needed by the Maven Swagger codegen plugin to generate the model and the API. |
| /doc | Documentation about the API. |
| /src/gen/java/io/swagger/api | API generated by the Maven Swagger codegen plugin. |
| /src/gen/java/io/swagger/model | Model generated by the Maven Swagger codegen plugin. |
| /src/gen/java/cxf | Model of the SRU API generated by JAXB. |
| /src/main/java/cxf/SruObject.java | Representation of a public service extracted from the SRU Response. |
| /src/main/java/io.swagger.api/impl/ApiApiServiceImpl.java | Implementation of the API. |
| /src/main/java/io.swagger.api/\* | Gathers all the providers (Avro, Protobuf, etc.) used by the API. |
| /src/main/resources/wsdl/\* | Xml schema used to generate the model of the SRU API. |
| /src/main/resources/applicationContext.xml | Application Context used by the Spring Framework. |
| /src/main/resources/netherlandsapi.yaml | API contract (expressed as YAML) used by the Maven Codegen plugin. |
| /src/main/webapp/WEB-INF/web.xml | Deployment file of the API. |

### 5.2 Requirements

The main three requirements to compile and deploy the CSPV-AP API are:

1. JDK 8
2. Apache Maven
3. Apache Tomcat 8

One of the dependency described inside the “pom.xml” is the Jackson/json-ld module which has been modified to include the CPSV-AP remote context[[3]](#footnote-3).

### 5.3 Compiling and Deploying

As the CPSV-AP API is implemented in Java as Maven project, it is easy to compile the project by means of the maven goal “compile”.

The “compile” goal will trigger the previous goals like “generate-sources” used by the maven-swagger-codegen plugin.

The “package” goal will create the war file in the */target* folder. Such war file needs to be copied under the webapps folder of Apache Tomcat, the application server used for the development. When Tomcat is up and running, the war file expands in a folder and the API can be used.

## Testing

It has been tested that:

* the API provides the right output format depending on the accept header with the REST Client of Firefox;
* the Swagger file is correctly imported by SOAP UI;
* the JSON-LD output has been verified with the json-ld.org/playground/ website[[4]](#footnote-4) and with the validator[[5]](#footnote-5) which is able to retrieve the remote JSON-LD context.

## Mapping spreadsheet

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



1. <https://www.logius.nl/diensten/samenwerkende-catalogi/documentatie> [↑](#footnote-ref-1)
2. https://standaarden.overheid.nl/owms/terms [↑](#footnote-ref-2)
3. https://github.com/catalogue-of-services-isa/jackson-jsonld [↑](#footnote-ref-3)
4. https://json-ld.org/playground/ [↑](#footnote-ref-4)
5. http://rdfvalidator.mybluemix.net/ [↑](#footnote-ref-5)