VESPA service tutorial with intermediate metadata table

Work in Progress

This tutorial is still under development, but nearly finalized. Please try out and post any necessary comment! 🙂

EPN2020-RI

EUROPLANET2020 Research Infrastructure

H2020-INFRAIA-2014-2015

Grant agreement no: 654208

Document: VESPA-WP6-2-018-TP-v0.1(39)

VESPA service tutorial with intermediate metadata table

Date: 2020-03-04

Start date of project: 01 September 2015 Duration: 48 Months

Responsible WP Leader: Stéphane Erard

Project co-funded by the *European Union's Horizon 2020 research and innovation programme*		
Dissemination level		
PU	Public	☐
PP	Restricted to other programme participants (including the Commission Service)	☐
RE	Restricted to a group specified by the consortium (including the Commission Services)	☐
CO	Confidential, only for members of the consortium (excluding the Commission Services)	☐

Project Number	654208
Project Title	EPN2020 - RI
Project Duration	48 months: 01 September 2015 – 30 August 2019

Document Number	WP6-task2-018-v0.1
Delivery date	
Title of Document	VESPA service tutorial with intermediate metadata table
Contributing Work package (s)	WP6
Dissemination level	PU
Author (s)	Baptiste Cecconi

Abstract: This tutorial

Document history (to be deleted before submission to Commission)

Date	**Version**	**Editor**	**Change**	**Status**
31 Mar 2017	0.0	Baptiste Cecconi	Initial version	DRAFT
04 Apr 2017	0.1	Baptiste Cecconi	first complete version	DRAFT

Table of Contents

Introduction

This document presents the set up of an EPN-TAP service using an intermediate metadata table stored in CSV format. The CSV file is placed on the DaCHS server. It is then imported into the internal DaCHS database as described in the resource descriptor for your test service. This resource descriptor is using a csvGrammar script to load your metadata from the CSV file into the EPNcore parameters to be shared with EPN-TAP.

This tutorial implies that you have a running DaCHS server (see EPN-TAP Server Installation for VESPA Data Provider Tutorial).

Preparing the metadata table

In this tutorial the EPNcore metadata are stored into a CSV file (Comma Separated Values). This file must contain a header row with column names. Each row then contains the metadata for a data product that you distribute. It is recommended to use the actual EPNcore keywords as column names in the file, as well as to prepare your metadata directly in the correct form or units. However, some editing can be done in the resource descriptor (it is demonstrated in this tutorial).

The example metadata catalog is extracted from the APIS (Auroral Planetary Imaging and Spectroscopy) service. A series of six data products have been selected from this database. The metadata CSV file has been prepared and is available for download: apis_test.csv. This file contains several columns but not all the EPNcore keywords are present. We have only put the columns that will be present (not filled with NULL values) in the database. We present here the various columns:

- Product identification:
  - **granule_uid**: This is built from the obs_id, with an adequate suffix ("_x2d" for raw data, "_proc" for processed data, "_proc_pdf" for processed data in PDF)
  - **granule_gid**: This a human readable group name to build sets of similar products (same processing)
  - **obs_id**: This is the original observation id from HST.
- Target description:
  - **target_name**: "Saturn" for those products. Note the capital S, which is standard.
  - **target_class**: "planet", this value is from a predefined list.
  - **target_region**: "planetary aurorae" This field a free text, preferably from the Astronomy Thesaurus, or similar controlled dictionaries.
- Temporal description:
  - **time_min**: here is ISO time, we will transform it into julian day in the resource descriptor
  - **time_max**: same as above
  - **time_exp_min**: the exposure time is constant, so the time_exp_min and time_exp_max are equal.
  - **time_exp_max**: see previous comment
- Spectral description:
  - **spectral_range_min**: already converted to Hz (beware: minimun frequency is maximum wavelength)
  - **spectral_range_max**: already converted to Hz (beware: maximum frequency is minimun wavelength)
  - **spectral_sampling_step_min**: already converted to Hz
  - **spectral_sampling_step_max**: already converted to Hz
  - **spectral_resolution_min**: already converted to Hz
  - **spectral_resolution_max**: already converted to Hz
- Observation geometry:
  - **phase_min**: in degrees
  - **phase_max**: in degrees
- Instrument description:
  - **instrument_host_name**: "hst"
  - **instrument_name**: "STIS"
- Data product description
  - **dataproduct_type**: "im" stands for "image"
  - **measurement_type**: "phot.count;obs.image"
  - **processing_level**: 3 for raw counts and 4 for calibrated products
- Access information:
  - **access_url**: link to data product
  - **access_format**: MIME type of the dat product ("image/fits" for FITS file, "application/pdf" for PDF file)
  - **access_estsize**: size of the data product in kbytes.
  - **publisher**: This si the datacenter providing access to the data
  - **service_title**: "test", to be replaced by the short name of your service
  - **creation_date**: the creation data of your dat product (can be the acquisition time)
  - **modification_date**: the last modification date of your data product (useful for checking if data have been updated or reprocessed)
  - **thumbnail_url**: a link to a small image containing an overview of the data content.

You may have other columns for your specific case. Before going to the next step, identify which optional columns (see EPNcore v2 specification) you need and write down the list. In the case of the example based on APIS data, the optional columns are:

```
target_region access_url access_format access_estsize publisher thumbnail_url
```

For extra columns that are not present in the current list of keywords (neither in the mandatory, nor in the optional section), you will have to declare them manually in the ressource descriptor. For this step you need to define the following elements (for each extra column):

- **name**. Must be lowercase, no space or special characters (only simple ASCII + underscore). Should not be too short or simple, as it may interfere with other existing names in other services
- **type**. This is the data type: usually one of "float", "integer" or "text". Other more complex types are possible, see documentation.
- **ucd**. Defines the quantity present in the column. Must be a valid Unified Content Descriptor, as defined by the IVOA. Ask for help or extra UCD request to VESPA helpdesk.
- **description**. Must be a short be adequate description of the quantity present in this column (think as an external user).

We also would like to be included in discussions about new columns, in order to make sure that there are no planned extension of EPNcore that would be conflicting with your proposed extra keywords.

Configuring the Resource Descriptor

The configuration of the service is done in a file called a resource descriptor, usually named "`q.rd`" in DaCHS. The service must be prepared as follows.

The first step is to decide on the short name of your service. In this tutorial, we opt for "**test**". The first step is to create the service directory in DaCHS.

Log on the DaCHS server, with your own user. In order to setup the service properly, you have to switch to the `dachsroot` user. You can do this from your user account, issuing the two following commands:

```
sudo -s
su dachsroot
```

The first line is switching to root account (your user password may have to be provided), and the second is switching to `dachsroot`.

Logged as `dachsroot` , create a new directory for your new service in the `/var/gavo/inputs` directory:

```
mkdir /var/gavo/inputs/test
```

The name of the directory must be the short name of your service. In that directory, you have to create a `data/` subdirectory. You can then copy your CSV metadata file in that directory:

```
mkdir /var/gavo/inputs/test/data
cd /var/gavo/inputs/test/data
wget https://voparis-confluence.obspm.fr/download/attachments/10944992/apis_test.csv?
version=3&api=v2&download=true
mv apis_test.csv metadata.csv
```

In the last command, we propose to copy and rename the example CSV file (that you have to copy to your user home directory before this step).

The resource description file q.rd is fully describing your service. It contains metadata concerning the service, that are used to fill out the registry record for your service. It also tells DaCHS how to build the table in the internal database. It finally tells DaCHS how to map from the internal database to the EPN-TAP interface. The example resource descriptor file q.rd hats be placed in the `/var/gavo/inputs/test/` directory.

```
cd /var/gavo/inputs/test
wget https://voparis-confluence.obspm.fr/download/attachments/10944992/q.rd?
version=1&api=v2&download=true
```

The resource descriptor can be validated:

```
cd /var/gavo/inputs/test
gavo val q.rd
```

If the file is valid, you won't have errors. A short message output states "`q -- OK`". You can then test the import in "dump" mode:

```
cd /var/gavo/inputs/test
gavo imp -d q.rd
```

This will throw a lot of output: each row from the CSV file (your metadata source) is interpreted and the result displayed. The real import step is done with the "modify" mode:

```
cd /var/gavo/inputs/test
gavo imp -m q.rd
```

At this stage, your service should be up and running.

Testing the service

You can test your service either from your DaCHS instance web interface (http://127.0.0.1:8000/adql in case of the local installation tutorial) , going to the ADQL query page, and issuing:

```
select * from test.epn_core
```

You can also use the VESPA main query interface (using the "custom resource" tab) to access your service if it is available on the internet, or on a local instance, if there is one available for you.