Publications Office of the European Union

Initiative for digital transformation in the Metadata and Reference Data Sector of the Publications Office of the European Union

Installation guide for the asset publishing workflow services

Disclaimer

This report was prepared for the Publications Office of the European Union by Infeurope.

Document metadata

Reference	WP 1.3.8: Installation guide for the asset publishing workflow services
Corporate Author	Publications Office of the European Union
Author	Eugeniu Costetchi
Reviewers	Denis Dechandon and Willem Van Gemert
Contractor	Infeurope S.A.
Framework contract	10688/35368
Work package	WP 1.3.8
Delivery date	06 November 2020
Suggested readers	technical staff, system administrators, enterprise architects, software developers

Abstract

This document provides technical guidance on how to install and configure the suite of micro-services and applications necessary for the asset metadata lifecycle process at the Standardisation Unit at the Publications Office of the European Union.

Contents

1 Introduction 5
2 Scope 5
3 Target audience 5
4 Technology background 6
5 Requirements 6
6 Installation 6
7 Configuration 8
7.1 RDF differ 9
7.2 Add a new application profile template 9
7.3 Configure and read logs 9
8 API documentation 10
8.1 Get application profiles names and their template variations . 10
8.2 List the existent datasets 11
8.3 Get specific dataset 12
8.4 Delete specific dataset 14
8.5 Create a diff 14
8.6 Create a report 15
8.7 Get report 16
8.8 List active tasks 17
8.9 Get specific task status 18
8.10 Stop task execution 19
9 SPARQL Queries 20
9.1 Naming conventions 20
9.2 Structure 21
9.3 Example of diffing query 24

1 Introduction

The Standardisation Unit (SU) at the Publications Office of the European Union (OP) is engaged in a digital transformation process oriented towards semantic technologies. In [1] is described a working definition of the architectural stance and design decisions that are to be adopted for the asset publication life-cycle process. The report describes the baseline (current) solution and the (new) target solution for the asset publication workflow that is part of the life-cycle process.

The software components building up the target publication workflow solution have been packaged as into a suite of interconnected services.

This document describes the installation and configuration procedures along with stating the scope, and target audience.

2 Scope

This document aims at covering the installation and configuration instructions for the suite of the following software services:

1. RDF differ
2. Celery worker

3 Target audience

The target audience for this document comprises the following groups and stakeholders:

- Technical staff in charge of operating workflow components
- System administrators
- Enterprise architects and data governance specialists
- Documentalists involved in the reference data life-cycle
- Developers in charge of workflow and component implementation
- Third parties using the SU services and data

4 Technology background

Infrastructure and deployment configuration rely on services deployed on a CentOS system.

5 Requirements

CentOS system with python version 3.8 a fuseki and redis service setup and available.

There is a range of ports that must be available on the host machine as they will be bound to by different services. Although the system administrator may choose to change them by changing the values in of specific environment variables. The inventory of pre-configured ports is provided in Table 1.

Service name	HTTP port UI	HTTP port API
RDF differ	8030	4030
dedicated Fuseki		3030
redis		6379

Table 1: Port usage inventory

The minimal hardware requirements are as follows

1. CPU: ???
2. RAM: ???
3. SDD system: ???
4. SDD data: ???

6 Installation

In order to run the services it is necessary to have the fuseki and redis services already setup and running with the appropriate ports and addresses configured in the environment variable file.

In case you are using Debian-like OS such as Ubuntu, you may simply run the following Bash commands to install and set the appropriate permissions.

Next, copy the rdf differ zip on the system you intend to run it and unzip it.

Then change directory into the *project* folder. Makefile commands to start and stop services will be available.

To start the services using Makefile

```
make install-python-dependencies
make run-api
make run-ui
```

To stop the services using Makefile

```
make stop-gunicorn
```

To start services without Makefile commands

```
set -o allexport; source bash/.env; set +o allexport

python3.8 -m venv env
source env/bin/activate
pip install -r requirements/prod.txt
```

then start the services

```
set -o allexport; source bash/.env; set +o allexport

source env/bin/activate
celery -A rdf_differ.adapters.celery.celery_worker worker --
   loglevel ${RDF_DIFFER_LOG_LEVEL} --logfile ${
   RDF_DIFFER_CELERY_LOGS} --detach
gunicorn --timeout ${RDF_DIFFER_GUNICORN_TIMEOUT} --workers ${
   RDF_DIFFER_GUNICORN_API_WORKERS} --bind 0.0.0.0:${
   RDF_DIFFER_API_PORT} --reload rdf_differ.entrypoints.api.run:app
    --log-file ${RDF_DIFFER_API_LOGS} --log-level ${
   RDF_DIFFER_LOG_LEVEL} --daemon
gunicorn --timeout ${RDF_DIFFER_GUNICORN_TIMEOUT} --workers ${
   RDF_DIFFER_GUNICORN_UI_WORKERS} --bind 0.0.0.0:${
   RDF_DIFFER_UI_PORT} --reload rdf_differ.entrypoints.ui.run:app
   --log-file ${RDF_DIFFER_UI_LOGS} --log-level ${
   RDF_DIFFER_LOG_LEVEL} --daemon
```

To stop the services run

```
source env/bin/activate

celery -A rdf_differ.adapters.celery.celery_worker control shutdown
pkill -f gunicorn
```

7 Configuration

The deployment suite of micro-services is defined docker-compose.yml file. At deployment and at runtime, the service configurations are provided through OS environment variables available in the *.env* file. The role of the .env file is to enable the system administrators to easily change default configurations as necessary in the context of their environment.

The suite of micro-services is built, started and shut down via Makefile commands.

In order to avoid hard coding parameters, they are defined externally in the *.env*. Having them in a single file makes much more sense and it is more pragmatic, as you can see and manage all parameters in one place, add the file to the version control system (the contents of the file will evolve and be in sync with the actual code) and have different files for different environments.

The file is named *.env* and contains all of the parameters that you want to be able to change and that you need to build and run the defined containers.

Having the parameters in an *.env* file is very useful in a multitude of scenarios, where you would want to have different configurations for different environments where you might want to deploy. As a more specific example, consider a continuous delivery pipeline and the URLs and ports you want your containers to bind (or to connect) to. You thus can easily have two *.env* files, one named *test.env* and one named *acceptance.env*. Each file would have the same declared variables, but with different values for each of the continuous delivery pipeline stage where it's being deployed. The benefit is that you deploy and test/use the same containers/artifacts and are able to configure them, on the spot, according to the environment that they are integrated with.

This section describes the important configurations options available for each of the services.

7.1 RDF differ

The RDF differ application exposes an API and an UI and depends on a dedicated triple store. The RDF diff API is the core service providing the RDF diffing functionality. The URL and port are described below, as well as the request timeout:

Description	Value	Associated variable
Service URL	http://localhost	RDF_DIFFER_API_LOCATION
Service API port	4030	RDF_DIFFER_API_PORT
Is in debug mode	True	RDF_DIFFER_DEBUG
Service UI port	8030	RDF_DIFFER_UI_PORT
Web server worker process timeout	1200	RDF_DIFFER_GUNICORN_TIMEOUT

Table 2: RDF differ configurations

7.2 Add a new application profile template

The default application profile template is the diff report template that resides in `resource/templates/diff_report` folder. For adding a new application profile create a new folder under `resource/templates` with the name of your new application profile and following the structure explained below. Folder structure needed for adding a new application profile:

```
templates/
 diff_report/
  new_application_profile/
   config.json    <--- configuration file
   queries/       <--- contains SPARQL queries
    query1.rq
    query2.rq
   template_variants/
    html/         <--- contains files for a html template
    json/         <--- contains files for a json template
```

7.3 Configure and read logs

Every service provided by the rdf differ has it's own log history and is configurable through the aforementioned *.env* file. The current configuration accepts a relative path to where the logs to be written *logs/api.log*, for example.

API log example

```
[2021-12-01 15:54:39 +0000] [7] [INFO] Starting gunicorn 20.1.0
[2021-12-01 15:54:39 +0000] [7] [DEBUG] Arbiter booted
[2021-12-01 15:54:39 +0000] [7] [INFO] Listening at: http
   ://0.0.0.0:4030 (7)
[2021-12-01 15:54:39 +0000] [7] [INFO] Using worker: sync
[2021-12-01 15:54:39 +0000] [9] [INFO] Booting worker with pid: 9
[2021-12-01 15:54:39 +0000] [10] [INFO] Booting worker with pid: 10
[2021-12-01 15:54:39 +0000] [7] [DEBUG] 2 workers
[2021-12-01 15:55:13 +0000] [9] [DEBUG] GET /diffs
[2021-12-01 15:55:13 +0000] [9] [DEBUG] start get diffs endpoint
[2021-12-01 15:55:13 +0000] [9] [DEBUG] finish get diffs endpoint
```

UI log example

```
[2021-12-01 15:55:21 +0000] [10] [DEBUG] GET /tasks
[2021-12-01 15:55:21 +0000] [10] [DEBUG] request active tasks view
[2021-12-01 15:55:22 +0000] [10] [DEBUG] render active tasks view
```

The RDF differ application uses the following environment variables to define logs location:

Description	Value	Associated variable
API logs	logs/api.log	RDF_DIFFER_API_LOGS
UI logs	logs/ui.log	RDF_DIFFER_UI_LOGS
Celery logs	logs/celery.log	RDF_DIFFER_CELERY_LOGS

Table 3: RDF differ log configurations

8 API documentation

8.1 Get application profiles names and their template variations

URL	ACTION
`/aps`	`GET`

Response

200


```
[
  {
    "application_profile": "skos-core-en-only",
    "template_variations": [
      "html",
      "json"
    ]
  },
  {
    "application_profile": "skos-core-lang-fallback",
    "template_variations": [
      "html",
      "json"
    ]
  }
]
```


8.2 List the existent datasets

URL	ACTION
`/diffs`	`GET`

Response

200


```
[
  {
      "current_version_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version/new",
      "dataset_description": null,
      "dataset_id": "/diff18H35CGpD",
      "dataset_uri": "http://publications.europa.eu/resource/
         authority/data-theme/",
      "dataset_versions": [
          "new1",
          "old1"
      ],
      "diff_date": null,
```

```
      "new_version_id": "new",
      "old_version_id": "old",
      "query_url": "http://fuseki:3030/diff18H35CGpD/sparql",
      "version_history_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version",
      "version_named_graphs": [
          "http://publications.europa.eu/resource/authority/data-
             theme/version/new",
          "http://publications.europa.eu/resource/authority/data-
             theme/version/old"
      ]
  },
  {
      "current_version_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version/new",
      "dataset_description": null,
      "dataset_id": "/diff2F4ZLMgNu",
      "dataset_uri": "http://publications.europa.eu/resource/
         authority/data-theme/",
      "dataset_versions": [
          "new1",
          "old1"
      ],
      "diff_date": null,
      "new_version_id": "new",
      "old_version_id": "old",
      "query_url": "http://fuseki:3030/diff2F4ZLMgNu/sparql",
      "version_history_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version",
      "version_named_graphs": [
          "http://publications.europa.eu/resource/authority/data-
             theme/version/new",
          "http://publications.europa.eu/resource/authority/data-
             theme/version/old"
      ]
  }
]
```


8.3 Get specific dataset

URL	ACTION
/diffs/{dataset_id}	GET

Parameters

Name	Description
dataset_id	dataset unique name

Response

200


```
{
  "current_version_graph": "http://publications.europa.eu/resource/
     authority/data-theme/version/new",
  "dataset_description": null,
  "dataset_id": "/diff18H35CGpD",
  "dataset_uri": "http://publications.europa.eu/resource/authority/
     data-theme/",
  "dataset_versions": [
    "new1",
    "old1"
  ],
  "diff_date": null,
  "new_version_id": "new",
  "old_version_id": "old",
  "query_url": "http://fuseki:3030/diff18H35CGpD/sparql",
  "version_history_graph": "http://publications.europa.eu/resource/
     authority/data-theme/version",
  "version_named_graphs": [
    "http://publications.europa.eu/resource/authority/data-theme/
       version/new",
    "http://publications.europa.eu/resource/authority/data-theme/
       version/old"
  ]
}
```


404


```
{
  "detail": "<datasetname> does not exist.",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

8.4 Delete specific dataset

URL	ACTION
`/diffs/{dataset_id}`	`DELETE`

Parameters

Name	Description
`dataset_id`	dataset unique name

Response

200

```
"<datasetname> deleted successfully."
```

404

```
{
  "detail": "<datasetname> does not exist.",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

8.5 Create a diff

URL	ACTION
`/diffs`	`POST`

Body

multipart/form-data

Name	Required	Type	Description
dataset_id	true	string	The dataset identifier. This should be short alphanumeric string uniquely identifying the dataset.
dataset_description	true	string	The dataset description. This is a free text description fo the dataset.
dataset_uri	true	string	The dataset URI. For SKOS datasets this is usually the ConceptSchema URI.
old_version_id	true	string	Identifier for the older version of the dataset.
new_version_id	true	string	Identifier for the newer version of the dataset.
old_version_file_content	true	file	The content of the old version file.
new_version_file_content	true	file	The content of the new version file.
new_version_id	true	string	Identifier for the newer version of the dataset.

Response

200

```
{
  "dataset_name": "diff2F4ZLMgNu",
  "task_id": "cee03499-41b2-41e4-ae75-95b9383eea0c"
}
```

8.6 Create a report

URL	ACTION
/diffs/reports	POST

Body

application/json

Name	Required	Type	Description
`dataset_id`	true	`string`	The dataset identifier. This should be short alphanumeric string uniquely identifying the dataset.
`application_profile`	true	`string`	The application profile identifier
`template_type`	true	`string`	The template type identifier
`rebuild`	false	`string`	Flag to signal rebuilding the report even if already exists. ("true" or "false")

Response

200

```
{
  "application_profile": "skos-core-en-only",
  "task_id": "3cf43787-18e8-4927-aa5f-198da6b8bba2"
}
```

8.7 Get report

URL	ACTION
`/diffs/report`	`GET`

Parameters

Name	Description
`dataset_id`	dataset unique name

Table 15 continued from previous page

`application_profile`	The application profile identifier
`template_type`	The template type identifier

Response

200

Report file in specified format

8.8 List active tasks

URL	ACTION
`/tasks/active`	`GET`

Response

200

```
[
  {
    "acknowledged": true,
    "args": [
        "diff2F4ZLMgNu",
        "skos-core-en-only",
        "html",
        "/usr/src/app/reports",
        "/usr/src/app/resources/templates/skos-core-en-only/
           template_variants/html",
        {
            ...
        },
        {
            "available_reports": [
                {
                    "application_profile": "diff_report",
                    "template_variations": [
                        "html"
                    ]
                }
```

```
            ],
            "current_version_graph": "http://publications.europa.eu
               /resource/authority/data-theme/version/new1",
            "dataset_description": null,
            "dataset_id": "diff2F4ZLMgNu",
            "dataset_uri": "http://publications.europa.eu/resource/
               authority/data-theme/",
            "dataset_versions": [
                "new1",
                "old1"
            ],
            "diff_date": null,
            "new_version_id": "old1",
            "old_version_id": "new1",
            "query_url": "http://fuseki:3030/diff2F4ZLMgNu/sparql",
            "version_history_graph": "http://publications.europa.eu
               /resource/authority/data-theme/version",
            "version_named_graphs": [
                "http://publications.europa.eu/resource/authority/
                   data-theme/version/new1",
                "http://publications.europa.eu/resource/authority/
                   data-theme/version/old1"
            ]
        }
    ],
    "delivery_info": {
        "exchange": "",
        "priority": 0,
        "redelivered": null,
        "routing_key": "celery"
    },
    "hostname": "celery@e5d79cc45ba2",
    "id": "3cf43787-18e8-4927-aa5f-198da6b8bba2",
    "kwargs": {},
    "name": "generate_report",
    "time_start": 1638810064.448214,
    "type": "generate_report",
    "worker_pid": 25
  }
]
```


8.9 Get specific task status

URL	ACTION
`/tasks/{task_id}`	`GET`

Parameters

Name	Description
`task_id`	task unique id

Response

200

```
{
  "task_id": "6ce77efc-667e-4cf4-bcec-cc7870fcc2db",
  "task_result": true,
  "task_status": "SUCCESS"
}
```

8.10 Stop task execution

URL	ACTION
`/tasks/{task_id}`	`DELETE`

Parameters

Name	Description
`task_id`	task unique id

Response

200

```
{
  "message": "task 3cf43787-18e8-4927-aa5f-198da6b8bba2 set for
     revoking."
}
```

406

```
{
  "detail": "task already finished executing or does not exist",
  "status": 406,
  "title": "Not Acceptable",
  "type": "about:blank"
}
```

9 SPARQL Queries

9.1 Naming conventions

Operations

Looking at patterns of change likely to occur in the context of maintaining SKOS vocabularies or to find in a diffing context we've identified 5 change types and mapped them for easy referencing as follows:

- Addition → Added
- Deletion → Deleted
- Value update → Updated
- Movement (cross property) → Changed
- Movement (cross instance) → Moved

Query file name

In order to name a query file that will represent what is the query for ,but in the same time to be easy to read, the file name will be constructed from five parts. These are operation, rdf type, class name, property name and object property name. All

names are only the local segment of the Qname (compressed URI) provided and the rdf types are instance, property and reified (reified property).

File name examples:

Structure → `operation_rdfType_className`

File name: added_instance_collection

Structure → `operation_rdfType_className_propertyName`

File name: changed_property_concept_broader

9.2 Structure

In this project the SPARQL query is constructed from five parts that will be explained below.

Prefixes section

Prefixes section are declared in here before the select statement of the query. It can have as many possible prefixes and values as the query will use only the ones that it needs.

Query variables

A SPARQL query file could sometimes be long and hard to read. To improve readability and minimize use of hidden variables, the query parameters should express in some manner what is the query used for and also to have the same values in the entire query. For easy referencing a change from the SPARQL query parameters, a good option is to add a prefix to the parameters that will have the value changed in the diffing context. To avoid pollutions of variables names the convention will add prefix in front of the variables that is changing in the diffing context by using the query. This can only be old or new (i.e ?oldInstance ?newInstance).

Version history graph block

This block will remain unchanged for all diffing queries as it's defining the graphs that are used later in the query. As a mention there is only one part that can change here and that is the value injection for the query which will be present in

the next section. The logic of the query is based on this four graphs that are built here. We can see below that we are going to have access to newVersionGraph, oldVersionGraph, insertionsGraph and deletionsGraph, so we can filter our data.

```
GRAPH ?versionHistoryGraph {

  # parameters
  VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class) {
      ( undef
        undef
        undef
        skos:Concept
      )
  }
  # get the current and the previous version as default versions
  ?versionset dsv:currentVersionRecord/xhv:prev/dc:identifier ?
     previousVersion .
  ?versionset dsv:currentVersionRecord/dc:identifier ?latestVersion
      .
  # select the versions to actually use
  BIND(coalesce(?oldVersion, ?previousVersion) AS ?
     oldVersionSelected)
  BIND(coalesce(?newVersion, ?latestVersion) AS ?newVersionSelected
     )
  # get the delta and via that the relevant graphs
  ?delta a sh:SchemeDelta ;
    sh:deltaFrom/dc:identifier ?oldVersionSelected ;
    sh:deltaTo/dc:identifier ?newVersionSelected ;
    sh:deltaFrom/sh:usingNamedGraph/sd:name ?oldVersionGraph ;
    sh:deltaTo/sh:usingNamedGraph/sd:name ?newVersionGraph .
  ?insertions a sh:SchemeDeltaInsertions ;
    dct:isPartOf ?delta ;
    sh:usingNamedGraph/sd:name ?insertionsGraph .
  ?deletions a sh:SchemeDeltaDeletions ;
    dct:isPartOf ?delta ;
    sh:usingNamedGraph/sd:name ?deletionsGraph .
}
```

Value injection

As mentioned in the previous section there is a value injection block where we can assign values to variables that are going to be used in the query logic.

```
# parameters
VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class) {
    ( undef
      undef
```

```
      undef
      skos:Concept
    )
}
```

Query logic

This part of the query is filtering the data by looking for triples in the graphs made available in the version history graph block. The graphs are made available through the delta generated (see Versions and Deltas as Named Graphs) by the diffing process, and they are as follows:

- oldVersionGraph - contains triples existing in the old version file
- newVersionGraph - contains triples existing in the new version file
- insertionsGraph - contains added triples to the old version file
- deletionsGraph - contains triples deleted from the old version file

As an example, will want to look for new instances that are of a certain class and to achieve this will want to look into the insertions graph.

```
GRAPH ?insertionsGraph {
  ?instance a ?class .

  optional {
    ?instance skos:prefLabel ?prefLabelEn .
    FILTER (lang(?prefLabelEn) = "en")
  }
}
```

The query logic can continue to filter the results by verifying existence of triples in other available graphs. This can be done by using another graph block as showed below.

```
# ... and the instance must not exist in the old version
FILTER NOT EXISTS {
  GRAPH ?oldVersionGraph {
    ?instance ?p [] .
  }
}
```

9.3 Example of diffing query

Building a query to get all added skos:altLabel property per instance of a certain class.

Prefixes section

```
# basic namespaces
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
.
.
.
# versioning namespaces
PREFIX dsv: <http://purl.org/iso25964/DataSet/Versioning#>
PREFIX sd: <http://www.w3.org/ns/sparql-service-description#>
PREFIX sh: <http://purl.org/skos-history/>
PREFIX xhv: <http://www.w3.org/1999/xhtml/vocab#>
```

Query variables

For the results and the query itself to be easy to read we need to choose good variable names and make sure the variables will bring the expected result format. In this case we want to see instance URI, instance label, property and value of the property.

```
SELECT DISTINCT ?instance ?prefLabel ?property ?value
WHERE {
```

Version history graph block and value injection

For the results and the query itself to be easy to read we need to choose good variable names and make sure the variables will bring the expected result format. In this case we want to see instance URI, instance label, property and value of the property.

```
GRAPH ?versionHistoryGraph {
  # defining values for our variables that are we going to use in
     the query (instance class and property)
  VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class ?
     property) {
    ( undef
      undef
      undef
```

```
      skos:Concept
      skos:altLabel
    )
  }
  # get the current and the previous version as default versions
  ?versionset dsv:currentVersionRecord/xhv:prev/dc:identifier ?
     previousVersion .
  ?versionset dsv:currentVersionRecord/dc:identifier ?latestVersion
      .
  # select the versions to actually use
  BIND(coalesce(?oldVersion, ?previousVersion) AS ?
     oldVersionSelected)
  BIND(coalesce(?newVersion, ?latestVersion) AS ?newVersionSelected
     )
  # get the delta and via that the relevant graphs
  ?delta a sh:SchemeDelta ;
    sh:deltaFrom/dc:identifier ?oldVersionSelected ;
    sh:deltaTo/dc:identifier ?newVersionSelected ;
    sh:deltaFrom/sh:usingNamedGraph/sd:name ?oldVersionGraph ;
    sh:deltaTo/sh:usingNamedGraph/sd:name ?newVersionGraph ;
    dct:hasPart ?insertions ;
    dct:hasPart ?deletions .
  ?deletions a sh:SchemeDeltaDeletions ;
    sh:usingNamedGraph/sd:name ?deletionsGraph .
  ?insertions a sh:SchemeDeltaInsertions ;
    sh:usingNamedGraph/sd:name ?insertionsGraph .
}
```

Query logic

In this part we need to filter the results to get only the instances that had the skos:altLabel property added. As a starting point we will look in the insertions graph to get all inserted skos:altLabel properties for all instances. For this to be a true addition and not a change or a movement operation we need to make sure that the property was not attached to some other instance before and to do this will look into the deletions graph and old version graph. After filtering the result we can get all the values that are needed from the new version graph.

```
# get inserted properties for instances
GRAPH ?insertionsGraph {
    ?instance ?property [] .
  }
  # ... which were not attached to some (other) instance before
  FILTER NOT EXISTS {
    GRAPH ?deletionsGraph {
```

```
      ?instance ?property [] .
    }
  }
  FILTER NOT EXISTS {
    GRAPH ?oldVersionGraph {
      [] ?property ?value .
    }
  }
  # get instances with those property values
  GRAPH ?newVersionGraph {
    ?instance a ?class .
    ?instance ?property ?value .

    optional {
      ?instance skos:prefLabel ?prefLabel .
      FILTER (lang(?prefLabelEn) = "en")
    }
  }
}
```

References

[1] E. Costetchi. Asset publication lifecycle architecture. Recommendation, Publications Office of the European Union, September 2020.