Simulation Data Access Layer Version 1.0

IVOA Working Draft 27 may 2015

Working group
DAL
This version
http://www.ivoa.net/documents/simdal/27 may 2015
Latest version
http://www.ivoa.net/documents/simdal
Previous versions
This is the first public release
Author(s)
David Languignon, Franck Le Petit, Gerard Lemson, Marco Molinaro, Carlos Rodrigo, Hervé Wozniak
Editor(s)
David Languignon, Franck Le Petit

Abstract

Status of This Document

Contents

Acknowledgments

1 Introduction

The Simulation Data Access Protocol (SimDAL) defines a set of functions (an API) to discover simulations and numerical models and to access data extracted from these simulations. Many notions in this document refer to classes of the Simulation Data Model (SimDM).

This document is divided in three parts corresponding to the three components of the SimDAL standard: 1) simulations repository, 2) simulations discovery and 3) raw data access.
As explained below, each part can be implemented separately. A publisher willing to publish theoretical data can do it by providing the API for only the data discovery and/or the raw data access parts.
Also, some components like the SimDAL Repositories may benefit from few centralized implementations by some Data Centers. They play the role of fine grain registries for simulations.

1.1 Role within the VO Architecture

2 Overview

2.1 SimDAL

The SimDAL layer places itself at the interface between end-users and services publishing theoretical data. It's intended to fulfil the following core use cases:

- **UC1:** Discover simulations projects dealing with one's research interest in a repository of simulations projects, then find associated services (conforming to the SimDAL API or not).

  Example: Discover services publishing structures of interstellar clouds.

- **UC2:** Find particular models producing quantities in a defined range (for exemple to fit them to real observations) in a grid of models.

  Example: In a service publishing synthetic stellar spectra, find models of O 9 stars.

- **UC3:** Access simulation raw dataset or subset of it (cutout)

  Example: Download a data cube of a simulation of large scale structures formation.
  Example: Download a theoretical spectrum in a wavelength range.

To fulfil these core use cases, three components are defined.

The first one is the **SimDAL Repository**. SimDAL repositories are used to store metadata that describe codes, projects and services. They can be seen as the traditional registries of the VO excepted that they contain finer grain informations than registries do and these informations are described following the Simulation Data Model. Simulation Repositories are used by end-users to discover services publishing theoretical data. SimDAL Repositories are not intended to be implemented everywhere. A team who wishes to set up a service to publish simulations does not have to implement a SimDAL Repository. Instead, it must provide a description of their service and its data to an existing SimDAL Repository so that it can be discovered.

The second component is the **SimDAL Search**. Published simulations and numerical models are characterised by a set of metadata. In most cases, these metadata are the values of the input parameters and / or other quantities that characterise the published numerical models / simulations (for example some computed quantities or some statistics on the results).

Finally the third component is the **SimDAL Data Access**. Numerical codes produce different kind of outputs. That can be data cubes as those produced by N-body or MHD simulation codes. Those cubes are generally stored in specific binary format (HDF5, VTK, ...). That can be synthetic spectra stored in FITS or VO-Table format. Other ones can produce catalogs that can be stored on a filesystem in ASCII format or in relational databases. Thus, code outputs are very heterogeneous. The SimDAL Data Access API defines how to interact with such data to download all or a part of them. Because there is no standard data format for numerical code outputs, SimDAL standard does not define a standard format. Obviously, it is recommended to provide data in VO-compatible format whenever possible (VO-Table, FITS) so that they can be manipulated in VO tools.

These three parts are independent. Most of data publishers should only implement the Data Access part and, if required, the SimDAL Search part. In general, the identifiers of the datasets exposed by the Data Access API are found after searching in a service implementing the SimDAL Search API.

2.2 SimDAL and the Simulation Data Model

To share descriptions of codes and their results between scientific teams which produce data, publishers who develop and maintain diffusion systems and clients, one has to rely on a standard description format. This standard format is the XML serialization of the Simulation Data Model (SimDM). As a reminder, key elements of SimDM and how they are related are summarized below:

```
Input         --->         Protocol              --->  OutputDataObjectType
Parameters            (the simulation code)            or TargetObjectType

    ^                       ^
    |                       |
    |    defined for        | conforms
    |       each            | to


Parameter     --->       Experiment              --->   OutputDataset of
Settings             (numerical model run)              objects of type
                                                        OutputDataObjectType
                            |
                            |                               ^
                            v                               |
                                                            |
                         Statistics ----------------- on properties
                                                      of the objects in
                                                      the datasets
```

The SimDM serialization is composed of several XML files that, put together, define the pivot format describing a simulation project. In particular, the pivot format for a whole simulation project is compound of:

- **project.xml** (1 file) holding metadata about the project to which the simulations belongs to.
- **protocol.xml** (1 file) holding metadata about the protocol that the simulations followed (i.e code used to run the models)
- **experiment.xml** (1 file per model run), holding the metadata about each model (which code settings, what calculated values, etc. . .).
- **parties.xml** (1 file) holding the metadata about the people, team, research groups implied in the simulation project and their roles.

As explained below, project.xml, protocol.xml and parties.xml are provided by the SimDAL Repository component. Experiment.xml are provided by the SimDAL Search component.

3 SimDAL API common

The services of the SimDAL components are implemented following a REST design [11] that conforms to the DALI resource [1] description.

3.1 Resources

The list of resources for each of the three SimDAL components are presented below, with their status, mandatory or not and if they are synchronous or not.
The implementer is free to name (set the path) for the resource however they like; the client will find the resource path using the VOSI- capabilities resource.
The resources with a leading * in the table below have their name (access uri) provided in other resource response instead of in the service capability resource. This allow for the implementer a free choice of how he wants to design its uris. For example, one would prefer uris like

```
http://<simdal-search-uri>/views/3
```

when others would prefer

```
http://<simdal-search-uri>/views?viedid=3
```

SimDAL Repository			
Ressource type	Resource name	Mandatory	Sync
{tsearch}	service specific	no	sync
{projects}	service specific	yes	sync
{protocols}	service specific	yes	sync
{services}	service specific	yes	sync
VOSI-availability	/availability	yes	sync
VOSI-capabilities	/capabilities	yes	sync
SimDAL Search			
Resource type	Resource name	Mandatory	Sync
{experiments}	service specific	no	sync
{views}	service specific	yes	sync
∗{fields}	service specific	no	sync/async
∗{cutout}	service specific	yes	async
∗{cutout-preview}	service specific	no	sync
VOSI-availability	/availability	yes	sync
VOSI-capabilities	/capabilities	yes	sync
SimDAL Data Access			
Resource type	Resource name	Mandatory	Sync
{datasets}	service specific	yes	async
∗{cutout}	service specific	no	async
∗{fields}	service specific	no	sync/async
∗{rawdata}	service specific	yes	sync
VOSI-availability	/availability	yes	sync
VOSI-capabilities	/capabilities	yes	sync

3.2 Response format

The response format of the SimDAL API conforms to DAL guideline of using VOTable.

Error The SimDAL API follows the DALI specification in case of service call error. The returned VOTable contains a list of couple (error messages, error code).

```
<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  ...
  <FIELD name="error_msg" datatype="char" arraysize="*"></FIELD>
  <FIELD name="error_code" datatype="int"></FIELD>

  <TABLE>...</TABLE>
</RESOURCE>
```

Result Unless explicitly indicated, the result data in a resource representation is located in the *results* TABLE inside the *results* RESOURCE.

3.3 The 'links' additonal table in VOTable responses

The SimDAL standard use a special table, named *links* in its VOTable response. This table has the following pre-requisite:

- a *link-rel* FIELD holding the semantic attached with what is pointed by the link.

- a *link-uri* FIELD holding the actual url

We chose to add this table mainly to:

- provide links towards resources. Actually, there is not any standard for the URI/links form of the resources described in the SimDAL protocol. This is why it is necessary to have a way to get this non-standard, publisher specific URIs.

  For example, publisher A could have defined it's simdal search views resources as:

  ```
  http://<uri>/my-views/for-id/3
  ```

  whereas publisher B would have implemented

  ```
  http://<uri-b>/views?id=3.
  ```

  There is no standard URI for resources but standard way to get the URI for the resources (as done in almost all the recent DAL standards, see Datalink for example). The "links" table is the way to get these specific URIs.

  Hypermedia Controls done this way is one of the traditional REST best practice, as far as today.

- allow to have some mirroring feature, providing backup URIs in case the main resource would come unavailable, or use them as load balancing.

- make the standard easily extendable by just defining more "link-rel" in future versions of the standard withtout the need to do more.

link with other tables One must know to which table/field the elements of the 'links' table refer to. To do that, we use a relational-like approach, as suggested in the VOTable 1.3 standard (4.1).

In the following example, the VOTable contains 2 tables, 'results', and the 'links' table. We use the grouping feature of VOTable to define that the field 'view' in the 'links' table refers to the field 'id' of the 'result' table.

```
...
<TABLE name="results">
  <FIELD name="id" datatype="char" arraysize="*"></FIELD>
  <FIELD name="created" datatype="char" arraysize="*"></FIELD>
  ...


<TABLE name="links">
  <FIELD name="view" datatype="char" arraysize="*"></FIELD>
  <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
  <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

  <GROUP name="foreign_key" ref="results">
    <GROUP>
      <PARAM name="local_field" value="view"/>
      <FIELDRef ref="id"/>
    </GROUP>
  </GROUP>
...
```

Thus, one defines the link between the table 'links' and other tables of the VOTable by defining:

- a GROUP with the special name *foreign_key*
- a child GROUP with the attribute `ref` set to the foreign table name in the VOTable
  - a PARAM with name `local_field` set to the 'links' table field pointing to the foreign table.
  - a FIELDRef with `ref` to the foreign table referenced field.

Note: here we use only 1 field to reference a foreign table, but it's possible to do the referal with several fields, like a multi attributes primary key in relationnal databses. In this case, the GROUP with name *foreign_key* would have as many child groups as attribute in the primary key.

3.4 Pagination

Because some resources can represent very large collections, a pagination mechanism must be set up.

Easy handling of that collections for the user is achieved by doing the maximum of processing serverside to deal with the pagination. In particular, for collections resources, the response contains the link for direct access to the next page of items.

Given the nature (size in particular) of the data in SimDAL, collections are considered as infinite streams and, as such, we consider only one operation on it: go forward (i.e give me the next page, until infinite...).

We define this API as a common "sub API" to be implemented when mentioned by the APIs in this document.

Resource parameters There is only one resource parameter, defining the collection stream page size.

- `per_page`: *integer*. This define the (maximum) number of items per `page` (i.e the page size), it defaults to `10`.

Resource representation The information about how to reach the next page of the stream is provided by the resource page representation.
Two informations are required in the VOTable:

- a `<LINK content-role='next_page'>` element with `href` pointing towards the next page
- a `<PARAM name='per_page'>` holding the value of the `per_page` parameter passed to the resource.

Here is an example with a VOTable resource representation:

```
...
<RESOURCE type="results">
  <INFO name="QUERY_STATUS" value="OK"/>
  ...
  <TABLE name="results">
    <LINK content-role="next_page" href="http://<simdal-search-uri>/experiments&per_page=5&page=2"/>
    <PARAM name="per_page" datatype="int" value="5"/>
  ...
</RESOURCE>
...
```

pagination link inference One should not infer a way to access directly a particular page from the URI pattern of the "next-page" link value. Indeed, nothing is standard here and it's up to the service publisher to define its own internal implementation.

VOSI-availability A SimDAL service must have a VOSI-availability resource.

VOSI-capabilities A standalone SimDAL service must have a VOSI-capabilities resource. The standardID for the resources is, in function of the sub-API :

```
ivo://ivoa.net/std/SimDALRepository#{resource}-1.0
ivo://ivoa.net/std/SimDALSearch#{resource}-1.0
ivo://ivoa.net/std/SimDALDataAccess#{resource}-1.0
```

3.5 Time

All timestamps are returned in ISO 8601 format `YYYY-MM-DDTHH:MM:SSZ`.

3.6 Examples conventions

In the examples provided with the API specs, we assume the following providers for the SimDAL components API implementation:
- SimDAL Repository at `http://<SimDAL Simulations repository URI>`
- SimDAL Search at `http://<SimDAL Simulations Search URI>`
- SimDAL Data Access at `http://<SimDAL Data Access URI>`

4 SimDAL Repository

A SimDAL Repository is basicaly a database of the metadata of project instances and of the protocol (simulation code) they use, along with the associated service instances.

These metadata are serialized as XML files as described in the Simulation Data Model standard.
SimDAL Repositories API aim to:

- search in the repository for metadata on project and/or protocol instances. This search is done on concepts that are found in the text attributes of the SimDM classes of the corresponding package (Examples: N-body, spectrum, proton density, star, ...).
- provide SimDM XML serializations of the project and protocol SimDM packages

In a SimDAL Repository, resources are identified by a repository local identifier, named a *vodid*. The definition and management of this ID is up to the service provider.

4.1 {tsearch}

The `{tsearch}` (text search) resource is used to search for a text pattern in the XML serialisations stored in a SimDAL Repository. Only the text attributes of the elements are searched for (i.e the text attributes of the SimDM class, for example `name`, `description`, etc...).

Example: get a VOTable representation of the `{tsearch}` resource parametrized by the text pattern "proton". Here the resource is located at `<SimDAL Repository URI>`.

```
http://<SimDAL Repository URI>/tsearch?q=proton
```

Parameter

- `q`: the text pattern to search for. `q` is not case sensitive. The search logic is up to the publisher (auto wild card insertion, spellchecking etc...), so is the support of extra features like wildcards. In this case, the documentation of such features is publisher responsability.

The example query would return:

```
...

<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  <PARAMETER name="q" value="proton"/>
  ...
  <table name="results">

    <LINK content-role="next_page"
          href="http://<simdal-search-uri>/experiments&per_page=5&page=1"/>
    <PARAM name="per_page" datatype="int" value="5"/>

    <FIELD name="match" datatype="char" arraysize="*"></FIELD>
    <FIELD name="simdm/class" datatype="char" arraysize="*"></FIELD>
    <FIELD name="simdm/class/attribute" datatype="char" arraysize="*"></FIELD>
    <FIELD name="authority" datatype="char" arraysize="*"></FIELD>
    <FIELD name="project" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
<tr>
  <td>Proton density</td>
  <td>InputParameter</td>
  <td>name</td>
  <td>ism.obspm</td>
  <td>20141027_pdr</td>
</tr>
....
      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

This resource should implement the pagination API.

Response schema

- `match`: the matching value

- `simdm/class`: the SimDM class containing the attribute with the matching value
- `simdm/attribute`: the attribute of the simdm/class containing the matching value
- `authority`: the VO authority ID
- `project`: the publisherdid of the project

In case of no match, the returned VOTable `<tabledata>` element is empty.

4.2 {projects}

The resource represents a list of links toward the available project SimDM XML serializations.

```
http://<simdal-repo-uri>/projects
```

Parameters Returns a list of available projects.

```
...
<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  ...
  <TABLE>
    <FIELD name="vodid" datatype="char" arraysize="*"></FIELD>
    <FIELD name="authority" datatype="char" arraysize="*"></FIELD>
    <FIELD name="name" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>

    <DATA>
      <TABLEDATA>
        <tr>
          <td>ism.obspm/20141027_pdr</td>
          <td>ism.obspm</td>
          <td>PDR 1.6 rev 312 grid, 201410</td>
          <td>2014-09-24T14:24:31.580+02:00</td>
        </tr>
        ...
      </TABLEDATA>
    </DATA>
  </TABLE>

  <TABLE name="links">
    <FIELD name="project" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
      <GROUP>
        <PARAM name="local_field" value="project"/>
        <FIELDRef ref="vodid"/>
      </GROUP>
    </GROUP>

    <data>
      <tabledata>
        <tr>
          <td>ism.obspm/20141027_pdr</td>
          <td>simdm/project/XML</td>
          <td>http://<simdal-repo-uri>/projects/ism.obspm/20141027_pdr</td>
        </tr>
        <tr>
          <td>ism.obspm/20141027_pdr</td>
          <td>simdm/protocol/XML</td>
          <td>http://<simdal-repo-uri>/protocols/3</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

</RESOURCE>
...
```

4.3 {protocols}

The resource represents a list of links toward the available protocol SimDM XML serializations.

```
http://<simdal-repo-uri>/protocols?name=*pdr*
```

Parameters

- `name`: a text pattern filtering the project on their name. It can contains a tiny regex subset: * (equivalent of .* in PERL regex fashion). This parameter is optional. If not provided all the names are considered.
- `project`: filters the protocols defined in the project having the specified vodid. This parameter is optional. If not provided return all the protocols from all the projects.

Returns the list of all available protocols in the repository.

```
...
<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  <PARAMETER name="name" value="*pdr*"/>
  ...
  <TABLE>
    <FIELD name="vodid" datatype="char" arraysize="*"></FIELD>
    <FIELD name="authority" datatype="char" arraysize="*"></FIELD>
    <FIELD name="name" datatype="char" arraysize="*"></FIELD>
    <FIELD name="project" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>

    <DATA>
      <TABLEDATA>
        <tr>
          <td>my_protocol_pubid</td>
          <td>my_vo_authority_id</td>
          <td>my_protocol_name</td>
          <td>my_project_pubid</td>
          <td>my_creation_date</td>
        </tr>
        ...
      </TABLEDATA>
    </DATA>
  </TABLE>

  <TABLE name="links">
    <FIELD name="protocol" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
```

```
      <GROUP>
        <PARAM name="local_field" value="protocol"/>
        <FIELDRef ref="vodid"/>
      </GROUP>
    </GROUP>

    <data>
      <tabledata>
        <tr>
          <td>my_protocol_pubid</td>
          <td>simdm/protocol/XML</td>
          <td>http://link_toward_protocol_xml_serialization</td>
        </tr>
        <tr>
          <td>my_protocol_pubid</td>
          <td>simdm/project/XML</td>
          <td>http://link_toward_project__xml_serialization</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

</RESOURCE>
...
```

4.4 {services}

Returns the services URIs associated with a particular project. In particular, it provides the URI of the SimDAL Search Service that will allow search of datasets & models in the project through views on the project instance.

```
http://<simdal-repo-uri>/services?project=my_project_pubid
```

will return

```
...
<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  <PARAMETER name="project" value="my_project_id"/>
  ...
  <TABLE>
    <FIELD name="vodid" datatype="char" arraysize="*"></FIELD>
    <FIELD name="authority" datatype="char" arraysize="*"></FIELD>
    <FIELD name="project" datatype="char" arraysize="*"></FIELD>
    <FIELD name="type" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>

    <DATA>
      <TABLEDATA>
        <tr>
          <td>my_search_service_pubid</td>
          <td>my_vo_authority_id</td>
          <td>my_project_id</td>
          <td>simdal_search_service</td>
          <td>my_creation_date</td>
        </tr>
        <tr>
          <td>my_access_data_service_pubid</td>
          <td>my_vo_authority_id</td>
          <td>my_project_id</td>
          <td>simdal_data_access</td>
          <td>my_creation_date</td>
        </tr>
        <tr>
          <td>my_summary_service_pubid</td>
          <td>my_vo_authority_id</td>
          <td>my_project_id</td>
          <td>custom_service</td>
          <td>my_creation_date</td>
        </tr>
        ...
      </TABLEDATA>
    </DATA>
  </TABLE>

  <TABLE name="links">
    <FIELD name="service" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
      <GROUP>
        <PARAM name="local_field" value="service"/>
        <FIELDRef ref="vodid"/>
      </GROUP>
    </GROUP>

    <data>
      <tabledata>
        <tr>
          <td>my_search_service_pubid</td>
          <td>simdal/search</td>
          <td>http://link_toward_search_service_ressource</td>
        </tr>
        <tr>
          <td>my_access_data_service_pubid</td>
          <td>simdal/data_access</td>
          <td>http://link_toward_access_data_ressource</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

</RESOURCE>
...
```

Parameters

- `project`: Filters the services defined in the project having the specified vodid. This parameter is mandatory.

Service type `link-rel` can have the following values:

link-rel	SimDAL API
simdal/repository	Simdal Repository
simdal/search	Simdal Search
simdal/data_access	Simdal Data Access

5 SimDAL Search

The SimDAL Search component is intended to search for datasets from models with specific input/output values.

In the normal use case, the user comes to a SimDAL Search service after having discovered a particular project of interest (and its protocols) in a SimDAL Repository. In particular, the URI of the SimDAL Search service is identified there.

A detailed description of the SimDAL Search API design can be found in Appendix B.

To allow search on values of models input parameters and output (such as statistics on produced datasets), a SimDAL Search service presents the data to users in the form of *views*. We can imagine it as an ASCII tab separated file, but so big that it's not possible to retrieve it through the network, so that it is kept on the server.
The only way to access it is to define subsets -cutouts- first. A cutout is defined by either filtering the columns -the fields- or the rows (by applying constraints on fields values) of the view. That is what would be done when performing a SQL query on a single flat table, SELECT filters on columns, and WHERE filters on rows. This server-side file is abstracted, in a VO context, as a VOTable.

A schema must be provided by the service for each defined view (see below), so that the available fileds are knows by the user.

SimDAL does not define how these data are stored and managed server-side because, depending on the data (volume and heterogeneity), different publishers will have to implement different technological solutions.

To have views of homogeneous objects, a view should be defined per combination Project, Protocol, ObjectType. The grain of the rows of the view would be in this case a SimDM OutputDataset instance (i.e the view allows to find output datasets). This view would contain, for each dataset, fields representing the input parameters values of the parent experiment and/or statistics values on the dataset, plus all other quantities the publisher thinks are useful to discover his data.

The SimDAL specification does not define any mandatory view. The choice is up to the publisher, as long as the corresponding schema is provided.

SimDM semantic In SimDAL we actually do not talk about SimDM semantic like InputParameter or Property. The user does not necessarily know (or want to know) about SimDM. So just present him views of homogenous objects, characterized by a set of fields (the link with SimDM is done through the use of a utype attribute to the fields).

5.1 SimDAL Search API design

As stated above, the SimDAL Search service allows queries on views of a SimDM Project instance. Each view is defined by a schema.

View schema In order to query the views of a SimDAL Search service, we must have a way to have informations about it, in particular about its fields. This schema is represented as a VODML-UTYPE VOTable.

Mandatory fields Although there are not any mandatory views, if a view contains datasets intended to be later accessed through a Data Access component, it must have a field that identify a dataset.

This dataset identifier filed is defined in the view schema through the special group *dataset_id*:

```
<GROUP name="dataset_id">
  <FIELDRef ref="vodid"/>
</GROUP>
```

Query A view query is done by specifying 2 parameters: `select` and `where`.

- `select` is a list of the fields to return the value of
- `where` is a list of constraints on fields to filter the objects of the view (it's very similar to the concepts of a simple SQL query)

The `select` and `where` parameters are JSON elements passed to the service as a POST http query. This has the following advantages:

- Not limited by a potential maximum GET URL length for some browsers/servers
- Not limited by the constraints about the data structures / allowed characters in a GET query string
- Allow JSON document as service input, i.e nice and clean data structures
- Enable future extension, for example to support basic arithmetic operations on fields

5.2 {experiments}

The resource represents a list of links toward the available experiment SimDM XML serializations. This resource should implement the pagination API.

```
http://<simdal-search-uri>/experiments?per_page=5
```

Returns all the available experiments in the project described by the SimDAL Search service.
The `type` field is used to indicate the SimDM *Experiment* concrete class, that is either *Simulation* or *PostProcessing*.

```
...
<RESOURCE type="results">
  <INFO name="QUERY_STATUS" value="OK"/>
  ...
  <TABLE name="results">
    <LINK content-role="next_page"
          href="http://<simdal-search-uri>/experiments&per_page=5&page=2"/>
    <PARAM name="per_page" datatype="int" value="5"/>
    <FIELD name="publisherdid" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>
    <FIELD name="type" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
```

```
        <tr>
          <td>my_simulation_pubid</td>
          <td>my_creation_date</td>
          <td>simulation</td>
        </tr>
        <tr>
          <td>my_postprocessing_pubid</td>
          <td>my_creation_date</td>
          <td>postprocessing</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

  <TABLE name="links">
    <FIELD name="experiment" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
      <GROUP>
        <PARAM name="local_field" value="experiment"/>
        <FIELDRef ref="publisherdid"/>
      </GROUP>
    </GROUP>

    <data>
      <tabledata>
        <tr>
          <td>my_publisher_id</td>
          <td>simdm/experiment/XML</td>
          <td>link_toward_experiment_xml_serialization</td>
          </tr>
        <tr>
          <td>my_publisher_id</td>
          <td>simdm/protocol/XML</td>
          <td>link_toward_protocol_xml_serialization</td>
        </tr>
        <tr>
          <td>my_publisher_id</td>
          <td>simdm/project/XML</td>
          <td>http://link_toward_project_xml_serialization</td>
        </tr>
        ...

      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

5.3 {views}

```
http://<simdal-search-uri>/views
```

Will return a list of metadata about the available views in the SimDAL Search Service. The metadata includes in particular a link towards the view schema (a VOTable-UTYPE header).
Because it's not possible to store structures in a table field, a special table holding the links (defined by 2 attributes: URI, rel) associated with a view is defined, with id="links".

```
...
<RESOURCE type="results">
  <INFO name="QUERY_STATUS" value="OK"/>
  ...

  <TABLE name="results">
    <FIELD name="id" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>
    <FIELD name="objecttype" datatype="char" arraysize="*"></FIELD>
    <FIELD name="protocol" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
        <tr>
          <td>my_view_pubid</td>
          <td>my_creation_date</td>
          <td>my_objecttype</td>
          <td>my_protocol_pubid</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

  <TABLE name="links">
    <FIELD name="view" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
      <GROUP>
<PARAM name="local_field" value="view"/>
<FIELDRef ref="id"/>
      </GROUP>
    </GROUP>

    <data>
      <tabledata>
        <tr>
          <td>my_view_pubid</td>
          <td>view/schema</td>
          <td>http://<schema_file></td>
        </tr>
        <tr>
          <td>my_view_pubid</td>
          <td>resources/fields</td>
          <td>http://<fields_ressource></td>
        </tr>
        <tr>
          <td>my_view_pubid</td>
          <td>resources/cutout</td>
          <td>http://<cutout_ressource></td>
        </tr>
        <tr>
          <td>my_view_pubid</td>
          <td>resources/cutout-review</td>
          <td>http://<cutout_preview_ressource></td>
        </tr>

        <tr>
          <td>my_view_pubid</td>
          <td>simdm/protocol/XML</td>
          <td>http://<protocol_xml_serialization></td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

Note Because we expect the collection resource to be quite small, the pagination API is not required, but should be implemented as soon as the expected collection size get large.

Link towards XML serializations In order to link the view with a standard simulation project description (i.e an IVOA SimDM serialization), a link towards the project.xml and protocol.xml describing the project in which the view is defined should be added in the ”links” table.

View schema A view schema is a VOTable following the VOTable/utype serialization of a subset of the SimDM, as defined in the VO-DML document.
In practice, this format is a VOTable without the DATA element, i.e containing only the metadata header describing the DATA section (its schema). This enforces the analogy of a view as a server-side VOTable.
Example:

```
<VOTABLE XMLns="http://www.ivoa.net/XML/VOTable/v1.2">
  <RESOURCE name="">
    <TABLE name="my_view_pubid">
      <GROUP utype="vo-dml:Instance.root">
        <PARAM name="type" utype="vo-dml:Instance.type"
               value="simdm:resource/experiment/output_dataset"/>
        <PARAM name="object"
               utype="simdm:/resource/experiment/output_dataset.object_type"
               value="grain"/>
        <FIELDRef ref="outputdataset_pubdid"
                  utype="simdm:/resource/experiment/output_dataset.publisherdid"/>
        <FIELDref ref="outputdataset_pubdid" utype="vo-dml:ObjectType.ID"/>

        <GROUP utype="simdm:resource/experiment/output_dataset.characterisation">
          <PARAM name="type" utype="vo-dml:Instance.type" value="vo-dml:Collection"/>
          <GROUP utype="vo-dml:Instance.item">
            <PARAM name="type" utype="vo-dml:Instance.type"
                   value="simdm:resource/experiment/statistical_summary"/>
            <PARAM name=""
                   utype="simdm:/resource/experiment/statistical_summary.axis"
                   value="temperature_grain_silicates"/>
            <PARAM name=""
                   utype="simdm:/resource/experiment/statistical_summary.statistic"
                   value="max"/>
            <FIELDRef ref="c:max:temperature_grain_silicates"
                      utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"/>
          </GROUP>
        </GROUP>
      </GROUP>

      <GROUP name="dataset_id">
        <FIELDRef ref="outputdataset_pubdid"/>
      </GROUP>

      <FIELD name="Max Grain temperature for silicates"
             ID="c:max:temperature_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <VALUES>
          <MIN value="16.9"/>
          <MAX value="77.2" inclusive="yes"/>
        </VALUES>
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
      </FIELD>
      <FIELD name="outputdataset publisherDID"
             ID="outputdataset_pubdid"
             utype="simdm:/resource/experiment/output_dataset.publisherdid"
             datatype="string"/>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

5.4 {fields}

The `{fields}` resource location is provided by the `{views}` resource, through the link-rel "resources/fields". Example:

```
http://<fields_ressource>
```

The resource itself does not provide a representation, because the collection may be very large and the data is not really relevant since it can be obtained through the `{views}` resource schema link.
Thus, the resource can be used in two ways, for two very common use-cases: for searching a field or get the schema of a specific field (once it has been identified through a previous search).

search The search is done by providing the REQUEST=search GET parameter. The search is done on the field name based on the text pattern provided through the parameter q.

```
http://<fields_ressource>?REQUEST=search&q="Grain temperature"
```

would return:

```
...
<RESOURCE type=results>
  <INFO name=QUERY_STATUS value=OK/>
  <PARAMETER name="q" value="Grain temperature for silicates"/>
  ...
  <table>
    <FIELD name="match" datatype="char" arraysize="*"></FIELD>
    <FIELD name="id" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
        <tr>
          <td>Max Grain temperature for amorphous carbons</td>
          <td>c:max:temperature_grain_amorphous_carbons</td>
        </tr>
  ...
        <tr>
          <td>Max Grain temperature for silicates</td>
          <td>c:max:temperature_grain_silicates</td>
        </tr>
      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

This resource should implement the pagination api. In case of no match found, the returned VOTable `<tabledata>` element is empty.

get field schema Getting the schema of a specific field is done by providing REQUEST=schema and `ID=<field id>`. For example, for the field of ID ‘`c:max:temperature_grain_silicat`

```
http://<field_ressource>?REQUEST=schema&ID=c:max:temperature_grain_silicates
```

That would return:

```
<VOTABLE XMLns="http://www.ivoa.net/XML/VOTable/v1.2">
  <RESOURCE name="">
    <TABLE name="my_view_pubid">
      <FIELD name="Max Grain temperature for silicates"
             ID="c:max:temperature_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <VALUES>
          <MIN value="16.9"/>
          <MAX value="77.2" inclusive="yes"/>
        </VALUES>
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
      </FIELD>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

5.5 {cutout}

The resource represents a VOTable encoded list of the values that fulfil the cutout request parameters.
It’s really a cutout in a hypercube (in this case the axis of the hypercube are the fields of the *view*).
Since the *views* in SimDAL are mainly about metadata, it’s then more a metadata cutout than a traditional data cutout (see Simdal Data Access for that).

```
curl -H "Content-Type: application/JSON" -d

"{
  where: [
    {':att': 'c:max:temperature_grain_silicates', ':val': 30, ':op': '>'},
    {':att': 'c:min:charge_grain_silicates', ':val': -1, ':op': '<'}
  ],

  select: [
    'c:gas_density',
    'c:max:temperature_grain_silicates',
    'c:gas_temperature',
    'c:min:charge_grain_silicates'
  ]
}"

http://<cutout_resource>?per_page=5
```

returns (with `per_page` to 5):

```
<VOTABLE XMLns="http://www.ivoa.net/XML/VOTable/v1.2">
  <RESOURCE name="">
    <TABLE name="my_view_pubid">
      <LINK content-role="next_page"
            href="http://<simdal-search-uri>/cutout?view=my_view_pubid&per_page=5&page=2"/>
      <PARAM name="per_page" datatype="int" value="5"/>
      <FIELD name="Max Grain temperature silicates"
             ID="c:max:temperature_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
      </FIELD>

      <FIELD name="Min Charge Grain silicates"
             ID="c:min:charge_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
      </FIELD>

      <FIELD name="Gas density"
             ID="c:gas_density"
             utype="simdm:/resource/experiment/parameter_setting.numeric_value.value"
             datatype="float" unit="cm-3">
      </FIELD>
      <FIELD name="Gas temperature"
             ID="c:gas_temperature"
             utype="simdm:/resource/experiment/parameter_setting.numeric_value.value"
             datatype="float" unit="K">
      </FIELD>

      <DATA>
        <tabledata>
          <TR>
            <TD>50.57</TD>
            <TD>-2.31</TD>
            <TD>1e5</TD>
            <TD>456.2</TD>
          </TR>
          <TR>
            <TD>52.34</TD>
            <TD>-1.11</TD>
            <TD>1e5</TD>
            <TD>402.3</TD>
          </TR>
          <TR>
            <TD>31.01</TD>
            <TD>-2.81</TD>
            <TD>1e5</TD>
            <TD>103.2</TD>
          </TR>
          <TR>
            <TD>37.09</TD>
            <TD>-1.49</TD>
            <TD>1e4</TD>
            <TD>110.2</TD>
          </TR>
          ...
        </tabledata>
      </DATA>
    </TABLE>
  </RESOURCE>
```

```
</VOTABLE>
```

This resource MUST implement the pagination api. In case of no match found, the returned VOTable `<tabledata>` element is empty.

Parameters The query is passed as a JSON document following the specification:

- `select`: a list of the view's fields to return the value of
- `where`: a list of constraints to filter the objects of the view

`constraint` is a JSON object with the following keys:

- `:att`: the field to constrain
- `:op`: the operator defining the nature of the constraint. Possible values are: `>, <, =, >=, <=, !=`
- `:val`: the value constraining the field

Response format The response format is a VOTable. The complete metadata header, describing the complete schema of the view is not required, with the exception of the FIELD elements.
Also, 2 informations are required in the VOTable:

- `<LINK content-role='next_page'>`: the URL towards the next page, as described in the "Pagination in SimDAL" part.
- `per_page`: the value of the `per_page` parameter used in the API call, as a PARAMETER element.

5.6 {cutout-preview}

As explained above, when a cutout is requested, the service returns, essentialy, a list of the values that fulfil the cutout request.
The cutout-preview optional feature provides a summary of the information that would be obtained in the cutout: minimal and maximum values for each of the requested fields and the total number of results that would be obtained. This is done similar to the one for the "cutout" but adding the VALUES element for each FIELD and no DATA.
This is useful so that the final user can get a hint about if he is asking for too many results or not and, thus, if he should refine the query.
Depending on how the service is implemented, in some cases it is easier and faster to generate this preview than the actual list of results, but in another cases it would be the opposite. Thus, the feature is optional.

```
curl -H "Content-Type: application/JSON" -d

"{
  where: [
    {':att': 'c:max:temperature_grain_silicates', ':val': 30, ':op': '>'},
    {':att': 'c:min:charge_grain_silicates', ':val': -1, ':op': '<'}
  ],

  select: [
    'c:proton_density',
    'c:max:temperature_grain_silicates',
    'c:gas_temperature',
    'c:min:charge_grain_silicates'
  ]
}"

http://<cutout_preview_resource>?per_page=5
```

returns:

```
<VOTABLE xmlns="http://www.ivoa.net/xml/VOTable/v1.2">
  <RESOURCE name="">
    <TABLE name="my_view_pubid">
      <PARAM name="nresults" datatype="int" value="5000"/>
      <FIELD name="Max Grain temperature silicates"
             ID="c:max:temperature_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
        <VALUES>
           <MIN value="30.1"/>
           <MAX value="60.2"/>
        </VALUES>
      </FIELD>

      <FIELD name="Min Charge Grain silicates"
             ID="c:min:charge_grain_silicates"
             utype="simdm:/resource/experiment/statistical_summary.numeric_value.value"
             datatype="float" unit="K">
        <!-- SKOS concept (SimDM 'label' attribute) -->
        <LINK content-role="type"
              href="http://purl.org/astronomy/vocab/PhysicalQuantities/TemperatureOfGrains"/>
        <VALUES>
           <MIN value="-4.1"/>
           <MAX value="-1.1"/>
        </VALUES>
      </FIELD>

      <FIELD name="Proton density"
             ID="c:proton_density"
             utype="simdm:/resource/experiment/parameter_setting.numeric_value.value"
             datatype="float" unit="cm-3">
        <VALUES>
           <MIN value="1.0e5"/>
           <MAX value="1.0e4"/>
        </VALUES>
      </FIELD>
      <FIELD name="Gas temperature"
             ID="c:gas_temperature"
             utype="simdm:/resource/experiment/parameter_setting.numeric_value.value"
             datatype="float" unit="K">
        <VALUES>
           <MIN value="2.7"/>
           <MAX value="11000.0"/>
        </VALUES>
       </FIELD>


    </TABLE>
  </RESOURCE>
</VOTABLE>
```

6 SimDAL Data Access

The basic use case is to find a dataset of interest by searching / browsing the pivot formats of various projects, then access the raw data to do further analysis (load them into vizualisation tools, extract pieces of cubes, download parts of spectra, etc. . .).

Simulation outputs are stored in various heterogenous formats: ASCII, HDF5, VTK, home-made format, ... Moreover, some of them can be big, forbidding a direct download of the whole data. In that case, it is necessary to extract the part of the data to be downloaded. When possible, it is recommended to provide downloaded data in VO-compatible formats so that they can be manipulated by VO-tools and VO-applications. When not possible, specific softwares have to be used to read the data.

This defines the need for a standard raw data access service. The contract is to accept a (simple SQL-like) query on properties of objects (the same than the one of the SimDAL Search API) of a dataset and to return a subset of the dataset containing the objects satisfying the query. The benefits are:

- downloading the whole data is no longer needed
- installing, configuring, using specific extraction tools no longer needed for the end user.
- knowledge (even basic) about the raw data format is no longer needed
- standard, interoperable, web accessible, VO access interface

6.1 SimDAL Data Access API design

The SimDAL Data Access component has a design very similar to the one of SimDAL Search. It allows the user to submit queries to perform cutout on the datasets (SimDM OutputDatasets) produced by the project's models (SimDM Experiment instances) of the project.
As with SimDAL Search, the datasets can be thought about as 'views' where rows are the objects and columns (fields) are the object properties (see SimDAL Design in appendix for more detail about this conceptual mapping).
In order to make the SimDAL Data Access component as independent as possible, the 'views' schemas are provided by the service, even if the information could theoretically by grabbed from the protocol.xml related to the dataset's ObjectType. This makes it possible for a publisher to provide its own implementation of the component, without having to provide neither the Repository, nor the SimDAL Search.
In the common use case, the dataset ID (publisherDID) to provide to the cutout resource is found in the SimDAL Search component.

6.2 {datasets}

This resource represents a set of basic metadata for all the datasets available in the service.

```
http://<simdal-data-access-uri>/datasets?dataset=my_dataset_pubdid
```

will returns the basic metadata of the dataset `my_dataset_pubdid`, along with a PARAM element to recall the value of the *dataset* parameter. In particular, it will provide the links towards the schema of the dataset and the associated fields resource.

```
...
<RESOURCE type="results">
  <INFO name="QUERY_STATUS" value="OK"/>
  <PARAM name="dataset" value="my_dataset_pubdid"/>
  ...

  <TABLE name="results">
    <FIELD name="id" datatype="char" arraysize="*"></FIELD>
    <FIELD name="created" datatype="char" arraysize="*"></FIELD>
    <FIELD name="objecttype" datatype="char" arraysize="*"></FIELD>
    <FIELD name="protocol" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
        <tr>
          <td>my_dataset_pubid</td>
          <td>my_creation_date</td>
          <td>my_objecttype</td>
          <td>my_protocol_pubid</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>

  <TABLE name="links">
    <FIELD name="dataset" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-rel" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <GROUP name="foreign_key" ref="results">
      <GROUP>
        <PARAM name="local_field" value="dataset"/>
        <FIELDRef ref="id"/>
      </GROUP>
    </GROUP>


    <data>
      <tabledata>
        <tr>
          <td>my_dataset_pubid</td>
          <td>dataset/schema</td>
          <td>http://<schema_file></td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>resources/fields</td>
          <td>http://<fields_ressource></td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>resources/cutout</td>
          <td>http://<cutout_ressource></td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>resources/rawdata</td>
          <td>http://<rawdata_ressource></td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>simdm/protocol/XML</td>
          <td>http://<protocol_xml_serialization></td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

parameters

- `dataset`: optional parameter to filter the returned list to only one dataset.

standalone Data Access component In cases where the publisher wants to publish only some datasets from a simulation project without setting up a Search Component, the datasets resource can be accessed withtout the *dataset* parameter, returning the list of the metadata for all the published datasets of the project. In this case, the number of datasets is generally small, so that the list size would be manageable.

One must keep in mind that, in this case, the only way to know what the datasets are about are the basic metadata provided by the dataset resource (ex: objecttype, protocol). This resource should implement the pagination API.

6.3 {fields}

The API is the same than the one of the SimDAL Search component. Having this here allows the users to query the Data Access component without requiring the metadata contained in the protocol.xml of the project.
This resource should implement the pagination API.

6.4 {cutout}

The API is the same as the SimDAL Search `{cutout}` resource with the exception of the following:

- it's an async resource conforming to DALI async
- the pagination API is optional, because according to the requested output method/format, it may not be relevant.
- the job result is a link towards the result data (whose format is defined by the 'RESPONSEFORMAT' parameter).
- has a 'RESPONSEFORMAT' parameter defining the expected result data format. It can be VOTable, txt, or 'raw'. In the later case, the job resulting VOTable must include the raw format definition, as a mime/type

```
curl -H "Content-Type: application/JSON" -d

"{
  where: [
    {':att': 'temperature_grain_silicates', ':val': 30, ':op': '>'},
    {':att': 'charge_grain_silicates', ':val': -1, ':op': '<'}
  ],

  select: [
    'temperature_grain_silicates',
    'charge_grain_silicates'
  ]
}"

http://<simdal-data-uri>/cutout?view=dataset_v1_232&format=raw&RUNID=j1
```

would return

```
<uws:job xsi:schemaLocation="http://www.ivoa.net/XML/UWS/v1.0 UWS.xsd "
 XMLns:XML="http://www.w3.org/XML/1998/namespace"
 XMLns:uws="http://www.ivoa.net/XML/UWS/v1.0"
 XMLns:xlink="http://www.w3.org/1999/xlink"
 XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <uws:jobId>ism-obspm-simdal-job124</uws:jobId>
  <uws:ownerId xsi:nil="true"/>
  <uws:phase>COMPLETED</uws:phase>
  <uws:startTime>2015-02-12T10:04:20.448478</uws:startTime>
  <uws:endTime>2015-02-12T10:04:20.448478</uws:endTime>
  <uws:executionDuration>1400</uws:executionDuration>
  <uws:destruction>2015-02-22T10:04:20.448478</uws:destruction>
  <uws:parameters>
    <uws:parameter id="dataset">my_dataset_pubid</uws:parameter>
    <uws:parameter id="runid">my_run_pubid</uws:parameter>
    <uws:parameter id="responseformat">raw</uws:parameter>
  </uws:parameters>
  <uws:results>
    <uws:result id="cutout"
xlink:href="http://<simdal-data-uri>/cutout/my_link_toward_results"/>
  </uws:results>
  ...
</uws:job>
```

async Unlike for the SimDAL Search component handling views of metadata, the SimDAL Data Access component deals with raw output data from the code (OutputDataset). This data can be very large and the process of extracting a subset of it (to cutout) may be very CPU intensive and result in very large time processing and final file size.
Because of that characteristic, the synchronous collection pagination approach used above may not be very relevant here, that's why it is defined as optional.

UWS extension The following 2 points differ in SimDAL Data Access from the standard UWS:

- the `{cutout}` resource (mapped to `{jobs}` in UWS document) does not return a list of the jobs. For various reasons but in particular because of security concerns.
- the job submission is done through a simple JSON document as described in SimDAL Search. Mainly because it's clean, simple, and homogenous with the SimDAL Search API.

6.5 {rawdata}

6.5.1 rationale

Because many publishers may just want to publish small output files of their experiments a resource must be defined to allow plain old download of raw material.

Another common use case is the publisher providing, along with the raw data, a specific analysis tool able to read the raw data. In this case, he would just provide the files, without publishing in a SimDAL Service for search capability, all the search work would be done through the specific analysis tool directly on the raw data files.

Also, because there can be various files associated with a single dataset, or the files can be big, the rawdata resource provide a list of links, along with their type description as a mime/type, from where the files can be downloaded.

6.5.2 resource interface

```
http://<simdal-data-access-uri>/rawdata?dataset=my_dataset_pubdid
```

would return the following VOTable

```
...
<RESOURCE type="results">
  <INFO name="QUERY_STATUS" value="OK"/>
  <PARAM name="dataset" value="my_dataset_pubdid"/>
  ...
  <TABLE name="links">
    <FIELD name="dataset" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-type" datatype="char" arraysize="*"></FIELD>
    <FIELD name="link-uri" datatype="char" arraysize="*"></FIELD>

    <data>
      <tabledata>
        <tr>
          <td>my_dataset_pubid</td>
          <td>text/plain</td>
          <td>link_toward_ascii_file</td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>application/fits</td>
          <td>link_toward_fits_file</td>
        </tr>
        <tr>
          <td>my_dataset_pubid</td>
          <td>application/x-hdf</td>
          <td>http://link_toward_hdf5_file</td>
        </tr>
        ...
      </tabledata>
    </data>
  </TABLE>
</RESOURCE>
...
```

A Frequently Asked Questions

A.1 Why have you called the tsearch parameter 'q' ?

Google and others use `q` in almost all its apis, in particular its search engine. We thought it would make it easy to understand by developper thanks to the analogy.

```
https://www.google.fr/search?q=ivoa
```

A.2 Why are the links towards the SimDAL components in the SimDAL repository ?

One could ask why the link toward the Data Access is not provided by the SimDAL Search service, because we need to know a dataset ID before doing a cutout on it through SimDAL Data Access. That ID is found in the SimDAL Search component.
But, because a publisher can choose to provide any of the (or publish to an existing) SimDAL services, there is not always a SimDAL Search. As a result, this makes it not very relevant to locate the link towards a SimDAL Access component there.

In addition, a SimDAL Access link may be the only one thing a user wants after discovering a project in a SimDAL repository. This is true in the common use case of just a plain old download of the project files.
In Practice, many publishers would just provide that raw files, would it be because the simulations are small enough or because it provide its own specific analyzing tool which take as input the code raw files.

A.3 Why do we need an access protocol ?

Aren't the SimDM serialization XML files self sufficients as a standard exchange -*pivot*- format ?

> The pivot format is convenient to *exchange* metadata
>
> It's not convenient to actually *use* that metadata.

The pivot format has many files A common use case in theoretical simulation projets is to build a grid -a collection- of models describing a range of parameters values.
In this case, a project is compound of a lot of model runs, i.e SimDM Experiment instances. This makes the pivot format for this project have a lot of .xml files.
Because that makes it hard to find what we look for in this mass of metadata, we need some way to search inside the whole project files set.
That challenge can be overcome by defining a *search* service, doing a smart full-text search in the pivot files. In particular in the text attributes of the SimDM class instances. Example:

```
search("cloud")

-> cloud      : object type produced by the code "PDR"
-> cloud_dens : property of the object type "cloud"
-> ...
```

So, we need help As a result of the inability of the pivot format to effectively serve the end user use cases, some help is needed.
SimDAL has been brought up to help the user get the best from the vo simulation pivot format (and so SimDM).
SimDAL is divided in 3 components, each serving a particuler coherent set of use cases.

- SimDAL Repository
- SimDAL Search

- SimDAL Data Access

Th SimDAL Data Access Layer places itslef between the data producer and the end user/consumer:

```
        IN                                   OUT

                                    <-- Repository API

SimDM serialization   -->  SimDAL   <-- Search API       <--    End User

Simulation Raw Data   -->           <-- Data Access API
```

B SimDAL Search design

The SimDAL Search is intended to search for models with specific input/output values.

To support that task we can define any number of views of an instance of a SimDM project (i.e of all its Experiment instances). We could, virtually, define views for any combination of SimDM classes attributes. But for our use case UC3, it's interesting to define a view presenting the experiments input/output as a big flat virtual table. It would aggregate in columns the InputParameter of Protocol used by the Experiment instances and some StatisticalSummary instances on the Properties of the objects in its OutputDatasets.

To have a view on homogeneous objects, we have to define a view per (Project, Protocol, ObjectType). The grain of the rows of the view is an OutputDataset instance.

We now need a way to attach metadata to this view, so that we know its structure/schema and how to query it.

In the VO, a similar issue is addressed through TAP_SCHEMA in the TAP standard for the tables in a relational database. In our case, we prefer the term *view* because it does not immediately refer to a relational database system (RDBMS) the way *table* does, thus decoupling our logical "view" to any specific implementation detail (RDBMS with TAP for example).

Now that we are decoupled from the implementations details related to RDBMS we need a way, similar to TAP_SCHEMA but not related to relational databases, to describe our view schema. It turns out that the VOTABLE standard is a very good candidate, allowing the description of tables, in its broadest sense. And in particular in the meaning of our view (a kind of virtual table). This approach has been introduced as part of the VO-DML standard, through the VOTable-Utype DM serialization.

In this approach, the view is associated with the TABLE element in the VOTABLE standard and the columns with the FIELD elements.
So, we have abstracted the table and its description from the underlying implementation.
Instead of the tightly coupled to RDBMS approach:
`table  <->  column  <-> TAP_SCHEMA`
we have the more abstract and generic:
`view   <->  field   <-> VOTABLE`

NOTE: Other semantic concepts mapping may be done by the user/reader to apply to its own problem domain, for example:

```
view          <->    field          <->   row
view          <->    column         <->   row
collection    <->    axis           <->   object
virtual table <->    virtual column <->   row
service       <->    parameter      <->   result
```