International

Virtual

Observatory

Alliance

Simple Numerical Access Protocol (SNAP) for theoretical data

Version 0.1

IVOA Technical Note 2006/09/14

This version:
Snap0.1-20060914
Latest version:

Previous version(s):

Author(s):
Claudio Gheller
Gerard Lemson
Laurie Shaw
Hervé Wozniak

Supprimé : e

Abstract

This specification defines a protocol for retrieving data coming from numerical simulations from a variety of data repositories through a uniform interface. The interface is meant to be reasonably simple to implement by service providers. A query defining the interesting physical models is used for searching for candidate simulations and related data. The service returns a list of the candidate simulations. The service can be further queried in order to get information on data associated to interesting simulations. Finally, data can be further selected, choosing specific quantities and extracting rectangular sub-samples from the simulated volumes. Data are be returned in VOTable simulation specific format, with support of external binary file management and data staging.

Status of This Document

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress".

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/Documents/.

Contents

1 Introduction

This specification defines a prototype standard for retrieving theoretical data from a variety of astronomical simulation repositories. These data can be the outcome of different kinds of numerical applications, like dynamical simulations, semianalytical models, montecarlo simulations etc.
However, the **Simple Numerical Access Protocol** (hereafter ***SNAP***) is designed to address numerical simulation outputs organized as follows. Time can be explicit (e.g. snapshots of an N-body simulation) or implicit (e.g. in case of a montecarlo realization). For each timestep, the information must be sampled in a 3D space called "volume" hereafter, and position in this volume are called x, y and z for reference to the configuration space. The sampling can be regular (e.g. cartesian mesh) or irregular (e.g. particle or adaptive mesh position). Each mesh/particle position in the 3D space hosts the same physical quantity (i.e. mass, density, velocity, etc) for each timestep.
Theoretical data can be classified according to the following *Levels*:

- **Level 0**: direct outcome of the simulation. Examples are the coordinates and velocities of files in an N-Body simulation, the density field on the computational mesh of a Jet simulation etc.
- **Level 1**: data extracted or derived from the simulation results, having the same characteristics of the simulation results themselves. For example, the coordinate of the points that build up a galaxy cluster extracted from a cosmological simulation using a friend of friends algortihm.[GL- Must they be similar to the simulation they were derived off, or to a typical SNAP simulation. Consider for example density fields due to binning of an N-Body result, or a "source catalogue" derived from a mesh simulation.]
- **Level 2**: results that have been obtained after an analysis process from Level 0 and Level 1 data. Examples are projected maps, statistical functions, Virtual Telescope applications.

SNAP is designed to deal with **Level 0** and **Level 1** data. SNAP specifies the following services:

- retrieval of the entire simulation outcome (the particle positions and velocities within the simulation box, or the physical quantaties at each grid point) – known as a snapshot – at one or more timesteps.
- retrieval of a specific subset of a simulation (e.g. all the particles/grid-points within a certain region)

The SNAP protocol, is designed primarily as a "data on demand" service, with dataset created on-the-fly by the service given the position and size of the desired output dataset as specified by the client. This is not a simple task for various reasons. First, simulations adopt specific units and coordinate systems, which depend on the nature of the problem, the characteristics of the algorithms and their implementation etc. Furthermore, there is nothing like a "position in the sky" as for astronomical images. Then, different simulation outputs can be represented by completely different data objects. For example, the output can consist in a set of particles or grid points in a given volume. Each particle or

mesh point has its physical position and a set of associated scalar and vector quantities, like velocity, mass density, temperature etc. On the other hand, mesh based simulations describe their data as discrete fields defined on a regular mesh. This may be cartesian (as for example, in cosmological simulations) or cilindrical (as in the case of jets simulations). SNAP has the goal of providing a uniform description of the selection service trying keep it simple and, at the same time, to include as many different kind of simulations and data as possible.

In operation, SNAP represents a negotiation between the client and the data service. The client searches for all the simulations available for a certain set of physical parameters (for example, the density parameters and the Hubble parameter for cosmological simulations) and the service returns a list, encoded as a XML file structured according to a data model defined in this specification (possibly VOTable), of the simulations that match the request. The client then examines the result to determine if it is interested in any of the available hits and possibly iterates with the service to refine the query. As soon as the interesting set of simulations is determined, the associated available dataset, quantities and, possibly, post-processed data, with all their basic features (units, size, dimensionality etc.), are proposed to the client. At this point, the end user can select some of the dataset, specify the quantities he/she is interested in and extract a sub-sample specifying a rectangular or spherical region inside the computational volume. Data (particles or mesh points) which fall inside the selected region are extracted and the result is delivered to the user as a VOTable. Since data size is usually large, specific care must be given to performance issues, both for in the elaboration and in the download phases. This is even a more serious issue, noticing that data collections are often distributed and the client may query multiple services simultaneously.

Supprimé : (and possibly

Supprimé :)

In summary, we can identify four main stages for the SNAP service.

1. **Search for available simulations and data** (Simulation Discovery, section 3)
   The query is on metadata
   The result is an XML document (maybe VOTable) with matching result metadata.
2. **Identification of subset of interest** (Subset Selection, Section 4)
   The user identifies a subset of the full simulation data which is of interest, which can used to select easily the region to focus on
   This subset is defined both in time and in space.
3. **Snap request** (Section 5)
   Send to the server the selection parameters for the Snap action
4. **Data staging and delivery** (Section 6 and 7)
   Metadata are immediately delivered to the client as a VOTable
   Data are delivered (possibly after some time, needed for extraction) via HTTP, FTP as binary files.
   Delivery of VOTable and binary data files can be in two separated stages.

2 Requirements for Compliance

The keywords "MUST", "REQUIRED", "SHOULD", and "MAY" as used in this document are to be interpreted as described in RFC 2119 [34].

An implementation is compliant if it satisfies all the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant".

Compliance with this specification requires that a simple numerical access web service, HTTP and/or SOAP/WSDL, be maintained with the following characteristics:

1. **The service MUST support a Simulation Discovery service as described in section 3 below.**
   Available data can be retuned as the result of a query based on a series of physical and technical parameters which specify the requirement of the user. These parameters can be general or specific of the discipline or research field of interest. The research can be further refined or used to select the dataset of interest and proceed with following steps of the SNAP procedure.

2. **The Snapshot Retrieval (getSnap) web method MUST be supported as defined in section 7 below.**
   This method allows clients to retrieve single simulation snapshots and cutouts

3. **Sub-Volume/Sub-Set determination methods MAY SHOULD be supported as defined in section 4 below**
   Contrary to real-sky-oriented protocols, there is in most simulations no natural sub volume that a user might be *a priori* interested in. The SNAP service SHOULD provide methods to perform the selection of a simulate volume. Services which has only download capabilities MAY provide this fuctionality. Services which support data cutout MUST provide such function. Examples are: regions of a certain size of higher (or lower) than normal density, or sub-volumes containing particularly massive objects. For particle simulations, subsets of the particles can be defined for example by all particles in a particularly heavy cluster.

4. **The Sub-Volume Extraction method MAY be supported as defined in sections 5 and 6 below**

Supprimé :

This method allows clients to retrieve data from a spatially defined sub-volume of the simulation box. The client determines the rectangular or spherical region within the simulation, the bounds and scale (i.e. units) of which are specified in the simulation metadata, and the service returns the simulation data contained within this region. The service MAY use a *staging* method (section 6) to return the particle file, as extracting a sub-sample of particles or grid points from a larger simulation box is likely to be a time-consuming process and would thus require some kind of caching.

5. **The SNAP service MUST be registered by providing the information defined in section 8 below.**
   Registration allows clients to use a central registry service to locate compliant simulation access services and select an optimal subset of services to query, based on the characteristics of each service and the simulation data collections it serves.

6. **A job status request MAY be supported.**
   This method allows users to inquire about the status of a staged request. It MAY allow a user to cancel a request.

3 Simulation Discovery

To be done according to the data model.

4 Subset Selection

To be done

5 Snap request

The main target of the SNAP service is the access to the raw data from a simulation, selected by a general Simulation Query, described in section 4. The SNAP service in general provides the following functionalities:

1. Extraction of a subset of data properly selected
2. Storage of the associated metadata in a VOTable (see…)
3. Delivery of the result to the user via http, ftp etc. (see…)

In principle, the extraction phase 1 could be performed using any set of N parameters that characterize the simulation. However, for simplicity, in a first stage of development, we will focus on geometric selections, allowing the user to select a either rectangular or spherical sub region of the whole computational volume, without having to download the whole dataset. Nevertheless, retrieving the complete dataset is still possible. This can be seen as a degenerate Cutout request, with a region of interest which covers the entire computational volume.

However, notice that this action is not just a simple download, since action 2 is still performed.
In order to select the region of interest, **only geometric parameters** are necessary. E.g., for a rectangular region, the user has to specify the center of the box and the length of each of its sides. For a spherical selection, center and radius of the sphere are required. This information can be either in **physical units** or in **fractions** of the box. The latter is simpler to handle, but it is not always possible. In this case, in fact, the knowledge of length units is not required. Unit conversions are deferred to the server, where units are fully specified. Anyway, in order to make the selection more effective and intuitive for the user, length can be presented and set in physical units (it is converted in fractions by the client application). [GL – Should we allow the server to define the units that a request is using (implicitly) ? If not, we have to find a standard for representing units inside of the model and explicitly in the protocol. Only then can a server do unit transformations.]
The selection can be performed on one or more snapshots (each snapshot is a output at a different time) at a time. One or more variables of a given snapshot can be selected in the same Cutout operation. [GL – Need to add this to the spec below.]

The input query must consist in a position in the simulation box denoting the centre of the cubic (or spherical) sub-volume, and the side-length (or radius) of the cube (sphere). For regions that intersect the boundary of the simulation box, the service has the option of applying *periodic boundary conditions* (if applicable). The resulting file will be made available through an access URL, possibly using the *Snapshot staging* method, notifying the client when the sub-volume extraction has been completed and the resulting particle file is available for retrieval. This appears to be a necessary feature due to the rapidly increasing size of data files associated to the larger and large available computing power. Consequently, the processing time to extract requested volumes could be high, larger than a typical working session. Furthermore it is important to stress that, differently from what generally happens when retrieving observational images and data, simulation data cannot be retrieved via http with some kind of encoding for the binaries (e.g. base64) since this is extremely expensive, both for CPU and for size if large datasets are being handled. Therefore, matadata and data must be splitted in two different files, delivered in separate stages. Metadata are returned immediately as VOTables with references to external raw binary files. These files can be downloaded, as soon as they are available, using HTTP with no encoding, FTP, or even GridFTP. The data files in general are pure binaries. However, special formats (FITS, HDF5…) may be supported. [GL – We need to specify the mapping from the VOTable metadata to the HDF5 metadata elements. Is that trivial ? FITS has a standard mapping from VOTable .]

5.1 Input

In order to start a SNAP request, the following parameters must be specified and passed to the server.

1. Region of interest

An input Sub-Volume query must consist of an x,y,z position in the box, plus the side lengths (or radius) of the rectangular (spherical) region surrounding this point. These quantities can be specified either as fraction of the box or in their specific units.

The service MUST support the following two parameters:

POS

The position of the center of the region of interest, expressed as a set of three coordinates in fractions of the corresponding box side. A comma should delimit the three values; embedded whitespace is not permitted. Example: "POS=0.3,0.25,0.9". A NULL value represents the center of the whole box (0.5,0.5,0.5).

SIZE

The size of the sides (or the radius) of the region given either in fractions of the corresponding box side or in physical units. The region may be specified using either one or three values. If only one value is given it applies to all coordinate axes (alternatively it is the radius of the sphere). [GL: In that case, how do we decide whether it's a sphere or cube we want ? I guess we can add an extra parameter, like: SHAPE=BOX or SPHERE?]

The format of the SIZE parameter is the same as that for POS. Example "SIZE=0.2,0.5,0.3". A special case is SIZE=NULL, which represents the whole box.

SHAPE

The SHAPE parameter can be either BOX or SPHERE in order to specify the shape of the selected region [GL – This can be made redundant if we follow Herve's propsal that a single value for SIZE would imply a radius of a sphere, 3 values would indicate a box (a cube would also need 3 values]

In addition, the service MAY support the following query constraint which is used on the server side specifying the type of boundary conditions must be adopted for the region of interest:

BOUNDARY

Also this parameter can have one or three values, one for each coordinate direction. If only one value is given it applies to all coordinate axes. Possible values are:

- TRUNC – if the interesting region exceeds the computational box, it is resized at the box boundary
- PERIODIC - if the interesting region exceeds the computational box, data are selected from the opposite side of the box

Metadata of the *service* indicates whether periodic is supported.

2. Fields of interest

The user can specify the physical quantities he is interested in, which can be a subset of the available ones.

FIELDS

The service MAY support an optional parameter with the name FIELDS, the value of which is a comma separated list of field names corresponding to the data elements the simulation can return. If the parameter is not provided the default behavior is to return all fields.

3. File Format

The service MUST support a parameter with the name **FORMAT** to indicate the desired format or formats of the data referenced by the output table. The value is a comma-delimited list where each element can be any recognized MIME-type. These will be of the major type "data". [GL – What would it mean to have multiple formats specified ? That the user wants one of these ? All of them ? I'd vote for allowing only one, with a default of binary VOTABLE if not specified.]

Possible formats are:

data/raw_tabular
data/raw_sequential
data/votable
data/hdf5
data/fits

This formats must be further specified…

4. Table Verbosity.

The service MAY support an optional parameter with the name **VERB** (denoting "verbose") whose value is a nonnegative integer. This parameter indicates the desired level of information to be returned in the output table, particularly the number of columns to be returned to describe each image. The following guidelines are recommended for determining which columns should be output at different verbosity levels:

- 0 - The output table should contain only the minimum columns required by section 5.2.
- 1 - In addition to level 0, the output table should contain columns sufficient for uniquely describing the image.
- 2 - In addition to level 1, the output table should contain, if possible, columns that contain values for all parameters supported as query constraints.
- 3 - The output table should return as much information about the images as possible. A table metadata query automatically implies the highest level of verbosity.

Services that do not support this parameter MUST permit it to be present without error.

[GL: I was never very happy with this term. It is very fuzzy. In the old protocols (cone search, SIAP it exists. SSA does not have it, and I think we should be more compatible with that spec. SSA has many more possible query parameters though. Here we can add support for indicating what attributes should be returned/available. For example, position, but not velocity, temperature … This is much more flexible in simulations than for spectra, where the main issue is in what units to return the fluxes/intensities. In that sense it is closer to cone searches, except that that protocol has no model whatsoever behind it.]

5. Service-Defined Parameters.
The service MAY support additional service-specific parameters. The names, meanings, and allowed values are defined by the service. The names need not be upper-case; however, they should not match any of the reserved parameter names defined above. Values must be simple and describable through the output of the **metadata query** as described below.

5.2 Successful Output

The output returned by a SNAP Query [GL: query for discovery, or staging request of a cutout ?] is a VOTable, an XML table format, returned with a MIME-type of text/xml. The table lists all the data files available to the client that match the query constraints. The following requirements are placed on the contents of the table when the query successfully returns a list of files:

Supprimé : s

1. The VOTable MUST contain a RESOURCE element, identified with the tag type="results", containing a single TABLE element which contains the results of the query. The VOTable is permitted to contain additional RESOURCE elements, but the usage of any such elements is not defined here. If multiple resources are present it is recommended that the query results be returned in the first resource element.
2. The RESOURCE element SHOULD contain an INFO with name="QUERY_STATUS". Its value attribute should set to "OK" if the query executed successfully, regardless of whether any matching images were found. All other possible values for the value attribute are described in section 5.3 below.
3. [GL – I think this belongs in “Succesful ouput” section of 3. I think I am supposed to write something about that there.] Each table row represents a different field [GL: Snapshot ? Simulation ?] available to the client. [GL: this makes me think this more properly belongs in the *Simulation Discovery* section. Each entry in the result of a simulation query would refer to a simulation/snapshot, the columns will contain metadata attributes, described by the FIELD elements. When retrieving a snapshot/cutout the columns will contain the data, possibly as arrays, FIELD contains the name of the variable (“temperature”, “X”). It would seem to me that a SNAP result, when serialised as binary VOTable will have the metadata describing a particular simulation/snapshot/cutout as a set of PARAM elements.]

4. Each TABLE in the output VOTable MUST contain FIELDs where the UCDs that follow have been set. These attributes are required to be able to return metadata separately from the data itself.
5. FIELDS refer to the variables stored in the external binary file. The variables can be organized both as tables (all quantities related to a particle or a grid point stored one after the other – sort of struct) and as complete sequences (all the values of a single field stored as a sequence, followed by the next field). In the former case, the order of the FIELDS in the table represents the order of columns in the table. In the latter, it represents the order in which fields are stored one after the other. If mixed data (mesh and particles in the same file) are requested only the second solution is possible.[GL – Could we insist that in such cases the service produces these differenmt types of datasets in different TABLEs, possibly in the same VOTABLE. I'd be in favour of that. Different objects in different datasets, but maybe someone can indicate whether such mixed datasets are common ?] The adopted ordering is specified in the TABLE field by the keyword *order*. Possible values are "tabular" and "sequential"[GL – Does this require an update of the VOTable schema ? *order* is not a VOTable keyword. We may use another element for that, though do we really need it ? VOTable already allows the different usages and I suppose it is implicit from the *array* attributes in the FIELDs how this is used ?]
6. Variables must be scalars, i.e. vectors (or more general multidimensional quantities) are not supported. This means that each FIELD represents a scalar value. E.g. temperature of each point, x coordinate of a particle.
7. Each FIELD must specify the datatype, the arraysize and the unit of the variable. Furthermore name, ID, and ucd has to be set. The ucds for simulations are still in progress, therefore we do not enter in more details.
8. Each field must specify the geometry parameter, which at present can have the values "n-body", "mesh" and "amr". [GL – This again would require an update of the VOTable schema. Instead we could make it a PARAM or INFO element. Also, in case we do not mix the files per table, we need only on such declaration per TABLE.]
9. The binary data filename is specified in a DATA section, according to the rules defined in other documents (e.g. SIAP specification)

Other parameters may be supported according to the services offered by the data provider.

Mis en forme : Police :Italique

Mis en forme : Police :Italique

Examples

1. VOTable for the velocity field of a fluid on a fixed 3D mesh

[GL – We still need a proper way I guess of indicating what the spatial dimensions are for a representation like this. FITS has its WCS system for implicitly specifying the spatial coordinates of a multidimensional array. Is something like this in existence for VOTable ? We need to inquire.]

```
<RESOURCE name="myVectorField" type="results" >
   <DESCRIPTION>Velocity Field from N-Body run</DESCRIPTION>
```

```
   <INFO name="QUERY_STATUS" value="OK"/>

   <TABLE name="VelocityField" ID="Vel" order="sequential">
      <FIELD name="vx" ID="vx1" ucd="phys.veloc;pos.cartesian.x"
datatype="float"
             arraysize="41x41x41"   unit="km/s" geometry="mesh" />
      <FIELD name="vy" ID="vy1" ucd="phys.veloc;pos.cartesian.y"
datatype="float"
             arraysize="41x41x41"   unit="km/s" geometry="mesh" />
      <FIELD name="vz" ID="vz1" ucd="phys.veloc;pos.cartesian.z"
datatype="float"
             arraysize="41x41x41"   unit="km/s" geometry="mesh" />
      <DATA>
        <BINARY>
          <STREAM href="file:///scratch/myhome/test.bin"/>
        </BINARY>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

2. VOTable for the velocity and position fields of particles from an N-Body simulation

```
<RESOURCE name=myParticles type="results">
   <INFO name="QUERY_STATUS" value="OK"/>
   <TABLE name="Particles" ID="NBody" order="tabular">
      <FIELD name="x" ID="x1" ucd="pos.cartesian;pos.cartesian.x"
        datatype="float" arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="y" ID="y1" ucd="pos.cartesian;pos.cartesian.y"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="z" ID="z1" ucd="pos.cartesian;pos.cartesian.z"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="vx" ID="vx1" ucd="phys.veloc;pos.cartesian.x"
        datatype="float"arraysize="100000"   unit="km/s"
geometry="particles" />
      <FIELD name="vy" ID="vy1" ucd="phys.veloc;pos.cartesian.y"
        datatype="float"arraysize="100000"   unit="km/s"
geometry="particles" />
      <FIELD name="vz" ID="vz1" ucd="phys.veloc;pos.cartesian.z"
        datatype="float" arraysize="100000"   unit="km/s" />
      <DATA>
        <BINARY>
          <STREAM href="file:///scratch/myhome/test.bin"/>
        </BINARY>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

3. VOTable for the temperature field of a mesh based quantity and the position of N-Body particles extracted from the same spatial region.

```
<RESOURCE name=myMixedData type="results">
   <INFO name="QUERY_STATUS" value="OK"/>
   <TABLE name="ParticlesAndMesh" ID="NBody" order="sequential">
      <FIELD name="x" ID="x1" ucd="pos.cartesian;pos.cartesian.x"
        datatype="float" arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="y" ID="y1" ucd="pos.cartesian;pos.cartesian.y"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="z" ID="z1" ucd="pos.cartesian;pos.cartesian.z"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="temperature" ID="temp"
ucd="phys.temperature;pos.cartesian.x"
        datatype="float"arraysize="41x41x41"   unit="K" geometry="mesh"
/>
      <DATA>
        <BINARY>
          <STREAM href="file:///scratch/myhome/test.bin"/>
        </BINARY>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

An alternative here is:

```
<VOTABLE>
  <RESOURCE name=myMixedData type="results">
   <INFO name="QUERY_STATUS" value="OK"/>
   <TABLE name="Particles" ID="NBodyParticles" order="sequential">
      <FIELD name="x" ID="x1" ucd="pos.cartesian;pos.cartesian.x"
        datatype="float" arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="y" ID="y1" ucd="pos.cartesian;pos.cartesian.y"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <FIELD name="z" ID="z1" ucd="pos.cartesian;pos.cartesian.z"
        datatype="float"arraysize="100000"   unit="Mpc"
geometry="particles" />
      <DATA>
        <BINARY>
          <STREAM
href=_mesh"file:///scratch/myhome/test_particles.bin"/>
        </BINARY>
      </DATA>
    </TABLE>
   <TABLE name="Mesh" ID="NBodyMesh" order="sequential">
      <FIELD name="temperature" ID="temp"
ucd="phys.temperature;pos.cartesian.x"
        datatype="float"arraysize="41x41x41"   unit="K" geometry="mesh"
/>
      <DATA>
        <BINARY>
          <STREAM href="file:///scratch/myhome/test.bin"/>
```

```
        </BINARY>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>

[GL – Do we need an example of an “ordinary” tabular VOTable as well ?
Something like
<RESOURCE name=myParticles type="results">
   <INFO name="QUERY_STATUS" value="OK"/>
   <TABLE name="Particles" ID="NBody" >
      <FIELD name="x" ID="x1" ucd="pos.cartesian;pos.cartesian.x"
        datatype="float" unit="Mpc" />
      <FIELD name="y" ID="y1" ucd="pos.cartesian;pos.cartesian.y"
        datatype="float" unit="Mpc" />
      <FIELD name="z" ID="z1" ucd="pos.cartesian;pos.cartesian.z"
        datatype="float" unit="Mpc" />
      <FIELD name="vx" ID="vx1" ucd="phys.veloc;pos.cartesian.x"
        datatype="float" unit="km/s"/>
      <FIELD name="vy" ID="vy1" ucd="phys.veloc;pos.cartesian.y"
        datatype="float"  unit="km/s" />
      <FIELD name="vz" ID="vz1" ucd="phys.veloc;pos.cartesian.z"
        datatype="float"  unit="km/s" />
      <DATA>
        <BINARY>
          <STREAM href="file:///scratch/myhome/test.bin"/>
        </BINARY>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>

]
```

6 Data Staging

By Data Staging we refer to the processing the server performs to retrieve or generate the requested simulation volume or subvolume from a similar box and cache them in online storage for retrieval by a client. Staging is necessary for large archives which must retrieve simulation data from hierarchical storage, or for services which can dynamically extract subvolumes, where it may take a substantial time (e.g., minutes or hours) to retrieve the particles in the relevant region of the simulation box. Issuing a staging request for a set of simulation subvolumes (e.g. for a set of small cubes randomly placed in a simulation box) also permits large servers to optimize subvolume extraction, for example to take advantage of parallelization for large requests.

The snapshot staging service is optional for the simulation server. If staging is not implemented, data should be immediately available for retrieval (URL direct to file). [GL – How do we indicate that this is the case ? Service metadata ?]

When staging of data is necessary, the technique used is to stage data on the server for later retrieval by the client; the data is only staged for a period of time

and is eventually deleted by the service. This therefore permits the getSnap method to be identical whether or not staging is used. The service can proceed to generate the simulation sub-volume regardless of the state or accessibility of the client.

The service provider can decide how to deal with multiple requests from the same user or large number of requests at the same time from different users. Queue, batches, limits are defined by the provider. The provider is only requested to publish and notify such features to the registry service (details to be defined [GL – Ask GWS working group ?]).

As soon as staged data are available at the given URL, the user can start the download procedure. The user can be informed of the availability of the data following two different approaches:

- The client searches for information on the service. [GL – Requires protocol, see proposal in section 2]
- The service searches for the client and, if present, sends information to it. [GL – Requires authentication]

The first approach is simpler. In its most strightforward implementation, it simply consists in making the client reload the data URL, to see if data are there.
In the second approach, the staging mechanism should provide a **messaging** capability. The service broadcasts messages to subscribing clients whenever a staging (processing) event occurs, such as when the sub-volume extraction has been completed and is available for retrieval. Service generated messages can also be used to pass informational or diagnostic messages to clients as processing proceeds. This type of messaging is asynchronous and one way: the service broadcasts *messages* to subscribing clients as things happen, whereas clients send *requests* to the service to invoke web methods. For example, to initiate staging, subscribe to staging-related messages, or abort a staging operation in progress, the client sends a request to (invokes a web method provided by) the service.
In the second approach, a unique identification of the client is required, so that in different session the user is always characterized by a unique id. This is required in some implementation of the first approach. This requires the introduction of certificates and related stuff, which could reduce the usability of the service. At the moment, simpler solution are suggested.

7 Data Delivery

The snapshot retrieval request (**getSnap** web method) allows a client to retrieve a single raw simulation file given an access reference or "acref" as returned by a prior simulation query. The file can contains more than one variable and can be in the formats defined in Section 5.
All the metainformation about the content and the structure of the data file are stored in the associated VOTable (see Section 5).
The retrieved data file is in a binary format as described in Section 5.

7.1 Input

The input to the getSnap web method is the simulation acref for the indicated raw simulation data or extract subvolume. The acref for a particular file is obtained through a prior call to the Simulation Query web method.

7.2 Successful Output

The output of getSnap MUST be a single data file returned with a MIME-type identifying the file format. The primary type of the MIME code should be "data/". Other MIME types are not expected.

7.3 Error Response

If a condition is encountered that prevents the requested image from being downloaded, the output MUST be a VOTable with a single RESOURCE element containing an INFO child with name="QUERY_STATUS". The allowed values for this INFO are the same as those defined for the Image Query; in addition, the following additional attribute MAY be supported:

DEFERRED

This indicates that the requested image is not yet available for some reason but that it will be at some time in the future. Clients that receive this type of message can periodically try (poll) the given acref URL until the image becomes available.

8 Service Registration

To be done…

Appendix A: “Appendix Title”

References

[1] R. Hanisch, *Resource Metadata for the Virtual Observatory* , http://www.ivoa.net/Documents/latest/RM.html
[2] R. Hanisch, M. Dolensky, M. Leoni, *Document Standards Management: Guidelines and Procedure* , http://www.ivoa.net/Documents/latest/DocStdProc.html