



International

Virtual

Observatory

Alliance

S3: Proposal for a simple protocol to handle theoretical data (microsimulations)

Version 0.1

IVOA Note 2008 October 15

Interest/Working Group:

<http://www.ivoa.net/twiki/bin/view/IVOA/IvoaTheory>

Author(s):

Carlos Rodrigo, Miguel Cerviño, Enrique Solano, Patrizia Manzato

Abstract

The aim of this document is to suggest a new protocol designed to provide access to theoretical data/services in the framework of the Virtual Observatory.

We call it “Simple Self-described Service” protocol as it is based in the ability of the data server to describe itself in a simple standardized way.

Status of This Document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of [current IVOA Recommendations and other technical documents](http://www.ivoa.net/Documents/) can be found at <http://www.ivoa.net/Documents/>.

Acknowledgements

The authors belong to the Spanish Virtual Observatory, a project supported from the Spanish MICINN through grant AyA2008-02156.

Contents

1 Introduction.....	2
1.1 Motivation.....	2
1.2 Requirements.....	3
1.2.1 Simplicity.....	3
1.2.2 Self-description.....	4
2 Service interface.....	5
2.1 Metadata.....	5
2.2 Data Query.....	8
2.3 Retrieve file.....	10
3 Model credits and references	13
4 References.....	14

1 Introduction

The aim of this document is to suggest a new protocol designed to provide access to theoretical data/services in the framework of the Virtual Observatory.

We call it “Simple Self-described Service” protocol as it is based in the ability of the data server to describe itself in a simple standardized way. We refer to the protocol, in what follows, as S3.

1.1 Motivation

Theoretical models are widely used in Astronomy. Physical properties of an object can be, for instance, inferred by comparing the observed spectra with a collection of theoretical spectra. There is also a large set of theoretical models that, having no direct observational counterpart, can be used too to infer physical properties of observed systems. The inclusion of such kind of models in the VO framework is fundamental for the development of analysis tools.

A large number of libraries of theoretical models are presently available in the Internet. They can be downloaded as a collection of data files with, in some cases, the help of a web form allowing a previous selection of the files of interest. The results are usually presented in different formats as ASCII or FITS files.

This scenario forces the user to perform a previous work in order to be able to infer physical properties of the observational data from theoretical models. The situation is even worse if several sets of theoretical models, with different formats and outputs, are used. This lack of homogeneity makes it difficult to design automatic tools to simultaneously work with different models and almost impossible to develop applications able to use the models on the fly.

One of the aims of the Virtual Observatory is to guarantee full interoperativity not only among observational data but also between them and theoretical data. However, there are a number of issues that make it difficult to achieve this goal. One of them is the fact that so far VO has clearly focused on observational data. This can be seen, for instance, in the main protocols and standards already developed: SSAP, SIAP, ConeSearch,..., all of them require the object position or name as mandatory, parameters that are meaningless when dealing with theoretical models. Another difficulty is that, depending on the physical problem to be tackled, the particular approach taken by the author of the model or even his/her own personal preferences, the set of parameters used to characterize the model usually differs from one to another. This makes it difficult to define a general data model for theoretical data, a major topic for the IVOA and Euro-VO Theory Groups.

So far, two different approaches have been proposed to tackle this problem:

- SimDB/SimDAP is a solution that is being developed with the main objective of handling complex simulations as, for instance, cosmological ones.
- The SSAP standard includes an use case for theoretical spectra (what was formerly know as TSAP) so that this kind of simulations can be handled in a similar way as the one for observed spectra.

The approach that we propose in this document is very similar to the one already present in SSAP. Actually we try to follow the same philosophy and generalize it to include other kind of theoretical data (isochrones, evolutionary tracks, etc.).

1.2 Requirements

1.2.1 Simplicity

A microphysics model is often developed by a small team, focused on science, not computing. They want to make their model available in the VO because they understand that it can be useful for other people, for instance, to infer physical properties of objects from observed data, and also to give more visibility to their model.

In fact, they would usually prefer to develop their own server by themselves, so that they can make available new versions as they are ready, correct errors as they are found, refine details...

But... there is a good chance that they don't have time, money or will to study long and complex protocol definitions or to invest much time (or people) in developing a complex service.

That's why we think that we should give the authors a protocol as simple as possible to make available their own models in the Virtual Observatory. The simpler the development of the service is, the more people will be willing to implement it and, thus, more theoretical models will be available in the VO.

This does not mean that more complex solutions can be developed to address complex models or advanced functionalities, but having a simple protocol can reduce the gap between scientific work and VO implementation that prevent most model developers to make their models available in the VO.

1.2.2 Self-description

A theoretical model is not related with a real object or with spatial coordinates. Instead, it is characterized by a set of parameters and the allowed values for each of them. Actually, those parameters and values are not the same for different models. Even models describing similar physics are often characterized using different types of parameters. This is not surprising. Each scientist develops his/her model focusing on the specific physical problem that he/she wants to address and there are reasons why each developer has chosen to characterize his model using a particular set of parameters.

It does not seem to be a good idea to predefine in a protocol how a scientist must characterize his model. It seems much better to give useful guidelines on how to describe that characterization.

Thus, we think that a useful protocol for microsimulations should be based on the service self-description approach.

The server offering the model must describe itself as clearly as possible. In particular, it must provide information on:

- What kind of model is being offered.
- What parameters characterize the model (what kind of queries can be done).
- What is the physical meaning of those parameters.
- What kind of results can be retrieved.

Moreover, the protocol must explain how an application/user can:

- Obtain that self-description in a standardized way.
- Build a viable query to the server.

2 Service interface

A service using S3 must be able to answer three different kind of queries:

1. **Metadata:** What parameters define your service?
2. **Data query:** What files do you have for given values (or ranges) of the parameters?
3. **Retrieve file:** Give me a particular file.

Each question will be done as an http request of the form:

`http://<server-address>/<path>?<extra-GET-arg>&...`

Examples:

<http://myservice.com/script?model=mymodel&...>

<http://myservice.com/s3.php?...>

In what follows we will use <http://myservice.com/s3.php?> as the base address for the service.

The main procedure to retrieve data can be visualized as a dialog between the user (or client program) and the model server.

- First the user asks the server for the description of the model that it offers. In particular, which parameters are available for queries (“metadata query”).
 - The server answers with a list of the parameters available for the model.
- Second, the user sends a query to the server (using the parameters listed in the metadata answer) searching for particular values (or ranges) of the available parameters (“data query”).
 - The server answers with a list of results for the search, including the link to retrieve each particular data file.
- Third, the user uses those links to retrieve the files in which he is interested (“retrieve data” query).

2.1 Metadata

The service must be able to answer a query with only one additional parameter:

<http://myservice.com/s3.php?format=metadata>

The answer must be a VOTable containing;

- A RESOURCE element with type="meta", containing:
- A DESCRIPTION element with a human readable description of the service.
- a PARAM element for each of the parameters that are accepted for the query.
 - The name attribute for each of these PARAMs should be "INPUT:queryname" where "queryname" will be the name used for this parameter in subsequent queries.
 - Optionally, additional properties can be added to the name in the following way: "INPUT:queryname:properties".
 - Optionally, a VALUES element can be part of the PARAM element, specifying the valid range of values for this parameter or even a complete list of the values accepted by the server.
 - The VALUE attribute of the PARAM element, if not empty, will be the default value used by the server for that parameter if another value is not specified in a subsequent query
- Optionally, other PARAMs explaining characteristics of the service (curation, provenance, credits, etc). Their names must not start with INPUT: as they are not query parameters.
- Nothing else (no TABLE, no FIELDS, no TABLEDATA...).

The *properties* optional part of the PARAM name can take the following values:

- range: a range of values is accepted in queries
- list: a comma separated list of single values is accepted in queries
- required: the parameter is mandatory in queries
- a comma separated list of all above, for instance
<PARAM name="INPUT:param1:range,required">
- fixed: the parameter must be included in queries with the value specified in the VALUE attribute of the PARAM.

It must be noted that the parameters that define the service are not restricted to the physical properties of the model (effective temperature, gravity...). Any other option offered by the service can be specified in this way. For instance, the model server can give options related to the method of calculation preferred for the model (that could be even computed on the fly using those options) or different available ways of generating the final results. In those cases, it is enough that those options are specified using the VALUES element so that the user can choose among them.

All this information is enough to make a query to the service to find out what actual files are available.

An example of valid "metadata" response would be as follows:

Example:

A simple response could be:

```

<VOTABLE version="1.1"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1">

<RESOURCE type="meta">
  <DESCRIPTION>
    Theoretical Isochrones for the MyModel model.
  </DESCRIPTION>
  <PARAM name="INPUT:age" ucd="phys.age" unit="Gyr">
    <DESCRIPTION>
      Age of the star in Gyr.
    </DESCRIPTION>
  </PARAM>
  <PARAM name="INPUT:metallicity" ucd="..." unit="">
    <DESCRIPTION>
      Metallicity of the star defined as Fe/H
    </DESCRIPTION>
  </PARAM>

</RESOURCE>
</VOTABLE>

```

This example VOTable says that two parameters are accepted in queries, being “age” and “metallicity” their names.

Example:
a more detailed one:

```

<VOTABLE version="1.1"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1">

<RESOURCE type="meta">
  <DESCRIPTION>
    Theoretical Isochrones for the MyModel model.
  </DESCRIPTION>
  <PARAM name="INPUT:age:range" ucd="phys.age" unit="Gyr">
    <DESCRIPTION>
      Age of the star in Gyr.
    </DESCRIPTION>
    <VALUES type="actual">
      <OPTION value="1"/>
      <OPTION value="2"/>
      <OPTION value="12"/>
    </VALUES>
  </PARAM>
  <PARAM name="INPUT:metallicity" ucd="..." unit="">
    <DESCRIPTION>
      Metallicity of the star defined as Fe/H
    </DESCRIPTION>
    <VALUES type="actual">
      <OPTION value="0"/>
      <OPTION value="0.5"/>
      <OPTION value="1"/>
    </VALUES>
  </PARAM>
  <PARAM name="INPUT:logg:fixed" ucd="phys.gravity" unit="" value="1"/>

</RESOURCE>
</VOTABLE>

```

This example VOTable says that two parameters are accepted in queries, being “age” and “metallicity” their names. The first one can be specified as a range of values in the queries, and the available values are specified as (1,2,12) for “age” and (0,0.5,1) for “metallicity”. Another parameter, “logg” must be included in queries with a fixed value of “1”.

Final user interaction

In most cases, this server/application dialog will need the interaction with a human user. The application asks the server for its self-description and uses the information included in the metadata VOTable to build a form so that the final user can choose the values of the parameters that fit his/her needs.

Once the user has chosen the values and submit the form, the application use those values to build a new query to the model server.

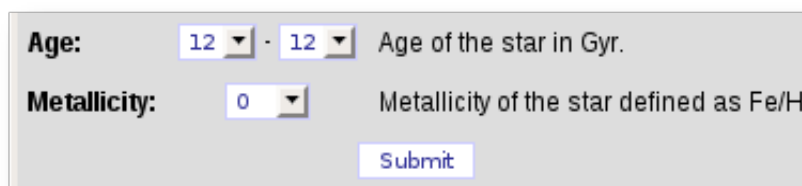
This is why it is important that the PARAMs include a human readable DESCRIPTION so that the application can show it in the form to help the user to understand the physical meaning of each parameter.

The protocol does not require such a form to be build (maybe the application could be able to choose values for the parameters in an automatic way without human interaction) and does not specify how the form must be build from the metadata.

However, it is easy to imagine some straightforward ways to do it. For instance, we could give some clues in terms of a web interface:

- A PARAM containing a VALUES element listing all the available values for that parameter, could be translated into a SELECT input field.
- A PARAM containing no VALUES element could be translated into a text input field so that the user writes the preferred value.
- A PARAM allowing “range” in its properties could be translated into two different input fields, one for the minimum value desired by the user and one for the maximum.
- A PARAM shown as “fixed” in the properties could be translated to a type=”hidden” input field in the form.
- ...

A simple example of a form that could be generated using the metadata VOTable of the example above could be:



Age: 12 - 12 Age of the star in Gyr.
Metallicity: 0 Metallicity of the star defined as Fe/H
Submit

Recursive metadata dialog

Although, for the sake of simplicity, a single step metadata is recommended, it can be built as a multi-step recursive dialog. Whenever an application accessing the service receives a VOTable containing a RESOURCE of type="meta", it must interpret that VOTable as a metadata description, choose values for the INPUT parameters specified in that VOTable and build a new query.

In other words, when a model server receives a query, it analyzes if enough parameters are specified in it for building a list of results. If not, the server answers with a metadata VOTable telling the application what other parameters must be specified.

An oversimplified example of this case would be that of a server offering isochrones for two different ages: 10 and 20 Gyr, and different metallicities depending on the age (let's say, metallicity=0,1,1.5 for age=10 Gyr and metallicity=0.2,0.5,0.7 for age=20 Gyr). Although this information could be shown in a single metadata VOTable, it is reasonable if the server produces it in two steps: first, a metadata VOTable with only one INPUT parameter, the age and second, when the age is chosen, another VOTable with the available values for the metallicity for that particular age.

Eventually, when the server has enough information for building a list of results, the query will be answered with a VOTable containing a RESOURCE of type="results" (as explained below) and that means the end of the *metadata dialog*.

2.2 Data Query

A S3 service must be able to answer a query build as:

<http://myservice.com/s3.php?param1=value1¶m2=value2...>

where:

- param1, param2,... are the parameters that the service accept for the queries and that the service itself lists in the format=metadata query explained above.
- Value1, value2... can be specified as single values, ranges or lists of values.
 - A range of values can be written as value1/value2 (starting and end of interval separated by "/")
 - A list of single values as value1,value2,value3 (values separated by commas)

The answer to this query must be a VOTable with:

- An INFO element with name="QUERY_STATUS".
 - If the query is ok, it will contain a "value"="OK" attribute.
 - If there has been some error, it will contain a value="ERROR" attribute and a child DESCRIPTION element explaining the reason of this error.
- If the query is ok, a RESOURCE element with type="results", containing:
 - A DESCRIPTION element with a human readable description of the service.
 - Optionally, one PARAM element for each of the values used in the query (this is recommended but not mandatory).
 - Other optional PARAMs.
 - A TABLE element containing:
 - A FIELD element for each column that describes the results.
 - A FIELD element with name="link" (the corresponding <TD> will contain the link to retrieve the actual file for each result).
 - A FIELD element with name="format". The corresponding <TD> will contain the mime-type of the file that would be downloaded by the *link* above.
 - A TABLEDATA element containing:
 - All the TR and TD elements for each result.

(Both the *link* and *format* fields, and the corresponding columns, are only mandatory when there is the need to point to a final data file. However, in some cases, this is not necessary because all the relevant information is provided in this "data query" step. If that is the case, the *link* and *format* fields and columns could be absent)

If the query status is "error" no RESOURCE element is necessary.

Example:

A possible query for the case shown in the previous example would be:

```
http://myservice.com/s3.php?age=0/10&metallicity=0,1
```

In this case, the client is asking the server for all files with an age between 0 and 10 Gyr and a metallicity either 0 or 1.

The model server should answer with a VOTable that could be similar to this one:

```
<VOTABLE version="1.1"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1">
<INFO name="QUERY_STATUS" value="OK"/>
<RESOURCE type="results">
```

```

<DESCRIPTION>
  Theoretical Isochrones for the MyModel model.
</DESCRIPTION>
<PARAM name="age" ucd="phys.age" unit="Gyr" value="0/10">
  <DESCRIPTION>
    Input value for the age of the star in Gyr.
  </DESCRIPTION>
</PARAM>
<PARAM name="metallicity" ucd="..." unit="" value="0,1">
  <DESCRIPTION>
    Input value for the metallicity of the star defined as
    Fe/H
  </DESCRIPTION>
</PARAM>
<TABLE>
  <FIELD NAME="age"/>
  <FIELD NAME="metallicity"/>
  <FIELD NAME="format"/>
  <FIELD NAME="link"/>
  <DATA>
  <TABLEDATA>
  <TR>
    <TD>1</TD>
    <TD>0</TD>
    <TD>application/x-votable+xml</TD>
    <TD><![CDATA[http://myservice.com/s3.php?id=12]]></TD>
  </TR>
  <TR>
    <TD>1</TD>
    <TD>1</TD>
    <TD>application/x-votable+xml</TD>
    <TD><![CDATA[http://myservice.com/s3.php?id=14]]></TD>
  </TR>
  <TR>
    <TD>2</TD>
    <TD>0</TD>
    <TD>application/x-votable+xml</TD>
    <TD><![CDATA[http://myservice.com/s3.php?id=121]]></TD>
  </TR>
  <TR>
    <TD>2</TD>
    <TD>1</TD>
    <TD>application/x-votable+xml</TD>
    <TD><![CDATA[http://myservice.com/s3.php?id=7]]></TD>
  </TR>
  </TABLEDATA>
</DATA>
</TABLE>
</RESOURCE>
</VOTABLE>

```

In this VOTable the client would see that there are four data files available for an age between 0 and 10 Gyr and a metallicity value of 0 or 1. For each case, a link is provided so that the client can retrieve the actual file, that is in VOTable format.

An example of error response could be:

```
<VOTABLE version="1.1"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1">
<INFO name="QUERY_STATUS" value="ERROR">
<DESCRIPTION>Not enough parameters for the query. Please, specify age and
metallicity</DESCRIPTION>
</INFO>
</VOTABLE>
```

2.3 Retrieve file

The protocol does not specify how to build a query to retrieve a particular data file offered by the model server. The user must make first a “data query” as explained above and then use the URL’s provided by the server to retrieve the actual files.

In fact, it must be noted that the file could either exist previously in the data server (and, thus, the link could point directly to the file) or it could be generated on the fly by some computer code for the case required by the user. There is no restriction about that.

These final results are not required to be VOTables. The service can offer its outputs in other formats (FITS, PNG, ASCII...) and, if that is the case, it must be specified in the “format” field in the previous step. If the model server gives the final results in any other format besides VOTable, the links provided in the “data query” are expected to download the final files in that format.

However, the VOTable format is preferred as it is designed to provide all the necessary information in a VO environment in a standardized way. All that follows apply only if the VOTable format is the used one.

The answer must be a VOTable containing:

- An INFO element with name="QUERY_STATUS".
 - If the query is ok, it will contain a “value”=“OK” attribute.
 - If there has been some error, it will contain a value="ERROR" attribute and a child DESCRIPTION element explaining the reason of this error.
- If the query is ok, a RESOURCE element with type="data", containing:
 - A DESCRIPTION element with a human readable description of the data.
 - One PARAM element for each of the values that describe the data file.
 - Other optional PARAMs.
 - A TABLE element containing:
 - A FIELD element for each column in the data. It is recommended that it contains a DESCRIPTION element with a human readable description of the column (This is

specially important and mandatory if the actual data is presented as TABLEDATA).

- The actual data, that can be:
 - A TABLEDATA element containing all the TR and TD elements for each row and column of the data.
 - A FITS serialization as described in the VOTable standard.
 - A BINARY serialization as described in the VOTable standard.

Using the query provided in the previous example for age=1,metallicity=0

<http://myservice.com/s3.php?id=12>

The server should answer with a VOTable similar to the following one:

```
<VOTABLE version="1.1"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1">
<INFO name="QUERY_STATUS" value="OK"/>
<RESOURCE type="results">
  <DESCRIPTION>
    Theoretical Isochrone for the MyModel model.
  </DESCRIPTION>
  <PARAM name="age" ucd="phys.age" unit="Gyr" value="1">
    <DESCRIPTION>
      Value for the age of the star in Gyr.
    </DESCRIPTION>
  </PARAM>
  <PARAM name="metallicity" ucd="..." unit="" value="0">
    <DESCRIPTION>
      Value for the metallicity of the star defined as Fe/H
    </DESCRIPTION>
  </PARAM>
  <TABLE>
    <FIELD name="t" ucd="time.age" unit="Gyr" datatype="float">
      <DESCRIPTION>Age of the star in Gyr</DESCRIPTION>
    </FIELD>
    <FIELD name="M" ucd="phys.mass" unit="" datatype="float">
      <DESCRIPTION>M/Ms = mass in Msun</DESCRIPTION>
    </FIELD>
    <FIELD name="teff" ucd="phys.temperature.effective" unit="K"
datatype="int">
      <DESCRIPTION>Effective temperature for the model. Temperatures
are given in K</DESCRIPTION>
    </FIELD>
    <FIELD name="Logg" ucd="phys.gravity" unit="" datatype="float">
      <DESCRIPTION>log g</DESCRIPTION>
    </FIELD>
    <FIELD name="Lum" ucd="phys.luminosity" unit=""
datatype="float">
      <DESCRIPTION>L/Lsun</DESCRIPTION>
```

```

</FIELD>
<DATA>
<TABLEDATA>
  <TR>
    <TD>0.001</TD>
    <TD>0.0005</TD>
    <TD>628</TD>
    <TD>2.645</TD>
    <TD>4.2658e-06</TD>
  </TR>
  <TR>
    <TD>0.001</TD>
    <TD>0.0010</TD>
    <TD>942</TD>
    <TD>2.996</TD>
    <TD>1.92752e-05</TD>
  </TR>
  ....
  ....
  <TD>0.001</TD>
  <TD>0.0070</TD>
  <TD>2098</TD>
  <TD>3.515</TD>
  <TD>0.00100462</TD>
</TABLEDATA>
</DATA>
</TABLE>
</RESOURCE>
</VOTABLE>

```

In this example, the data file contains five columns: the age of the star, its mass, the effective temperature, the logarithm of the gravity and the luminosity. But this is not part of the protocol. Each model can provide the columns preferred by the author.

3 Model credits and references

In the VO, it is important that the user can recover easily the origin of the data and the processes and tools used to transform and analyze the original data.

This is particularly important in the case of theoretical models, both for:

- scientific reasons: the user must be aware of the technical details of the model that he is using for his work (being able, for instance, to access a scientific paper with those details).
- adequately reference or acknowledge the scientific work and technical tools used, which is very relevant in order to encourage model developers to make available their work in the VO.

Thus, the following information must be included in all the VOTables produced by the server:

Bibliographic references:

- One or more PARAM element with name="bibcode" ucd="meta.bib" containing, as value="...", the bibcode for a publication describing the model. It can contain also a DESCRIPTION element with the text corresponding to that bibcode (authors, title...).
 - These references are intended to be the ones that should be cited if someone uses this VO service for scientific publication.
 - In the case that the model developer decides to include several of these PARAMs, it is recommended that they are listed in decreasing order of importance.

Acknowledgments

- One or more PARAM element with name="acknowledgments" containing, as a DESCRIPTION element, the text that should be included in the acknowledgments section of a publication that has made use of the model.

If someone uses the VO service is encouraged to:

- Read the scientific material contained in the references to be aware of the technical details, being sure that the model can be used for the scientific case considered by the user, etc.
- Use these references in any forthcoming publication that makes use of the model.

Any application that automatically uses this service (for instance, to fit observational data on the fly) is also encouraged to give the user an easy way to recover this information.

4 References

[1] D. Tody et al, Simple Spectral Access Protocol

<http://www.ivoa.net/Documents/latest/SSA.html>

[2] G. Lemson et al, Proposal for a Simulation Database Standard

<http://www.ivoa.net/Documents/latest/SimDBTrack.html>

[3] C. Gheller et al, Simulation Data Access Protocol (SimDAP),

<http://code.google.com/p/volute/source/browse/trunk/projects/theory/snap/SimDAP.html>