International

Virtual

Observatory

Alliance

UTYPEs: Portable Data Model References

Version 0.5-20130422

Working Draft 2013 April 22

This version:
0.x-20130412
Latest version:
-
Previous version(s):
Older mapping documents?

Editors:
Gerard Lemson
Omar Laurino
Mireille Louys

Authors:
UTYPE-s tiger team

Abstract

Data providers and curators provide a great deal of metadata with their data files: this metadata is invaluable for users and for Virtual Observatory software developers. In order to be interoperable, the metadata must refer to common Data Models. We propose a scheme for annotating data files in a standard, consistent, interoperable fashion, so that each piece of metadata can unambiguously refer to the correct Data Model element it expresses. We also describe in detail how to represent Data Model instances in the VOTable format. The mapping is operated through opaque, portable strings: UTYPEs.

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

Data providers put a lot of effort in organizing and maintaining metadata that precisely describes their data files. This information is invaluable for users and for software developers that provide users with user-friendly VO-enabled applications. For example, such metadata can characterize the different axes of the reference system in which the data is expressed, photometric information, or the history of a measurement, like the publication where the measurement was drawn from, the calibration type, and so forth. In order to be interoperable, this metadata must refer to some Data Model: the IVOA defines and maintains such standardized Data Models that describe astronomical data in an abstract, interoperable way.

However, most often each astronomical facility, instrument, or mission is in need to express their measurements and the metadata attributes unique to the facility, instrument, or mission.

In other terms, a Data Provider can *extend* a Data Model, adding to the common information about astronomical measurements the metadata that is specific for their instruments or domain.

Thus, in order to enable such interoperable, extensible, portable annotation of data files, one needs:

i) Pointers linking a specific piece of information (data or metadata) to the Data Model element it represents (UTYPEs).
ii) A language to describe Data Models and their valid pointers, so to support extensibility and efficient software development (VO-DML)

Without a consistent language for describing Data Models, pointers alone are ambiguous and redundant: as such, they have small value. On the other hand, the language must be expressive and formal enough to enable the development of reusable, extensible software components and libraries that can make the technological uptake of the VO standards seamless and scalable.

Also, one needs to map the abstract Data Model to a particular format meta-model. For instance, the VOTable format defines RESOURCEs, TABLEs, PARAMs, FIELDs, and so forth, and provides explicit attributes such as units and utypes: in order to represent instances of a Data Model, one needs to define an unambiguous mapping between these meta-model elements and the Data Model language, so to make it possible for software to be able to parse a file according to its Data Model.

While a standard for portable, interoperable Data Model representation would have been required in the past, we are specifying it only at a later stage. This means that several different interpretations of UTYPEs have been proposed and used: some explicitly require the parsing of the UTYPEs strings, others make UTYPEs very redundant, by defining many UTYPEs (as many as hundreds) for expressing the same concept (e.g. the accuracy of a measurement) in different contexts: for instance, the accuracy of the measurement on the spectral axis, the accuracy of the measurement on the time axis, the overall accuracy on the spectral axis for a dataset, an accuracy provided by a service response or the one provided by a standalone file, all have different UTYPEs, while an error bar is an error bar regardless of the quantity to which it is referred, or whence the response originally came from.

Any standard trying to reconcile these very different usages must take into account the current usages and make the transition from the current usages to the new standard as seamless as possible. For this reason, this document also shows how the current UTYPEs usages can be seamlessly integrated with the new scheme, so to minimize the transition effort.

Actually, since this standardization is required by new, more complex Data Models (such as data cubes and their projections) than the ones already defined, it will be possible to adopt this standard only to the new Data Models and Data Access Services, thus avoiding any transition efforts. We also show how Data Access Services can employ a convenient representation of the Data Models fields with query-like strings that resembles the current usage of UTYPEs in DAL services, or for future data access features.

This is a very technical and formal document that can enable the development of flexible, reusable, user-friendly libraries and applications that abstract and generalize the input/output access to VO compliant files, thus facilitating the uptake of the VO in the astronomical community, and the simplification of the process for publishing data to the VO.

This document defines the scope and usage of standardized UTYPEs as means to annotate and describe Data Model instances.

It also describes how to represent Data Model instances using the VOTable schema. This representation uses UTYPE attributes and the structure of the VOTable meta-model elements to indicate how instances of data models are stored in VOTable documents. We show many examples and give a complete listing of allowed mapping patterns. We believe this approach to be a complete and in a certain sense most explicit mapping language.

It is however *not* meant to be *the* normative specification for using utypes in VOTable. But it can be used as a basis for such a specification when also other issues have to be taken into consideration.

In sections 1-6 we give an introduction to the UTYPEs approach, an overview of VO-DML as a meta-model, and several examples that illustrate the mapping. Section 7 aims to be a more rigorous listing of all valid annotations. In order to make this formal part of the document easy to browse, the titles in this section formally describe the mapping in a concise yet complete way. Section 8 discusses possible annotations that "we" propose should *not* be supported, though they may seem obvious. The appendices have some supplementary material, partially there as placeholder.

2 Use Cases

The use cases enabled by this mapping definition are limitless. This bold statement can be easily validated by considering that what we describe is analogous to the natural mapping between Data Models and XSD schemata, where instances are expressed in XML documents. XML is widely used in so many ways that it is impossible to list them all. As a matter of fact, XML can even express lists of its own use cases.

However, to give a sense of what it is possible to accomplish with this specification, we provide some explicit use cases relative to the VO domain.

Reuse of instances: this use case is very low level, but explains the very simple mechanism that enables all the other advanced, higher-level use cases, such as those described below.
An instance is defined as a UTYPE-annotated GROUP in a VOTable: for example, a GROUP with the utype “sdm:PhotometryPoint” may represent a photometry measurement in the N-Dimensional Energy-Flux-Time-Polarization-… space. If such an instance is compliant to the standard expressed in this document, then the whole GROUP can be inserted in any context and still be a compliant, meaningful instance. It might be one of the measurements in a photometry catalog, described by one or more columns in a table, or it can describe the rows in a SED built by a SED Tool. All Data Providers will annotate all Photometry Points in exactly the same way, and all applications will be able to treat it as a single entity regardless of the context. In the end, a photometry point is always the same, whether it is part of a catalog or of a SED, whether it is plotted or fitted to a model: the current standard enormously simplifies the representation, discoverability, and use of such ubiquitous objects.

VO Publisher.

VO Importer.

Plotting.

Fitting.

3 UTYPEs as mapping language

When encountering a data container, i.e. a file or database containing data, one may wish to interpret its contents according to some external, predefined data model. That is, one may want to try to identify and extract instances of the data model from amongst the information. For example in the “global as view” approach to information integration, one identifies elements (e.g. tables) defined in a global schema with views defined on the distributed databases [TBD refs].
If one is told that the data container is structured according to some standard serialization format of the data model, one is done. I.e. if the local database is an

exact *implementation* of the global schema, one needs no special annotation mechanism to identify these instances. An example of this is an XML document conforming to an XML schema that is an exact physical *representation* of the data model.

But in an information integration project like the IVOA, which aims to homogenize access to many distributed heterogeneous data sets, databases and documents are in general *not* structured according to a standard representation of some predefined, global data model. The best one may hope for is to obtain an *interpretation* of the data set, defining it as a *custom serialization* of the result of a *transformation* of the global data model[1]. For example, even if databases themselves are exact replications of a global data model, results of general queries will be such custom serializations.

To interpret such a custom serialization one generally needs extra information that can provide a *mapping* of the serialization to the original model. If the serialization *format* is known, this mapping may be given in phrases containing elements both from the serialization format and the data model. For example if our serialization contains data stored in 'rows' in one or more 'tables' that each have a unique 'name' and contain 'columns' also with a 'name', you might be able to say things like:

– The rows in this table named SOURCE contain instances of object type 'Source' as defined in data model 'SourceDM'.
– The type's 'name' attribute (having datatype 'string', a primitive type) also acts as the identifier of the Source instances and is stored in the single column with name ID.
– The type's 'classification' attribute is stored in the table column CLASSIFICATION (from the data model we know its datatype is an enumeration with values ..., these are represented here by ...).
– The type's 'position' attribute (being of structured data type 'Coordinate' defined in model 'STC2.0') is stored over the two columns RA and DEC, where RA stores the Coordinate's attribute 'longitude', DEC stores the 'latitude' attribute. Both must be interpreted using an instance of the CoordinateSystem type, This instance is stored in 1) another document elsewhere and a reference to it is this url: http://ivoa.net/documents/STC-Instances/ICRS, or 2) in this document and its identifier is ....
– Instances from the collection of luminosities of the Source instances are stored in the same row as the source itself. Columns MAG_U and ERR_U give the 'magnitude' and 'error' attributes of type LuminosityMeasurement in the "u band", an instance of the Filter type. (stored elsewhere in this document (a reference to this Filter instance is ...). Columns MAG_G and ERR_G ... etc.

In this example the underlined words refer to concepts defined in VO-DML, an example of a meta-model that can be used as a formal language for expressing data models. The use of such a modeling language lies in the fact that it provides formal, simple and implementation neutral definitions of the possible structure, the 'type' and 'role' of the elements from the actual data models that one may

[1] Or alternatively as a transformation of a (standard) serialization of the data model.

encounter in the serialization (SourceDM and STC2.0). This can be used to constrain or validate the serialization, but more importantly it allows us to formulate mapping rules between the serialization format (itself a kind of meta-model) and the meta-model, independent of the particular data models used; for example rules like:

- An object type MUST be stored in a 'table'.
- A 'primitive type' MUST be stored in a 'column'.
- A reference MUST identify an object type instance represented elsewhere, either in another 'table', possibly in the same table, possible in another document.
- An attribute SHOULD be stored in the same table as its containing object type.
- etc

Clearly free-form English sentences as the ones in the example are not what we're after. If we want to be able to identify how a data model is represented in some custom serialization we need a formal, computer readable mapping language.
One part of the mapping language should be anchored in a formally defined serialization language. After all, for some tool to interpret a serialization, it MUST understand its format. A completely freeform serialization is not under consideration here. This document assumes VOTable. Alternatives that have been mentioned in our discussion are FITS and TAP_SCHEMA. VOTable is richer and has components the other two miss. Hence the language we propose here is not necessarily applicable to these two. We discuss possible ways how to deal with that impedance mismatch[2].
The mapping language must support the interpretation of elements from the serialization language in terms of elements from the data model. If we want to define a generic mapping mechanism, one by which we can describe how a general data model is serialized inside a VOTable, it is necessary to use a general data model *language* as the target for the mapping, such as the one described above. This language can give formal and more explicit meaning to data modeling concepts, possibly independent of specific languages representation languages such as XML schema, Java or the relational model. In this document we will use VO-DML as the target language; section 4 discusses this meta-model, but a full spec of the language should be made.
The final ingredient in the mapping language is a mechanism that ties the components from the two different meta-models together into "sentences". This generally requires some kind of explicit annotation, some meta-data elements that provide an identification of source to target structure. In general the ease with which a link can be made between two meta-models depends largely on how "similar" they are. The more similar, the easier an annotation can be.
In this document we will argue that in VOTable the 'utype' attribute can provide this link in a rather simple manner:

- The value of a utype attribute must correspond to the utype identifier of an element explicitly defined in a VO-DML document, or the concatenation of two such identifiers.

[2] http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

- The VOTable element owning the utype attribute is said to *represent* the identified VO-DML data model element. It identifies one or more instances of the data model element, the identification depends on the kind of element and on the context in which it appears.
- There is a set of rules that constrain *which* VOTable elements can be identified with *which* type of VO-DML element and how the context plays a role here.

We show that this solution is sufficient and that it is in some sense the simplest and most explicit approach for annotating a VOTable. It may *not* be the most natural or suitable approach for other meta-models such as FITS or TAP_SCHEMA. For example the current approach relies heavily using on GROUPs to identify most of the structural mapping. FITS and TAP_SCHEMA do currently not possess such a construct. Massaging the approach so it works also for those cases, as well as producing some level of backwards compatibility *may* be possible though not necessarily the only solution. In any case it might be derived from the more explicit solution presented here. We will discuss this at the end of this document.

In the next section we review the main concepts in VO-DML, whilst we assume that all readers are familiar with VOTable. The following section shows examples of mappings as they may occur in realistic VOTables. We will discuss the different VO-DML elements and show how they might be serialized and how annotation with simple utype and other meta-data elements can help one interpret the VOTable. In the later normative sections we turn this around and explicitly list all legal annotations, their constraints and interpretation.

4 Summary of VO-DML concepts

VO-DML is ... TBC.

Containers:

- Model
- Package

Types:

- PrimitiveType
- Enumeration
- DataType
- ObjectType

Property:

- Attribute
- Reference
- Collection
- Extends

4.1 VO-DML/XML

VO-DML has an XML schema representation that defines the structure of XML documents containing valid data models The schema is complemented by a Schematron file that defines rules on the documents that are hard if not impossible to define in XML schema; examples are constraints on valid data types for particular roles and are rules constraining links to external models. The schema files are located in vo-dml.xsd and vo-dml.sch.xml and example models can be found in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models.

4.2 Default utype syntax

The vo-dml.xsd schema defines for each referenceable element an identifier named 'utype'. The value of each instance of this must be unique in the data model document and this is the *only* requirement on its value. To guarantee this uniqueness one may use some syntax rules to derive the utype value from the location of its parent in the document. The uniqueness assumes certain constraints on valid data models such as uniqueness of names of types in a package, or of roles in a type. The syntax we use is the one first proposed in the Simulation Data Model (section 4.1), later taken over by a first draft of the UTYPE document, and reproduced here:

```
utype := [model-utype | package-utype | type-utype | attribute-utype | collection-utype
       | reference-utype | container-utype | identifier-utype | extends-utype
model-utype := <model-name>
package-utype := model-utype ":" package-hierarchy
package-hierarchy := <package-name> ["/" <package-name>]*
type-utype := package-utype "/" <type-name>
attribute-utype := type-utype "." attribute
attribute := [primitive-attr | struct-attr]
primitive-attr := <attribute-name>
struct-attr := <attribute-name> "." attribute % not explicitly represented in VO-DML/XML
doc %
collection-utype := type-utype "." <collection-name>
reference-utype := type-utype "." <reference-name>
container-utype := type-utype "." "CONTAINER"
identifier-utype := type-utype "." "ID"
extends-utype := type-utype "." "EXTENDS"
```

[TBD this started as a copy from the SimDM doc; changes in red; need to check whether it still conforms to our desired practice.]

4.3 'vo-dml:' utypes

It is sometimes useful to be able to use utypes to refer to elements in the *meta-model*. And more subtly, we need a formal way of indicating that certain VOTable elements represent *instances* of the meta model that are not instances of type definitions in a particular model. For example we want to be able to indicate that certain models are represented in a VOTable. To do so with utypes we have created a small model in VO-DML representation representing some super types and meta-types. This model is located in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/vo-dml/VO-DML.vo-dml.xml[3] and contains the following main elements.

- `vo-dml:Model`

  Can be assigned to a GROUP element to indicate a certain model is used
    - `vo-dml:Model.url`

      Can be assigned to an PARAM inside a Model GRUOP element to hold the URL of the VO-DML document defining the model.
    - `vo-dml:Model.prefix`

      Can be assigned to an PARAM inside a Model GROUP element to hold the prefix used to identify utype-s form this model.
- `vo-dml:ObjectType`

  Indicates the ObjectType definition, can also be seen more accurately as the (implicit) common base class of all ob ject type instances..
    - `vo-dml:ObjectType.ID`

      Indicates the ID attribute (implicitly) defined on ObjectType. Can be assigned to a FIELDref, PARAM or GROUP to indicate it represents an ObjectType's identifier.
- `vo-dml:Identifier`

  A generic DataType indicating an identifier. When used as type part of a concatenated utype in an annotation of a reference, indicates that the annotated element(s) together act as a foreign key to a remote object that must have the same identifier contents.

One could imagine others such as vo-dml:Collection to indicate that a certain element is a collection rather than a collection element. One also could also consider using implicitly defined vo-dml:ObjectType.CONTAINER and vo-dml:Type.EXTENDS iso of the generated CONTAINER and EXTENDS utypes. It might facilitate the writing of a model a bit, requiring fewer explicit utype identifiers. And after all containers and extensions are 0..1, so there is no possibility of confusion.

5 Format of @utype: Mapping types and roles

[NOTE in telecom 2013-04-09 it was decided to support type casting on GROUP *not* using concatenation of role+type utype, but using a PARAM with special utype indicating it reflects the actual data type of the instance, and value the utype f the corresponding type. In sample VOTable in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/doc/examples/Sample_1.votable this is implemented using utype="vo-dml:Instance.type".]

[3] This model still depends for its PrimitiveType on the IVOA_Profile model in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/profile/IVOA_Profile.vo-dml.xml. We might decide to merge the two into a VO-DML Profile or so. That model would have to be part of a spec and be separately maintained.

[NOTE we need to decide on format of utype in VO-DML *and* VOTable, in particular how to use the prefix. A utype used in a VOTable MUST have a prefix that indicates the model using the vo-dml:Model GROUPs declared in the VOTABLE itself. IN VO-DML documents though, should that same prefix be used, or can we leave it out?]

In general a mapping from VOTable to VO-DML uses a @utype attribute of a VOTable element to identify the VO-DML element it represents. The approach here assumes that the @utype takes the value of the utype identifier of the VO-DML element it represents, or at least is built from such identifiers. This value may be built according to the default syntax described in section 4.2, but that is not required. These utypes generally identify type definitions or property definitions. A property is a role played by a type in the definition of another type and is an important piece of metadata to add to one's VOTable.
However, a @utype attribute identifying only a property may not provide sufficient information about the contents of the annotated element. This is due to the fact that our data modeling language supports inheritance and polymorphism. Polymorphism is the common object-oriented design concept that says that the declared type of a property may not be the same as the type of an instance of that property that is actually serialized. In particular, the value of a property may be an instance of a *subtype* of the declared type. So in general it is not enough to know the type of the attribute (for example) to uniquely know which type of instance to expect. And it may also not be possible to infer the instance type uniquely from the contents of the element representing the attribute.
Hence a single @utype attribute with value indicating only the role may not be sufficient to infer all DM information about a VOTable element. Typed languages such as Java support a casting operation, which provides more information to the interpreter about the type it may expect a certain instance to be. We might have to consider adding a similar kind of explicit *casting* to utype attributes, allowing clients to identify both role and type.
A separate motivation for such explicit casting support is simply to improve usability of the annotation. A typical usage scenario is [TBD may be?] a VOTable client tool that is sensitive to certain models only, say STC. Such a tool can be written to understand annotation with STC types. Finding an element mapped to a type definition from STC it might infer or example that it represents a coordinate on the sky and “do something with that information”. Such a tool would not necessarily understand other models where such an STC type is *used* as a role. If the annotation only refers to the attribute’s utype, a tool will have to infer that it represents the role played by a known type after parsing the data model document. If instead the actual type can be inferred from the annotation directly usability might be increased.
Two possibilities have been proposed to support this kind of type casting, other might be considered:

1. Concatenate utypes. Identify both role and type using the corresponding utype-s, separated by a separator, for example a ‘+’.

2. Use the @utype to identify the type, and another attribute, for example the @name to identify the role. This may work for GROUP and PARAM on a GROUP, but not for FIELDref and PARAMref. In those latter cases one might use the @utype of the …ref element to indicate the role, and the @utype of the referenced FIELD and PARAM respectively to indicate the type.
   Or a new attribute could be used. E.g. urole for the role, utype for the type.

In the rest of this note we assume option 1, including the use of the '+'. We could consider using a different separator, say a ';'. Reason could be that ';' has been used in other interpretations already, basically to give an *alternative* annotation. We prefer the '+' as it makes explicit that the two utypes concatenated together are *complementary*; both are needed for a complete interpretation of an annotation aimed at a VO-DML document. It should be straightforward to change the description if another option is chosen.
TBD Note that the choice of how to do casting *does* have some consequences, namely that the utype identifiers used in VO-DML documents must not use this separator, or at least follow a syntax that allows the concatenation of utypes to be discovered.
Another argument against concatenation is that simple string-2-string comparison of a utype may not work. This should not really be a big deal for languages such as Java where a simple split("[+]") operation will produce one or two results that can be compared. Similar things will have been required for UCD-s already as well, and there no problem was indicated. So why would it be a problem for UTYPEs? However the solution proposed here is less normalized and less XML-ish. It is harder to use in XPath/XSLT type processing for example.
Note that we MUST conclude that this issue is a consequence of proposing a solution before a proper analysis of the problem was made. For example if instead of using an XML attribute, an element of type UTYPE was used (where that type was TBD), we would not have these problems.

6 Examples: Mapping VO-DML ⇒VOTable

In this section we list some mappings from VO-DML to VOTable. We use examples extracted from a sample model with sample instances described in the next section. These should be seen as an introduction to the complete and formal specification in section 7 on how one SHOULD use utypes in VOTable to indicate mapping to VO-DML.

6.1 Sample model and instances

For examples we use a highly simplified version of a possible Source data model, illustrated by its UML representation in Figure 1. It has a VO-DML representation which is reproduced in Appendix A.

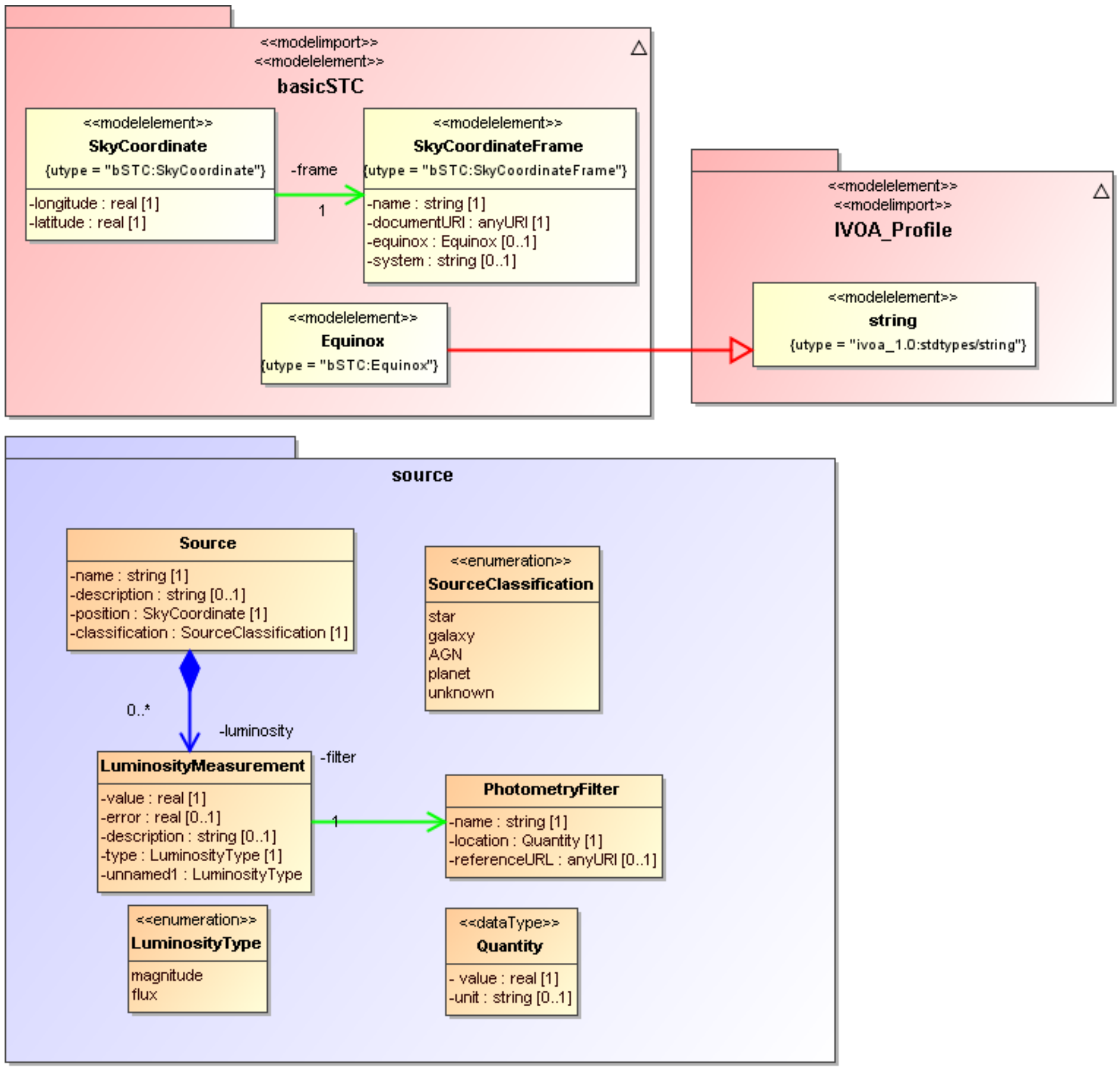


Figure 1 Example data model used in example. It represents a simplified Source data model, containing luminosities following a simplified PhotDM model. It imports a simplistic version of an STC model with some types for defining coordinates on the sky.

The model defines some types allowing one to define a Source with position on the sky and a collection of luminosities. The position is modeled as a DataType, ‘SkyCoordinate’ in a “basicSTC” model that is imported by the Sample model. SkyCoordinate has a reference to a coordinate frame that is required to interpret its longitude and latitude attributes. The luminosities are really *measurements* of luminosities in a given filter that is indicated by a reference to a PhotometryFilter; hence they have a value *and* an error. A Quantity DataType is introduced that provides a real value and a unit. The latter is actually the cause of a possible problem discussed in section 10.
The models are by no means meant to be comprehensive and include some admittedly artificial elements such as an Equinox PrimitiveType, which is

supposed to be a simple string and might carry enough semantic value of its own to use it as an annotation on PARAM elements for example.
Note that this sample model defines a Package [TBD ref to VO-DML document needed] that contains all the types. This package shows up in the values of the utype-s we use to identify the different elements. The values we use for these utype identifiers are generated from the VO-DML using the default utype described in section. I.e. they are path-like expressions that are guaranteed unique and give some impression of the location in the data model.
We also use some sample instances of the models. These are here illustrated by UML instance diagrams. The diagram in Figure 3 represents the first two lines returned from a query to the SDSS DR7 database.

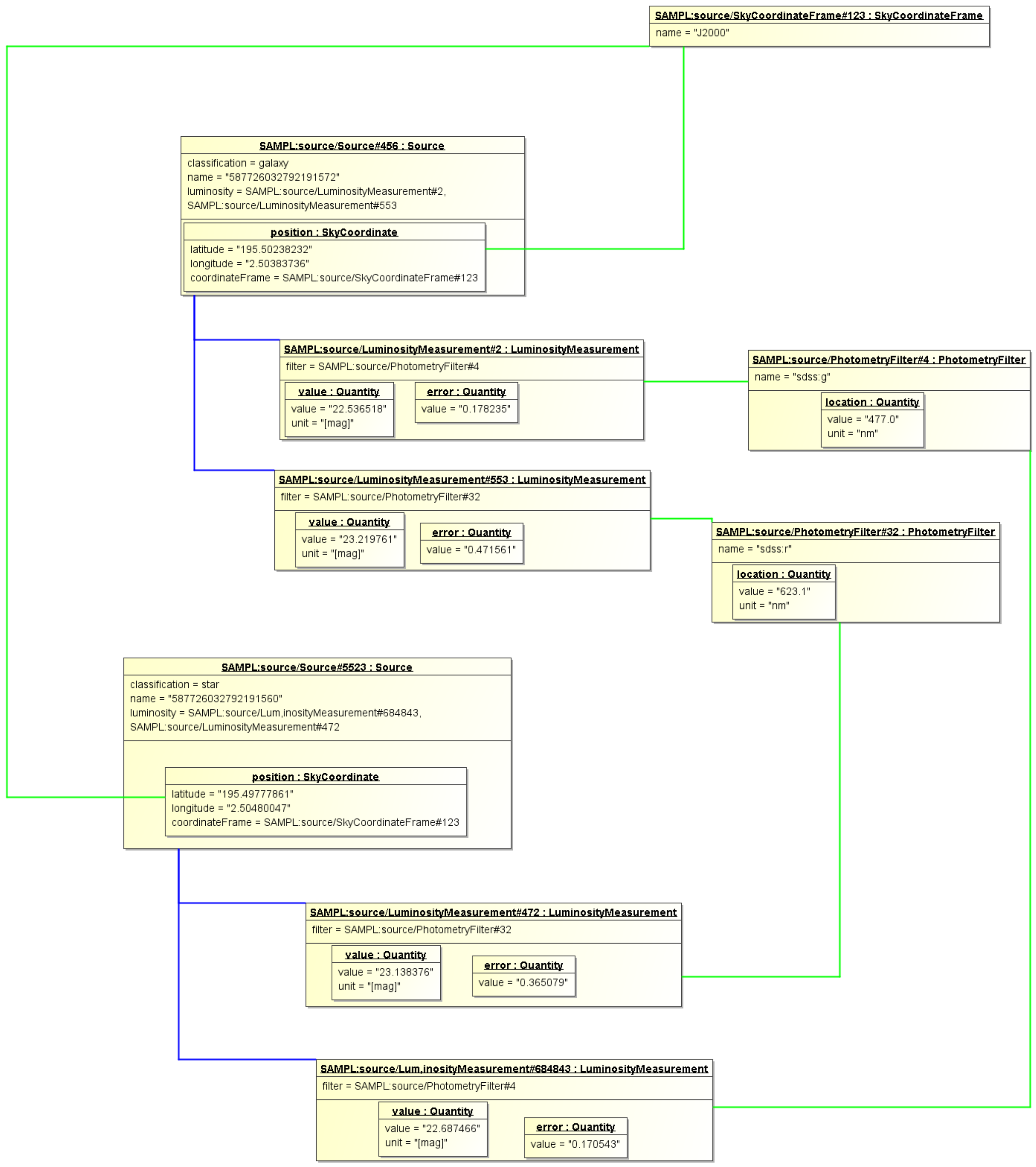

SAMPL:source/SkyCoordinateFrame#123 : SkyCoordinateFrame
name = "J2000"
SAMPL:source/Source#456 : Source
classification = galaxy
name = "587726032792191572"
luminosity = SAMPL:source/LuminosityMeasurement#2, SAMPL:source/LuminosityMeasurement#553
position : SkyCoordinate
latitude = "195.50238232"
longitude = "2.50383736"
coordinateFrame = SAMPL:source/SkyCoordinateFrame#123
SAMPL:source/LuminosityMeasurement#2 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#4
value : Quantity
value = "22.536518"
unit = "[mag]"
error : Quantity
value = "0.178235"
SAMPL:source/PhotometryFilter#4 : PhotometryFilter
name = "sdss:g"
location : Quantity
value = "477.0"
unit = "nm"
SAMPL:source/LuminosityMeasurement#553 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#32
value : Quantity
value = "23.219761"
unit = "[mag]"
error : Quantity
value = "0.471561"
SAMPL:source/PhotometryFilter#32 : PhotometryFilter
name = "sdss:r"
location : Quantity
value = "623.1"
unit = "nm"
SAMPL:source/Source#5523 : Source
classification = star
name = "587726032792191560"
luminosity = SAMPL:source/Lum,inosityMeasurement#684843, SAMPL:source/LuminosityMeasurement#472
position : SkyCoordinate
latitude = "195.49777861"
longitude = "2.50480047"
coordinateFrame = SAMPL:source/SkyCoordinateFrame#123
SAMPL:source/LuminosityMeasurement#472 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#32
value : Quantity
value = "23.138376"
unit = "[mag]"
error : Quantity
value = "0.365079"
SAMPL:source/Lum,inosityMeasurement#684843 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#4
value : Quantity
value = "22.687466"
unit = "[mag]"
error : Quantity
value = "0.170543"

Figure 3 Instance diagram representing SDSS objetcs as sources in the sample data model. [TBD update to latest form of model]. The first few results are represented from the default radial SDSS query at http://skyserver.sdss.org/dr7/en/tools/search/radial.asp corresponding to the SQL query:

```
SELECT  top 10   p.objID, p.run, p.rerun, p.camcol, p.field, p.obj,
   p.type, p.ra, p.dec, p.u,p.g,p.r,p.i,p.z,
   p.Err_u, p.Err_g, p.Err_r,p.Err_i,p.Err_z
   FROM fGetNearbyObjEq(195,2.5,3) n, PhotoPrimary p
   WHERE n.objID=p.objID
```

The instance diagram in

Figure **4** represents one of the sources returned by a cone search query vs. the 2MASS catalogue, represented as instances of the Sample data model.

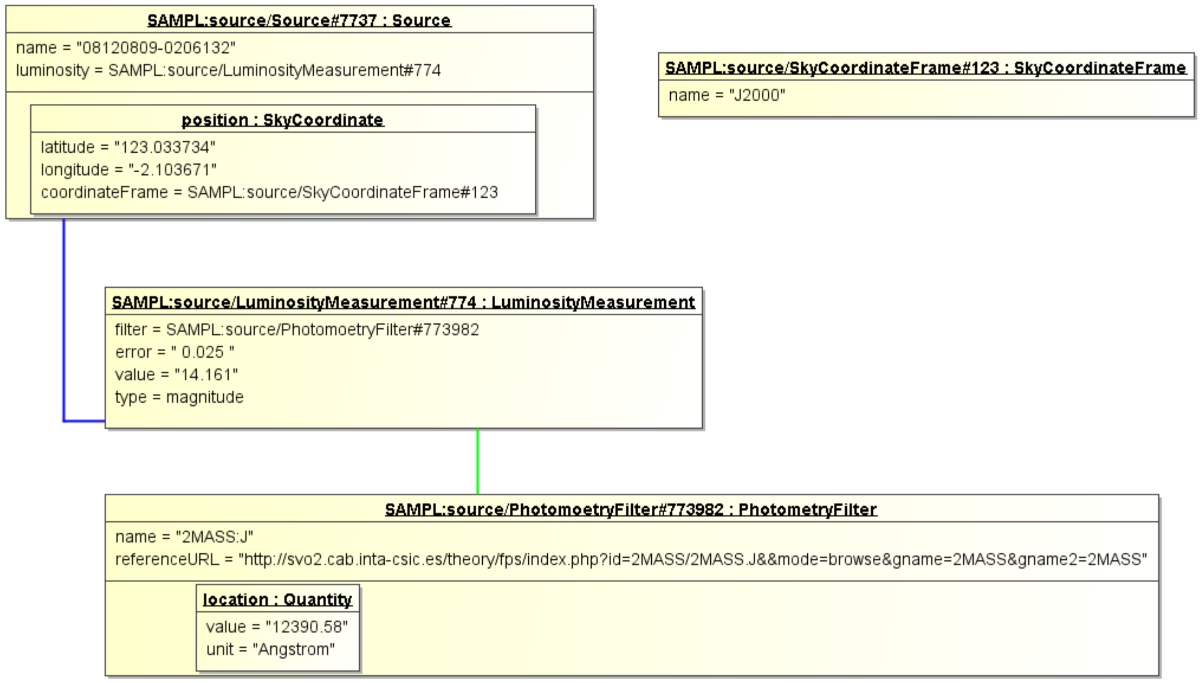

SAMPL:source/Source#7737 : Source
name = "08120809-0206132"
luminosity = SAMPL:source/LuminosityMeasurement#774
position : SkyCoordinate
latitude = "123.033734"
longitude = "-2.103671"
coordinateFrame = SAMPL:source/SkyCoordinateFrame#123
SAMPL:source/SkyCoordinateFrame#123 : SkyCoordinateFrame
name = "J2000"
SAMPL:source/LuminosityMeasurement#774 : LuminosityMeasurement
filter = SAMPL:source/PhotomoetryFilter#773982
error = " 0.025 "
value = "14.161"
type = magnitude
SAMPL:source/PhotomoetryFilter#773982 : PhotometryFilter
name = "2MASS:J"
referenceURL = "http://svo2.cab.inta-csic.es/theory/fps/index.php?id=2MASS/2MASS.J&&mode=browse&gname=2MASS&gname2=2MASS"
location : Quantity
value = "12390.58"
unit = "Angstrom"

Figure 4 A 2MASS source is partially represented here according to the SAMPLE model in Figure 1. Result is obtained from http://irsa.ipac.caltech.edu/ using coordinate "123 -2.1" with search radius 3'. TBD update to latest form of model

6.2 Data carriers in VOTable

VO-DML describes four different kinds of types: PrimitiveType (PT), Enumeration (E), DataType (DT) and ObjectType (OT). PT, E and DT are value types; OT is an object - or reference type [TBD add few sentences]. PT and E are atomic, their values consist of a single value; DT and OT are structured, they are built from multiple values, organized as attributes, and possibly of reference relations to OTs. An OT can also have collections of other OTs, and can have an identifier, an attribute of undetermined type that is implicitly defined for ObjectType-s.
To store instances (*values* and *objects*) of these types in a VOTable various options are available. Atomic values (i.e. instances of PT and E) are stored in cells in a row in a table (i.e. a TD) or in the value attribute of a PARAM (@value). To store an instance of a structured type one must store its components. To identify the structured instance in a serialization one must be able to identify the individual components and how they are to be combined. This last identification is done by the GROUP element. It is the main representation of structured types, both ObjectType and DataType. It is also used to represent relations to object types. These may be stored using foreign key like mechanisms or through some kind of hierarchy.
In fact in the approach described here virtually *all* mapping of VOTable to VO-DML is performed by GROUP elements and their components. They identify which elements are to be combined to create the object or DataType instance,

what the roles are that these components play (attribute, reference) and they do so simply by using the utype of the corresponding type or role as defined in VO-DML for the value of their @utype attribute.
In the next few subsection we provide some examples how this mapping can be performed. It starts with the ObjectType and then discusses its components, Attribute, Reference and Collection. The mapping of the value types is discussed in the mapping of Attributes. The mapping of Reference and Collection relations is the most complex part of the whole mapping story and treated separately.

6.3 Mapping ObjectType

ObjectTypes consist of Attributes, References and Collections. How these are mapped is described in more detail below. But the important part for representing structured instance like an object is that these components must be combined together to construct a complete instance. In VOTable this is done using a GROUP element.
In the representation of ObjectType instances, GROUPs can be used in two different modes that are distinguished by the way the instance's data are ultimately stored. If all values are eventually stored exclusively in @value attributes of PARAM elements in the GROUP (or possibly outside the GROUP but accessed through PARAMrefs), the GROUP represents a complete instance *directly*. If even only one of the attributes is stored in a FIELD and accessed through a FIELDref, the GROUP is said to *indirectly* represents possibly multiple instances, one for each TR.
We will use these terms, "direct GROUP" and "indirect GROUP" as shorthand phrases for these representations all through the document. This freedom of choice *where* to store objects complicates the mapping of *relations* between objects, as many different referencing mechanisms must be taken into account. This is particularly important when discussing how to represent References in section 7.7 below.
We illustrate the two modes of mapping by showing an example how each mode may represent exactly the same object. For this we use the object type in Figure 5 and a corresponding instance in Figure 6.

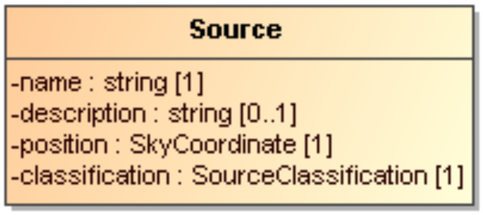

Source
-name : string [1]
-description : string [0..1]
-position : SkyCoordinate [1]
-classification : SourceClassification [1]

Figure 5 ObjectType representing a Source.

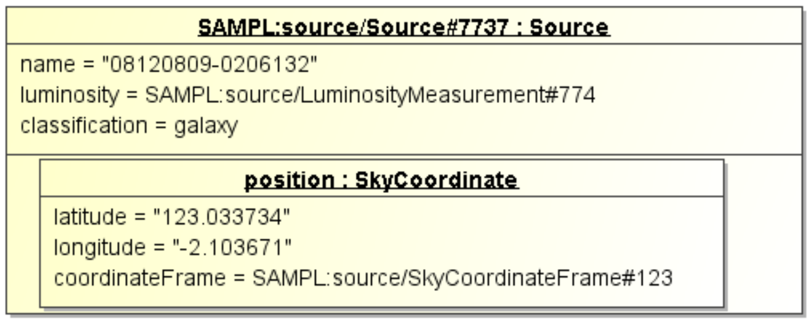

SAMPL:source/Source#7737 : Source
name = "08120809-0206132"
luminosity = SAMPL:source/LuminosityMeasurement#774
classification = galaxy
position : SkyCoordinate
latitude = "123.033734"
longitude = "-2.103671"
coordinateFrame = SAMPL:source/SkyCoordinateFrame#123

Figure 6 Instance of Source ObjectType.

Indirect serialization to a TABLE:
Source instance from Figure 6 is stored in the first TR below. The TABLE is annotated using a GROUP with utype attribute identifying the type. Some atomic attributes are stored in FIELDs annotated by FIELDref-s, some in PARAMs; a structured attribute is represented by the child GROUP with its attributes annotated by FIELDref. A reference is also represented by a GROUP (for discussion see section 6.5):

```
<TABLE>
<GROUP utype="SAMPL:source/Source" >
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classifiication" value="galaxy"/>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype=" bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype=" bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
```

Note the special representation of the identifier of the Source object. This attribute is not defined explicitly in the model; hence no utype exists for it there. We can interpret each ObjectType as ultimately being a subclass of some ObjectType class (just as in Java all classes ultimately extend java.lang.Object, generally implicitly). In VO-DML we can represent this by allowing an ObjectType to have a component of as yet unspecified type, that represents the ID attribute (implicitly) defined on this base class. We use a special utype for this attribute: vo-dml:ObjectType.ID, obtained from the VO-DML model discussed in 4.3.
UPDATE 2013-03-19: in section 4.3 we have introduced a small model that contained base classes to be used in instantiations of types. There we introduce a DataType `vo-dml:Identifier` that acts as the datatype of the ID atrribute. If we take that seriously, in serializations it should be represented by a GROUP. In section 4.3 we note that the VO-DML base model may have to be part of a VOTasble-UTYPE spec, hence we might decide to make special allowances

Note furthermore that the same FIELD is referenced for holding both the ‘name’ attribute of the Source class, and for its identifier, albeit from different GROUPs!
Direct serialization to a GROUP:
Here the instances is directly represented by a GROUP containing only PARAMs for the atomic attributes, and a GROUP with PARAMs for the structured attribute (and again a reference, see section 6.5).

```
<GROUP utype="SAMPL:source/Source" >
  <PARAM utype=”vo-dml:ObjectType.ID" value=”08120809-0206132” .../>
  <PARAM utype=”SAMPL:source/Source.name" value=”08120809-0206132” .../>
  <PARAM utype=”SAMPL:source/Source.classifiication” value=”galaxy” .../>
  <GROUP utype="SAMPL:source/Source.position+ bSTC:SkyCoordinate">
    <PARAM utype=" bSTC:SkyCoordinate.longitude" value=”123.033734” .../>
    <PARAM utype=" bSTC:SkyCoordinate.latitude" value=”-2.103671” .../>
    <GROUP ref="_icrs" utype=" bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
```

Details on the mapping of the components are discussed next.

6.4 Mapping Attribute

An attribute is the role a value type plays in the definition of a structured type. They may represent atomic or may represent structured (Data)types themselves. These are represented differently.

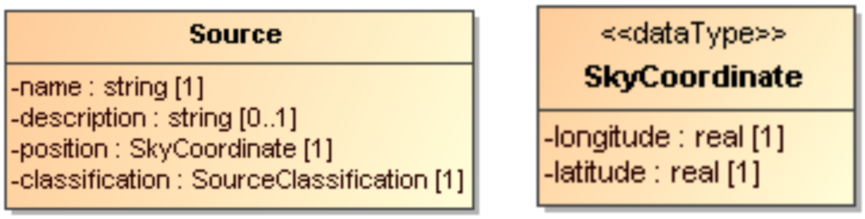

Source
-name : string [1]
-description : string [0..1]
-position : SkyCoordinate [1]
-classification : SourceClassification [1]
<<dataType>>
SkyCoordinate
-longitude : real [1]
-latitude : real [1]

Figure 7 The ObjectType Source and the DataType SkyCoordinate both define attributes. Attributes can represent a PrimitiveType (‘name’ and ‘description’ in Source, ‘longitude’ and ‘latitude’ in SkyCoordinate), an Enumeration (‘classification’ in Source), or a structured DataType (‘position’ in Source).

PrimitiveType attribute as FIELDref, Enumeration as PARAM:
In the indirect representation of the ObjectType below a FIELDref indicates that an attribute is stored in field with ID=”_designation”. It does so using the utype of the attribute: SAMPL:source/Source.name, identifiying the name attribute of the Source type. Note that in our example we use a utype syntax derived from the VO-DML model itself. From this string one can here infer directly that some role with name ‘name’ defined on a type named Source is represented. But that is all one might infer. In principle this could have been a string like SAMPL: _12_1_1dfa04c4_1354024272942_181358_515. In both cases one would need to inspect the formal data model to find out precisely *what* kind of model element this utype represents.
Another attribute, ‘classification’, is represented by a PARAM through its utype value
SAMPL:source/Source.classification and assigns it the value ‘galaxy’. The attribute has as datatype an Enumeration, SourceClassification and indeed the PARAM defines a VALUES element with various OPTIONs (note that that is

almost useless for a PARAM that represents a single value directly anyway). Its set of Literals indeed contains a value ‘galaxy’. In general however, especially for existing “legacy” databases, one cannot expect that enumerated values will exactly correspond to those in a model. Some type of mapping might be required, howevere OPTION does not support this in VOTable (i.e. has no @utype attribute). The fact that this attribute is stored in a PARAM in the GROUP indicates also that all Source instances stored in the TABLE are classified as galaxies.

```
<TABLE>
<GROUP utype="SAMPL:source/Source" >
  <FIELDref ref="_designation" utype=" vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classification" value="galaxy">
  <VALUES><OPTION value="galaxy"/><OPTION value="star"/>...</VALUES></PARAM>
  <GROUP utype="SAMPL:source/Source.position+ bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype=" bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype=" bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype=" bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
```

DataType attribute as GROUP:
As DataType-s are structured, their natural representation in a VOTable is as a GROUP, whether used directly or indirectly. Hence if an attribute is defined in a VO-DML data model as representing a DataType, it is most naturally represented by a GROUP embedded in the GROUP of the structured type owning the attribute. This is the mapping we [TBD that is, I, Gerard] favor and makes the utype annotation very simple. But see elsewhere for a discussion of an alternative approach [TBD].
The example below shows in red a GROUP representing the attribute ‘position’ (identified by utype SAMPL:source/Source.position) that has as data type an STC SkyCoordinate, which itself consists of a ‘longitude’ and ‘latitude’ attribute (we defer discussing the reference to the next section). Note that the attributes of the STC type have a different prefix (bSTC) to indicate they are imported from a different model. And note their structure indicates *only* their relation to their defining type, bSTC:SkyCoordinate (though this, as discussed above, is unimportant for the current approach which, apart from the prefix, assigns no importance to the syntax of the utype identifiers in VO-DML models).

```
<TABLE>
<GROUP utype="SAMPL:source/Source" >
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classifiication" value="galaxy">
  <VALUES><OPTION value="galaxy"/><OPTION value="star"/>...</VALUES></PARAM>
  <GROUP utype="SAMPL:source/Source.position">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
```

```
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
```

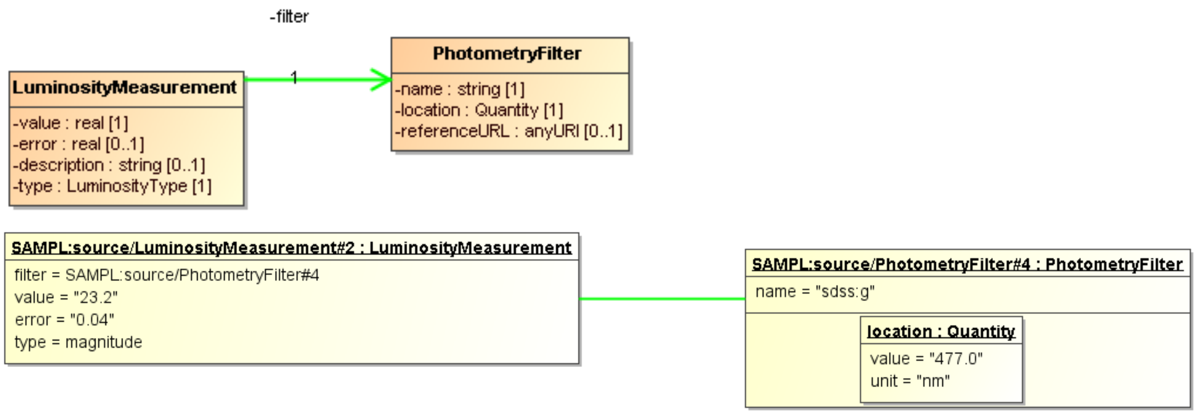
In this example the utype of the child GROUP *only* indicates its role in the definition of its parent GROUP, namely the attribute SAMPL:source/Source.position. It does not indicate the type, which would be bSTC:SkyCoordinate. In principle the role alone should be sufficient, as its definition in a VO-DML data model will provide its data type.
However there are arguments why one might wish to add information about the type as well. As we will argue in section 5, sometimes it is necessary to supply information about the type, as a kind of *casting* operation. There we propose a particular format for this, which consist of a concatenation of the utype identifiers of both role and type, separated by a '+'. According to that proposal the above example could have been written as follows:

```
<GROUP utype="SAMPL:source/Source" >
...
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
...
  </GROUP>
</GROUP>
```

DataType attribute as collection of fields with path-like utype:
An alternative approach that will be discussed in section 9 in more detail aims to avoid the nesting of GROUP elements as much as possible. A structured attribute as 'position' in the example would not be represented with its own GROUP. To ensure one is able to collect the components of the SkyCoordinate type together one might put explicit structure on the utype attribute. One approach is some kind of path expression. In that approach the example could be written as follows:

```
<TABLE>
<GROUP utype="SAMPL:source/Source" >
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classifiication" value="galaxy">
  <VALUES><OPTION value="galaxy"/><OPTION value="star"/>...</VALUES></PARAM>
    <FIELDref ref="_ra" utype="SAMPL:source/Source.position.longitude"/>
    <FIELDref ref="_dec" utype="SAMPL:source/Source.position.latitude"/>
    <GROUP ref="_icrs" utype="SAMPL:source/Source.position.frame"/>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
```

6.5 Mapping Reference


-filter
1
LuminosityMeasurement
-value : real [1]
-error : real [0..1]
-description : string [0..1]
-type : LuminosityType [1]
PhotometryFilter
-name : string [1]
-location : Quantity [1]
-referenceURL : anyURI [0..1]
SAMPL:source/LuminosityMeasurement#2 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#4
value = "23.2"
error = "0.04"
type = magnitude
SAMPL:source/PhotometryFilter#4 : PhotometryFilter
name = "sdss:g"
location : Quantity
value = "477.0"
unit = "nm"

Referencing as ”GROUPref” to direct GROUP:
The example below uses a GROUP+@ref to represent the reference from a position object stored in a TABLE to a SkyCoordinateFrame stored in a direct GROUP. Hence all rows in the table use the same frame and the reference needs no structure.

```
<GROUP utype="bSTC:SkyCoordinateFrame" ID="_icrs">
  <PARAM utype="bSTC:SkyCoordinateFrame.name" value="ICRS" .../>
  <PARAM utype="bSTC:SkyCoordinateFrame.equinox+bSTC:Equinox" value="J2000.0" .../>
</GROUP>
<TABLE>
<GROUP utype="SAMPL:source/Source" id="_source">
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
<FIELD id="_ designation " name="parentId" datatype="char"/>
<FIELD id="_ra" name="ra" datatype="float"/>
<FIELD id="_dec" name="dec" datatype="float"/>
...
<DATA><TABLEDATA>
<TR><TD>08120809-0206132</TD><TD>123.034</TD><TD>-2.1037</TD>...</TR>
...
</TABLEDATA></DATA>
</TABLE>
```

ORM-like referencing from TABLE to TABLE:
A table with PhotometryFilter instances is referenced from a table with LuminosityMeasurement instances. The reference is represented by a GROUP that uses its @ref attribute to identify the GROUP representing the target filters. The reference GROUP contains a FIELDref.
Note, the ID of the PhotometryFilter is constructed from 2 columns, representing (through their @utype) each a field of the Identifier, one identifying a catalogue, one a band. Hence the referencing object must use also two fields. As there is no explicit double linking possible in VOTable (FIELDref cannot use @ref both for indicating the local FIELD and the remotely referenced one), we propose the rule that to reference an Identifier one must replicate its structure. See 7.6 for more details.

```
<TABLE>
<GROUP utype="SAMPL:source/PhotometryFilter" id="_filters">
  <GROUP utype="vo-dml:ObjectType.ID+vo-dml:Identifier"
    <FIELDref ref="_cat" utype="vo-dml:Identifier.field"/>
    <FIELDref ref="_bans" utype="vo-dml:Identifier.field" />
  </GROUP>
  <FIELDref ref="_name" utype="SAMPL:source/PhotometryFilter.name"/>
</GROUP>
<FIELD name="catalogue" id="_cat" .../>
<FIELD name="band" id="_band" .../>
<FIELD name="name" id="_name" .../>
..
<DATA><TABLEDATA>
<TR><TD>SDSS</TD><TD>g</TD><TD>SDSS g-band</TD></TR>
<TR><TD>SDSS</TD><TD>u</TD><TD>SDSS u-band</TD></TR>
<TR><TD>2MASS</TD><TD>J</TD><TD>2MASS J-band</TD></TR>
</TABLDATA></DATA>
</TABLE>
<TABLE>
<GROUP utype="SAMPL:source/LuminosityMeasurement" id="SDSS_mags">
  <GROUP utype="SAMPL:source/LuminosityMeasurement.CONTAINER+vo-dml:Identifier">
    <FIELDref ref="_container" />
  </GROUP>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.value"/>
  <PARAM name="type" utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude"/>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.error"/>
  <GROUP ref="_filters"
         utype="SAMPL:source/LuminosityMeasurement.filter+vo-dml:Identifier">
    <FIELDref ref="_fcat" utype="vo-dml:Identifier.field" />
    <FIELDref ref="_fband" utype="vo-dml:Identifier.field" />
  </GROUP>
</GROUP>
<FIELD id="_container" name="parentId" datatype="char"/>
<FIELD id="_mag" name="mag" datatype="float"/>
<FIELD id="_eMag" name="eMag" datatype="float"/>
<FIELD id="_fcat" name="catalogue" datatype="char"/>
<FIELD id="_fband" name="band" datatype="char"/>
<DATA><TABLEDATA>
<TR><TD>08120809-0206132</TD><TD>23.2</TD><TD>.04</TD><TD>SDSS</TD><TD>g</TD></TR>
<TR><TD>08120809-0206132</TD><TD>25.2</TD><TD>.1</TD><TD>2MASS</TD><TD>J</TD></TR>
...
</TABLEDATA></DATA>
</TABLE>
```

Referrer and target in same TR:
Following example shows a LuminosityMeasurement stored in a TR together with information about its PhotometryFilter. This might be the result of a query like:

```
SELECT M.value,M.error,F.id,F.name
  FROM LUMINOISOTY_MEASUREMENT M
  ,      PHOTOMETRY_FILTER F
 WHERE F.ID=M.FILTER_ID
```

It joins the filter to the measurement and stores some of its fields.

```
<TABLE>
<GROUP utype="SAMPL:source/PhotometryFilter" id="_filters">
  <GROUP utype="vo-dml:ObjectType.ID"
    <FIELDref ref="_cat"/>
    <FIELDref ref="_bans"/>
  </GROUP>
  <FIELDref ref="_name" utype="SAMPL:source/PhotometryFilter.name"/>
</GROUP>
<FIELD name="catalogue" id="_cat" .../>
<FIELD name="band" id="_band" .../>
<FIELD name="name" id="_name" .../>
```

```
..
<DATA><TABLEDATA>
<TR><TD>SDSS</TD><TD>g</TD><TD>SDSS g-band</TD></TR>
<TR><TD>SDSS</TD><TD>u</TD><TD>SDSS u-band</TD></TR>
<TR><TD>2MASS</TD><TD>J</TD><TD>2MASS J-band</TD></TR>
</TABLDATA></DATA>
</TABLE>
<TABLE>
<GROUP utype="SAMPL:source/LuminosityMeasurement" id="SDSS_mags">
  <GROUP utype="SAMPL:source/LuminosityMeasurement.CONTAINER+vo-dml:Identifier ">
    <FIELDref ref="_container" />
  </GROUP>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.value"/>
  <PARAM name="type" utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude"/>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.error"/>
  <GROUP ref="_filter" utype="SAMPL:source/LuminosityMeasurement.filter/>
</GROUP>
<GROUP utype="SAMPL:source/PhotometryFilter" id="_filter">
  <FIELDref ref="_filter.id" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_filter.name" utype="SAMPL:source/PhotometryFilter.name"/>
</GROUP>
<FIELD id="_container" name="parentId" datatype="char"/>
<FIELD id="_mag" name="mag" datatype="float"/>
<FIELD id="_eMag" name="eMag" datatype="float"/>
<FIELD id="_filter.id" name="filter_id" datatype="char"/>
<FIELD id="_filter.name" name="filter_name" datatype="char"/>
<DATA><TABLEDATA>
<TR><TD>08120809-0206132</TD><TD>23.2</TD><TD>.04</TD><TD>SDSS:g</TD><TD>SDSS g-
band</TD></TR>
...
</TABLEDATA></DATA>
```

Note that both LuminosityMeasurement and PhotometryFlter are explicitly (and indirectly) represented by GROUPs that identify their type. The reference between the two is represented by a "GROUPref" construct without contents, i.e. *no* vo-dml: Identifier is used.

Referencing unspecified GROUP from within TABLE through vo-dml:Identifier:

It may be that objects of the same type are represented by different GROUPs in the same VOTable. It may then also be that from within a TABLE one may wish to reference one of these, but different rows may have references to different instances. The following example shows this.

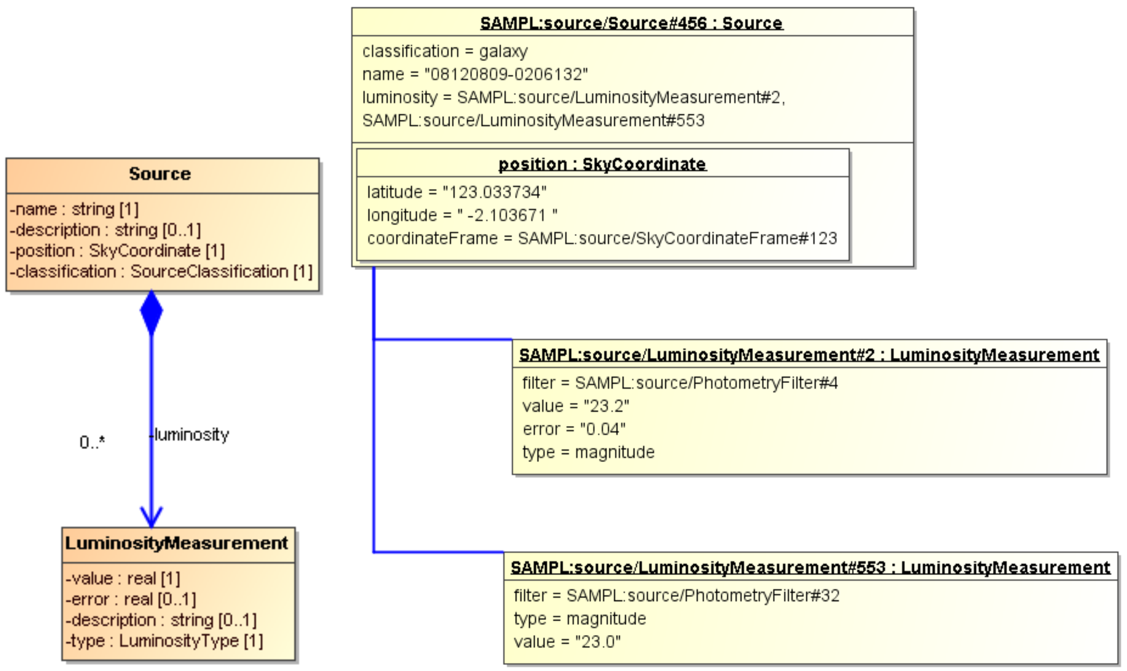
```
<GROUP utype="SAMPL:source/PhotometryFilter">
  <PARAM utype="vo-dml:ObjectType.ID" value="sdss:g" .../>
  <PARAM type="SAMPL:source/PhotometryFilter.name" value="SDSS g band" .../>
</GROUP>
<GROUP utype="SAMPL:source/PhotometryFilter">
  <PARAM utype="vo-dml:ObjectType.ID" value="sdss:u" .../>
  <PARAM type="SAMPL:source/PhotometryFilter.name" value="SDSS u band" .../>
</GROUP>
..
<TABLE>
<GROUP utype="SAMPL:source/LuminosityMeasurement" id="SDSS_mags">
  <GROUP utype="SAMPL:source/LuminosityMeasurement.CONTAINER+vo-dml:Identifier">
    <FIELDref ref="_container" utype="vo-dml:Identifier.field"/>
  </GROUP>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.value"/>
  <PARAM name="type" utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude"/>
  <FIELDref ref="__mag" utype="SAMPL:source/LuminosityMeasurement.error"/>
  <GROUP utype="SAMPL:source/LuminosityMeasurement.filter+vo-dml:Identifier">
    <FIELDref ref="_filter" utype="vo-dml:Identifier.field"/>
  </GROUP>
</GROUP>
<FIELD id="_container" name="parentId" datatype="char"/>
```

```
<FIELD id="_mag" name="mag" datatype="float"/>
<FIELD id="_eMag" name="eMag" datatype="float"/>
<FIELD id="_filter" name="filter" datatype="char"/>
<DATA><TABLEDATA>
<TR><TD>08120809-0206132</TD><TD>23.2</TD><TD>.04</TD><TD>sdss:g</TD></TR>
<TR><TD>08120809-0206132</TD><TD>22.5</TD><TD>.05</TD><TD>sdss:u</TD></TR>
...
</TABLEDATA></DATA>
</TABLE>
```

The mapping in this example is *indirect*, for the GROUP representing the target is not explicitly identified by the reference. This reference is here nevertheless represented by a GROUP with a FIELDref. The GROUP represents a copy the vo-dml:Identifier instance used as value of the remote object's vo-dml:ObjectType.ID; the FIELDref the single vo-dml:Identifier.field. Different rows reference different GROUPs, hence a "GROUPref" construct is not applicable here. To find the target the client will have to search the document for a GROUP representing the right type and with an ID with a single field of the right value. *If* the value of a TD could be a @ref, we could use XMLs ID/IDREF mechanism which in some tools facilitates the lookup. But see section [TBD] for ideas how to best analyze a VOTable with data model structure, in particular how to deal with referencing. [TBD do we want such a section? It might borrow from some more VO-URP implementation ideas.]

6.6 Mapping Collection

A collection is a composition relation between a parent ObjectType and a child, or *part*, ObjectType. The fact that objects can be stored in different ways implies many different ways in which the relation may need to be expressed. In XML serializations of a data model (such as used in VO-URP and the Simulation Data Model) one may choose to have the contained objects serialized *inside* the serialization of the parent. It is possible to do so in VOTable as well using child GROUP-s representing a complete child object embedded in the GROUP representing the parent object. This is natural for this relationship because a collection element is really to be considered as a *part* of the parent object.
This may be used to represent flattening of the parent-child relation. One or more child objects may be stored together with the parent object in the same row in a TABLE. Such a case is actually very common, for example when interpreting tables in typical source catalogues. These generally contain information of a source together with one or more magnitudes. The latter can be seen as elements form a collection of photometry points contained by the sources.
Hence a GROUP inside another GROUP MAY represent a collection. It can do so in different modes that are illustrated by the following examples and described in more detail in section 7.8.


SAMPL:source/Source#456 : Source
classification = galaxy
name = "08120809-0206132"
luminosity = SAMPL:source/LuminosityMeasurement#2, SAMPL:source/LuminosityMeasurement#553
position : SkyCoordinate
latitude = "123.033734"
longitude = " -2.103671 "
coordinateFrame = SAMPL:source/SkyCoordinateFrame#123
Source
-name : string [1]
-description : string [0..1]
-position : SkyCoordinate [1]
-classification : SourceClassification [1]
0..*
-luminosity
LuminosityMeasurement
-value : real [1]
-error : real [0..1]
-description : string [0..1]
-type : LuminosityType [1]
SAMPL:source/LuminosityMeasurement#2 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#4
value = "23.2"
error = "0.04"
type = magnitude
SAMPL:source/LuminosityMeasurement#553 : LuminosityMeasurement
filter = SAMPL:source/PhotometryFilter#32
type = magnitude
value = "23.0"

Container and contained objects in separate TABLE linked by "GROUPrefs":

The basic Object-Relational mapping has containers stored in one table, the objects in the collection in another. The child objects need a foreign key/pointer to the container object. In the example below we have the collection represented on the container using a "GROUPref" to the collection. The utype identifies *only* the collection, the @ref identifies the GROUP where the collection element(s) can be found. In this case that GROUP annotates a TABLE and has as @utype the type of the collection. It also contains a GROUP representing a pointer to the container object. Its utype is a concatenation of the CONTAINER utype and of the predefined vo-dml:Identifier to indicate it represents the ID of the container object.

```
...
<TABLE>
<GROUP utype="SAMPL:source/Source" id="_source">
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classifiication" value="galaxy"/>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
  <GROUP utype="SAMPL:source/Source.luminosity" ref="SDSS_mags"/>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
...
<TABLE>
<GROUP utype="SAMPL:source/LuminosityMeasurement" id="SDSS_mags">
```

```
  <GROUP utype="SAMPL:source/LuminosityMeasurement.CONTAINER+vo-dml:Identifier ">
    <FIELDref ref="_container"/>
  </GROUP>
  <FIELDref ref="_mag" utype="SAMPL:source/LuminosityMeasurement.value"/>
  <PARAM name="type" utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude"/>
  <FIELDref ref="_eMag" utype="SAMPL:source/LuminosityMeasurement.error"/>
  <FIELDref ref="_filter" utype="SAMPL:source/LuminosityMeasurement.filter+vo-
dml:Identifier"/>
</GROUP>
<FIELD id="_container" name="parentId" datatype="char"/>
<FIELD id="_mag" name="mag" datatype="float"/>
<FIELD id="_eMag" name="eMag" datatype="float"/>
<FIELD id="_filter" name="filter" datatype="char"/>
<DATA><TABLEDATA>
<TR><TD>08120809-0206132</TD><TD>23.2</TD><TD>.04</TD><TD>sdss:g</TD></TR>
<TR><TD>08120809-0206132</TD><TD>23.0</TD><TD>.03</TD><TD>sdss:r</TD></TR>
</TABLEDATA></DATA>
</TABLE>
```

Container and contained objects in same row in same TABLE:
A very typical case is the following example, which shows a Source object serialized in the same row as its luminosities. The Sample data model models luminosity measurements as a collection, which is flexible and allows measurements from different source to be combined in a natural manner. But for a given catalogue such as SDSS or 2MASS, it is known *a priori* how many magnitudes will be supplied and which bands they will correspond to. Hence for a given catalogue the natural representation is to simply add these as attributes to their model. Interestingly enough, the approach proposed here is able to support this mapping without any problem, even making a natural link to the actual photometry filter used for the measurements. And note that to indicate the fact the instances of the contained types are indirectly represented by these GROUPs, the utype attributes MUST be concatenations of the role (the collection) and the type.

```
<TABLE>
<GROUP utype="SAMPL:source/Source" id="_source">
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <PARAM name="type" utype="SAMPL:source/Source.classifiication" value="galaxy"/>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
  <GROUP utype="SAMPL:source/Source.luminosity+SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="__gmag" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude" .../>
    <FIELDref ref="_egMag" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="_gfilter" utype="SAMPL:source/LuminosityMeasurement.filter"/>
    </GROUP>
  </GROUP>
  <GROUP utype="SAMPL:source/Source.luminosity+SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="__rmag" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude" .../>
    <FIELDref ref="_erMag" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="_rfilter" utype="SAMPL:source/LuminosityMeasurement.filter"/>
    </GROUP>
  </GROUP>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
```

```
<FIELD id="_gmag" name="gmag" datatype="float"/>
<FIELD id="_egMag" name="egMag" datatype="float"/>
<FIELD id="_rmag" name="rmag" datatype="float"/>
<FIELD id="_erMag" name="erMag" datatype="float"/>
<TR>
<TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD><TD>23.2</TD><TD>.04</TD>
    <TD>23.0</TD><TD>.03</TD>
</TR>
...
</TABLE>
```

Container and collection all as direct instances in GROUPs:
When not using tables at all in a mapping, the individual elements form a collection can be directly represented inside the parent GROUP. To indicate this explicitly the utype attributes MUST be concatenations of the role (the collection) and the type.

```
<GROUP utype="SAMPL:source/Source" id="_source">
  <PARAM utype="SAMPL:source/Source.name" value="08120809-0206132" .../>
  <PARAM utype="SAMPL:source/Source.classifiication" value="galaxy" .../>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <PARAM utype="bSTC:SkyCoordinate.longitude" value="123.033734" .../>
    <PARAM utype="bSTC:SkyCoordinate.latitude" value="-2.103671" .../>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
  <GROUP utype="SAMPL:source/Source.luminosity+SAMPL:source/LuminosityMeasurement">
    <PARAM utype="SAMPL:source/LuminosityMeasurement.value" value="23.2" .../>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude" .../>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.error" value="0.04" .../>
    <GROUP ref="_gfilter" utype="SAMPL:source/LuminosityMeasurement.filter"/>
    </GROUP>
  </GROUP>
  <GROUP utype="SAMPL:source/Source.luminosity+SAMPL:source/LuminosityMeasurement">
    <PARAM utype="SAMPL:source/LuminosityMeasurement.value" value="23.0" .../>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.type" value="magnitude" .../>
    <PARAM utype="SAMPL:source/LuminosityMeasurement.error" value="0.03" .../>
    <GROUP ref="_rfilter" utype="SAMPL:source/LuminosityMeasurement.filter"/>
    </GROUP>
  </GROUP>
</GROUP>
```

6.7 Mapping value types

The examples above started from the assumption that the basis of a serialization was an ObjectType. Value types only show up as attributes. There may be some use cases howevere where one wishes to indicate *only* that a certain column or set of column represent some known value type. One reason may be that a standard, global data model does not exist that defines ObjectType-s matching the one in one's serialization, but that one can identify some sub-components that could be mapped to a DataType for example.
In fact the Sample used here is an example. It is a model for Source-s. The IVOA does currently not have an accepted model for this concept, though attempts in this direction have been made[4]. This implies many tables of interest in the VO can currently not formally declare they store instances of a Source. However they could declare they have columns that together correspond to a coordinate on the

[4] See http://wiki.ivoa.net/twiki/bin/view/IVOA/IVAODMCatalogsWP.

sky in the STC model. This could lead to a VOTable fragment as the following, which is a version of an example in 6.4, but with altered utypes, and removal of the GROUP representing the Source.

```
<TABLE>
<GROUP utype="bSTC:SkyCoordinate ">
  <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
  <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
  <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
</GROUP>
<FIELD name="designation" ID="_designation" .../>
<FIELD name="ra" ID="_ra" unit="deg" .../>
<FIELD name="dec" ID="_dec"  unit="deg" .../>
<TR><TD>08120809-0206132</TD><TD>123.033734</TD><TD>-2.103671</TD></TR>
...
</TABLE>
```

Tools that understand" STC may be able to do something with this annotation, even though they cannot know what role the coordinate plays.
Another example arises from a possible access protocol specification. Say Simple Cone Search (SCS) would declare that the result of a request MUST be a VOTable with a GROUP representing the actual request consisting of a position and a search radius. It could insist the position must be serialized using a GROUP representing an STC coordinate following our data modeling serialization prescription. E.g. as in the following example:

```
<GROUP utype="bSTC:SkyCoordinateFrame" ID="_icrs">
  <PARAM utype="bSTC:SkyCoordinateFrame.name" value="ICRS" .../>
  <PARAM utype="bSTC:SkyCoordinateFrame.equinox+bSTC:Equinox" value="J2000.0" .../>
</GROUP>
...
<GROUP name="SCS">
  <INFO value="The SCS request "/>
  <PARAM name="SR" datatype="float" utype="ivoa_1.0:stdtypes/real"/>
  <GROUP name="center" utype="bSTC:SkyCoordinate">
    <INFO value="The center coordinate of the simple cone search"/>
    <PARAM name="ra" utype="bSTC:SkyCoordinate.longitude" value="123.00000"
datatype="float"/>
    <PARAM name="dec" utype="bSTC:SkyCoordinate.latitude" value="-2.10000"
datatype="float"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
...
```

Note, this example even assigns a utype for the search radius SR, identifying it as the PrimitiveType "ivoa_1.0:stdtypes/real". This shows that also PrimitiveType-s and Enumerations could be used directly, i.e. outside of a role they play. The use of this is probably limited, but we may allow it (see section 7.2 for details and constraints.
One could argue that alternatively a standard protocol like SCS might define a little standard data model to represent its full request and use it in the serialization of result. In that case also the parent group, currently named "SCS'', could have been declared to represent say an "scs:Request", as follows:

```
 <GROUP utype="bSTC:SkyCoordinateFrame" ID="_icrs">
  <PARAM utype="bSTC:SkyCoordinateFrame.name" value="ICRS" .../>
  <PARAM utype="bSTC:SkyCoordinateFrame.equinox+bSTC:Equinox" value="J2000.0" .../>
</GROUP>
...
<GROUP name="SCS" utype="scs:Request">
  <INFO value="The SCS request"/>
  <PARAM name="SR" datatype="float" utype="scs:Request.SR "/>
  <GROUP name="center" utype="scs:Request.center+bSTC:SkyCoordinate">
    <INFO value="The center coordinate of the simple cone search"/>
    <PARAM name="ra" utype="bSTC:SkyCoordinate.longitude" value="123.00000"
datatype="float"/>
    <PARAM name="dec" utype="bSTC:SkyCoordinate.latitude" value="-2.10000"
datatype="float"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
</GROUP>
...
```

[TBD I assume we do not have to spell out the SCS data model?]

7 Patterns for annotating VOTable: specification

In this section we list all legal mappings where a VOTable element uses its @utype to identify one or two VO-DML elements; we describe how such an annotation should be interpreted and what restrictions there are on the association. In its subsections it explicitly lists possible annotations of a VOTable with a utype value that one may encounter following this spec.
The organization of the following sections is based on the different VO-DML concepts that can be represented. Each of these subsections contains sub-subsections which represent the different possible ways the concept may be encountered in a VOTable and discuss rules and constraints on those annotations. We start with Model, and then we discuss value types (PrimitiveType, Enumeration and DataType) and Attributes. Then ObjectType and the relationships, Collection, Reference and Inheritance (extends). We leave out Package as we have not yet found a convincing use case why one might want to point at one from a VOTable (though we did not really look for a reason either).
The title of each subsection describing a legal mapping is structured according to a "grammar'. It starts with a VOTable element on the left, the utype assignment and possible other constraints in square brackets []. A possible context is indicated by an ∊ with to its right a container formatted like the element of the left. Finally, an '→' indicates a relation to another element; this will generally be defined in the same VOTable, but might be serialized elsewhere.
For example the section with title “GROUP[@utype ⇒ ObjectType]” discusses the rules on linking a GROUP to an ObjectType; similarly “GROUP[@utype ⇒ ObjectType] ∊ RESOURCE” discusses the same case restricted to GROUPs defined directly on a RESOURCE, i.e. *not* on a TABLE or another GROUP.
Where relevant, subsections will be used for different cases or contexts within which to interpret the annotation. Appendix A contains an attempt at formalizing this grammar so one may argue about valid instances of it in a more explicit manner. This is TBD and likely too scary. Instead each section should spell state the meaning of the title explicitly.
Some comments on how we refer to VOTable and VO-DML elements

- When referring to VOTable elements we will use the notation by which these elements will occur in VOTable documents, i.e. in general “all caps”, E.g. GROUP, FIELD, (though FIELDref).

- When referring to rows in a TABLE element in a VOTable, we will use TR, when referring to individual cells, TD. Even though such elements only appear in the TABLEDATA serialization of a TABLE. When referring to a column in the TABLE we will use FIELD, also if we do not intend the actual FIELD element annotating the column.
- When referring to an XML attribute on a VOTable element we will prefix it with an '@', e.g. @utype, @ref.
- References to VO-DML elements will be capitalized, using their VO-DML/XSD type definitions. E.g. ObjectType, Attribute.
- Some mapping solutions require a reference to a GROUP elsewhere in the VOTable. We refer to such a construct as a "GROUPref. In the examples we use a concrete implementation of this concept which (currently) does not exists in VOTable (and is likely not desired). The solution assumed in this document is to use a GROUP with a @ref attribute which we will assumes *always* identifies another GROUP in the same document. This target GROUP must have an @id attribute. In cases where this is important we will indicate that this combination is to be interpreted as a "GROUPref", including the quotes.

The following list defines some shorthand phrases (underlined) which we use in the descriptions below:

- Generally when using the phrase type we mean a "kind of" type as defined in VO-DML. These are PrimitiveType, Enumeration, DataType and ObjectType.
- With atomic type we will mean a PrimitiveType or an Enumeration as defined in VO-DML.
- A structured type will refer to an ObjectType or DataType as defined in VO-DML.
- With a property available on or defined on a (structured) type we will mean an Attribute or Reference, or (in the case of ObjectTypes) a Collection defined on that type itself, or inherited from one of its base class ancestors.
- A VO-DML type plays a role in the definition of another (structured) type if the former is the declared data type of a property available on the latter.
- When writing that a VOTable element represents a certain VO-DML type, we mean that the VOTable element is mapped either directly to the type, or that it identifies a role played by the type in another type's definition.
- A descendant of a VOTable element is contained in that element, or in a descendant of that element. This is a standard recursive definition and can go up the hierarchy as well: an ancestor of an element is the direct container of that element, or an ancestor of that container.

Short list of all patterns:

- ObjectType direct to GROUP
- ObjectType indirect to GROUP in TABLE
- DataType direct to GROUP
- DataType indicrect to GROUP in TABLE
- Atomic attribute to FIELDref

- Atomic attribute to PARAM in GROUP
- Atomic attribute to PARAMref
- Structured attribute to GROUP in GROUP
- Reference to GROUP in GROUP
    - As "GROUPref"
    - As vo-dml:Identifier (~foreign key)
- Collection element to GROUP in GROUP
- Collection to GROUPref in GROUP (indicates remote table/element)

7.1 Model

There has been quite some discussion how to indicate which models provide the utypes used in a certain VOTable. Options are to make an explicit statement through some INFO element for example, or to leave model usage implicitly defined through a predefined set of model prefixes. We here follow the former, explicit approach. We propose that for each model that is used an INFO element should exist which identifies the location of the VO-DML file representing the model. This element should also define the prefix used by utypes corresponding to that model.

7.1.1 GROUP[@utype ⇒ Model] ∊ VOTABLE

A GROUP element with @utype attribute identifying a Model and placed directly under the root VOTABLE element indicates that the corresponding VO-DML model is used in @utype associations.

Restrictions

- GROUP element must exist directly under VOTABLE and have @utype=”vo-dml:Model”
- Must have child PARAM element with @utype=”vo-dml:Model.url” and @value the url of the VO-DML document representing the model. @name is irrelevant, @datatype=”char” and arraysize=”*”.
- SHOULD have child PARAM element with @utype=”vo-dml:Model.prefix” and @value the prefix used in the current VOTable document for utypes coming from that model. @name is irrelevant, @datatype=”char” and arraysize=”*”.
  TBD How do we deal with prefix? Can it be different from utype prefix in VO-DML document? Do utypes in VO-DML document *have* a prefix?

Example

```
<VOTABLE>
<GROUP utype="vo-dml:Model" name="Sample">
  <PARAM utype="vo-dml:Model.url" name=”url” datatype=”char” arraysize=”*”
value="http://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/sample/Sample.vo-
dml.xml"/>
  <PARAM utype="vo-dmlLModel.prefix" name=”prefix” datatype=”char” arraysize=”*”
value="SAMPL"/>
</GROUP>
<GROUP utype="vo-dml:Model" name="IVOA_Profile">
<PARAM utype="vo-dml:Model.url" name=”url” datatype=”char” arraysize=”*”
value="http://volute.googlecode.com/svn/trunk/projects/dm/vo-
dml/models/profile/IVOA_Profile.vo-dml.xml"/>
```

```
<PARAM utype="vo-dml:Model.prefix" value="ivoa_1.0" name=”prefix” datatype=”char”
arraysize=”*” />
</GROUP>
...
```

7.2 Atomic types

DEPRECATED propose not to use this pattern

[NOTE in telecom 2013-04-09 we decided that FIELDs and standalone PARAMs would *not* fall under the spec. I.e. the use of utype attributes on those elements is allowed, but not standardized by this spec. Same for TABLE, RESOURCE, INFO. These can be used for example by protocols if these wish to define a standard VOTable-based result format, but we suggest that new protocols should use the standard form as well.]

Instances of atomic types (PrimitiveType and Enumeration in VO-DML) are *values*. In general these will be used as values for Attributes in instances of structured types (DataType and ObjectType) as will be described below. But there may be valid use cases for indicating simply that certain values occurring in a VOTable’s data are instances of the VO-DML type. Such atomic values will be stored in a TD or in the @value attribute of a PARAM. They will be annotated by the @utype attribute of the corresponding FIELD or the PARAM itself respectively.

One might expect that Enumerations are represented by a FIELD or PARAM with a VALUES component. Its Literals might correspond to the OPTION values. *How* that association would be made is TBD. After all OPTION elements do not have a @utype attribute. Until that time the OPTION values may have to be identical to the Literals.

TBD We might wish to consider the more complex (if not perverse) possibility where an atomic value is cut up in two or more parts, each of these stored separately. An example could be a ‘date’, which might be distributed over three columns representing year, month and day. Do we want to support this?

Question: is it useful to declare that a value or set of values in a column represent some primitive type or enumeration? Could we not use UCDs for this?

Possible answer: This would seem to depend on the “expressivity” of the type. In general FIELD-s will be used as attributes of structured types, i.e. through the role the type plays in the definition of a parent ObjectType or DataType. But if no model exists that allows a parent to be defined, this link to a type might still be useful, probably more so for Enumerations.

7.2.1 FIELD[@utype ⇒ PrimitiveType]

A FIELD’s @utype may identify a PrimitiveType directly (and without a role!). This indicates that the TDs corresponding to the FIELD hold values whose type may be interpreted as the identified PrimitiveType.

Restrictions

- This FIELD SHOULD NOT be used from within a GROUP with a FIELDref. (For reasons of trying to avoid redundancy.) If this is done anyway the @utype annotation of the

FIELDref MUST refer to an Attribute whose data type is the same as the PrimitiveType identified by the FIELD. Otherwise the annotation is invalid.

- The FIELD's @datatype SHOULD (MUST?) define a valid serialization type for the PrimitiveType. (For reasons of translational semantics: It must be possible to create an instance of the PrimitiveType from the value stored in the VOTable).

Example

```
...
<TABLE>
...
<FIELD name="designation" ID="_designation" utype="ivoa_1.0:stdtypes/string">
<DESCRIPTION>source designation formed from sexigesimal coordinates</DESCRIPTION>
</FIELD>
...
</TABLE>
...
```

7.2.2 FIELD[@utype ⇒ Enumeration & VALUES]

A FIELD with a VALUES element may identify an Enumeration directly through its @utype. This indicates that the TDs in the FIELD hold values whose type may be interpreted as the identified Enumeration. The OPTION values should be interpreted in terms of the Literals of the Enumeration.

Restrictions

- A FIELD directly annotated with a type SHOULD NOT be used from within a GROUP with a FIELDref. (For reasons of trying to avoid redundancy).
  If this is done anyway the @utype annotation of the FIELDref MUST refer to an Attribute whose data type is the same as the Enumeration identified by the FIELD. Otherwise the annotation is invalid.
- The FIELD's data type SHOULD (MUST?) be an exact serialization type for the Enumeration (For reasons of translational semantics: It must be possible to create an instance of the Enumeration from the value stored in the VOTable).
- The FIELD definition SHOULD (MUST?) have a VALUES element with OPTIONS corresponding to the Literals in the Enumeration's definition.
  How that correspondence is made is TBD. A possible solution might be to have a link (through @utype) from the different OPTION values to the corresponding Literal. But OPTION does not have a @utype attribute. So currently the only way to make the identification is if the OPTION values are exact copies of the Literals.

Example

```
...
<TABLE>
...
<FIELD name="type" utype="SAMPL:source/SourceClassification">
<VALUES>
<OPTION value="star"/>
<OPTION value="galaxy"/>
<OPTION value="AGN"/>
```

```
<OPTION value="planet"/>
<OPTION value="unknown"/>
</FIELD>
...
</TABLE>
...
```

7.2.3 PARAM[@utype ⇒ PrimitiveType]

A PARAM's @utype may identify a PrimitiveType directly. This indicates that the @value of the PARAM may be interpreted as the identified PrimitiveType. The rules and interpretation are similar to those of a FIELD's mapping to a PrimitiveType (see section 7.2.1).

Restrictions

- A PARAM annotated with a @utype like this FIELD SHOULD NOT be used from within a GROUP with a PARAMref. (For reasons of trying to avoid redundancy.) If this is done anyway the @utype annotation of the PARAMref MUST refer to an Attribute whose data type is the same as the PrimitiveType identified by the PARAM. Otherwise the annotation is invalid.
- The PARAM's data type SHOULD (MUST?) be a valid serialization type for the PrimitiveType. (For reasons of translational semantics: It must be possible to create an instance of the PrimitiveType from the value stored in the VOTable).

Example

```
<GROUP name="SCS">
<PARAM name="SR" utype="ivoa_1.0:stdtypes/real"/>
...
</GROUP>
```

7.2.4 PARAM[@utype ⇒ Enumeration & VALUES]

A PARAM's @utype may identify an Enumeration directly. This indicates that the @value of the PARAM may be interpreted as the identified Enumeration. The rules and interpretation are similar to those of a FIELD's mapping to a Enumeration, see section 7.2.2.

Restrictions

- A PARAM mapped directly to an Enumeration MUST NOT be contained in a GROUP that is mapped to a structured type (see below).
- The PARAM's data type SHOULD (MUST?) be a valid serialization type for the Enumeration. (For reasons of translational semantics: It must be possible to create an instance of the Enumeration from the value stored in the VOTable).
- The PARAM definition SHOULD (MUST?) have a VALUES element with OPTIONS corresponding to the Literals in the Enumeration's definition.
  How that correspondence is made should follow the same rules as for a FIELD mapped to an Enumeration in section 7.2.2.

7.3 DataType

A DataType instance (also a *value*) is structured; it consists of values assigned to each of its attributes and possibly references. To represent the complete instance of a DataType the various attributes (and references) must be grouped together; in VOTable this is done using a GROUP element. GROUPs in fact can play two different roles, depending on where the instance's data is really stored. If all values are eventually stored exclusively in PARAMs in the GROUP, or possible outside the GROUP but accessed through PARAMrefs, the GROUP *directly* represents a complete instance. If even only one of the attributes is stored in a FIELD and accessed through a FIELDref, the GROUP *indirectly* represents possibly multiple instances, one for each TR. This is also true in any child GROUP contains a FIELDref and so on. We will use these terms, direct and indirect representation all through the document. And note that this same classification holds for ObjectTypes discussed in 7.5, though with a twist related to possible child GROUPs representing Collections.
The attribute values of a DataType are stored according to the prescription for storing instances of their data type. For attributes with declared data type a PrimitiveType or Enumeration section 7.2 provides details. If the attribute's data type is itself a DataType the prescription in the current section should be used recursively. DataType-s can also have Reference-s to ObjectType-s, but a discussion of how to store References is deferred to section 7.7, after the discussion of storing ObjectType instances, which is provided in section 7.5.
Polymorphism
When assigning values to attributes and references an important issue arises due to polymorphism. Though an attribute is assigned a certain data type in the data model, instances of the attribute MAY have a different data type, namely one that is (ultimately) a subclass of the declared type. This is especially important for attributes with a structured DataType. Subclasses of PrimitiveTypes and Enumerations remain atomic, but subclasses of DataTypes may add attributes and references to the ones inherited from the base class. For this reason we advocate that the utype attribute of a component representing an Attribute SHOULD (MUST?) be a concatenation of the utypes of the Attribute *and* the type. See the discussion on the syntax of the utype attribute in section 5.

7.3.1 GROUP[@utype ⇒ DataType]

A GROUP's @utype may identify a DataType directly. Depending on the contents and context, i.e. the location of the GROUP within other VOTable elements, such a mapping indicates the existence of one or more instances of the DataType (i.e. *values* really, as a DataType is a value type). Different cases are discussed in subsets. In all cases the following restrictions hold:
Restrictions

- The components of a GROUP mapped to a DataType MUST be mapped to properties defined on that DataType. The following options are available:

- FIELDref[@utype ⇒ Attribute [+ (PrimitiveType | Enumeration)]]
  ∊ GROUP[@utype ⇒ (DataType | ObjectType)] (see 7.4.1)
- PARAM [@utype ⇒ Attribute + [(PrimitiveType | Enumeration)]]
  ∊ GROUP[@utype ⇒ (DataType | ObjectType)] (see 7.4.2)
- PARAMref [@utype ⇒ Attribute + [(PrimitiveType | Enumeration)]]
  ∊ GROUP[@utype ⇒ (DataType | ObjectType)] (see 7.4.3)
- GROUP [@utype ⇒ Attribute [+ DataType]]
  ∊ GROUP[@utype ⇒ (DataType | ObjectType)] (see 7.4.4)
- GROUP [@utype ⇒ Reference [+ ObjectType]]
  ∊ GROUP[@utype ⇒ (DataType | ObjectType)] (see 7.7)

- If the GROUP is contained in a parent GROUP, the parent MUST NOT use its @utype attribute. This is consequence of previous rule really.
- ...

7.3.1.1 GROUP[@utype ⇒ DataType] ∊ RESOURCE

A GROUP in a RESOURCE cannot have FIELDrefs, only PARAMs, PARAMrefs and GROUPs. The GROUP represents therefore a single instance of the DataType directly as defined in 7.3, with the PARAM-s etc. providing the values of the components of the DataType.
The possible role this instance plays in the VOTable document must have been defined outside of any mapping to a data model. After all, in contrast to instances of ObjectTypes, the existence of instances of value types need not be explicitly stated (see the description of value types in section TBD). An example could be a VOTable containing the result of a simple cone search. A RESOURCE in that document might contain a GROUP representing the position of the cone, which may be mapped through its @utype to a DataType “stc:SkyCoordinate” (if such a type existed, our sample model contains a type like it).

Example

```
<RESOURCE>
...
<GROUP utype="bSTC:SkyCoordinate">
  <INFO value="The center coordinate of the simple cone search"/>
  <PARAM name="ra" utype="bSTC:SkyCoordinate.longitude" value="123.00000"/>
  <PARAM name="dec" utype="bSTC:SkyCoordinate.latitude" dec=-2.10000/>
  <PARAMref ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
</GROUP>
...
```

7.3.1.2 GROUP[@utype ⇒ DataType &¬FIELDref] ∊ TABLE

A GROUP defined on a TABLE, annotated with a DataType, but without FIELDref-s in the GROUP or any of its descendant GROUPs, represents a DataType instance directly, just as was the case for the GROUP in a Resource in the previous subsection. In this case we can then interpret the GROUP “merely” as a structured PARAM following the VOTable spec [REF] “A PARAM may be viewed as a FIELD which keeps a *constant value* over all the rows of a table". I.e. a GROUP without FIELDref is a structured set of columns all with the same value in each row. As the GROUP is mapped directly to a DataType rather than to a role, the use of such a GROUP in the context of the TABLE is not defined by a mapping, but possibly by a protocol as described in section 7.3.1.1.

7.3.1.3 GROUP[@utype ⇒ DataType & FIELDref] ∊ TABLE

If the GROUP is a descendant of a TABLE, is annotated with a DataType and contains at least one FIELDref element, either directly or in a descendant GROUP, the GROUP represents an instance of the DataType for each TR. The interpretation of the possible role it plays in terms of

Example

```
<TABLE>
...
  <GROUP utype="bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
...
<FIELD name="ra" ID="_ra" unit="deg">
<DESCRIPTION>right ascension (J2000 decimal deg)</DESCRIPTION>
</FIELD>
<FIELD name="dec" ID="_dec"  unit="deg">
<DESCRIPTION>declination (J2000 decimal deg)</DESCRIPTION>
</FIELD>
...
```

7.3.1.4 GROUP[@utype ⇒ DataType] ∊ GROUP[¬@utype]

A GROUP may be defined inside another GROUP and be mapped to a DataType. There are some restrictions. NONE of its ancestor GROUPs MAY be mapped to a data model element. This would conflict with rules of mapping such an ancestor GROUP that state that children of such GROUPs MUST be mapped to properties of the structured type represented by the containing GROUP (see the restrictions defined in 7.3.1).
Whether the GROUP represents the DataType instance directly or indirectly is independent of this embedding in a parent GROUP, only of the existence of a FIELDref in it or its descendants.
Example A GROUP representing parameters of a Simple Cone Search

```
<GROUP name="SCS">
...
<GROUP name="center" utype="bSTC:SkyCoordinate">
  <INFO value="The center coordinate of the simple cone search"/>
  <PARAM name="ra" utype="bSTC:SkyCoordinate.longitude" value="123.00000"/>
  <PARAM name="dec" utype="bSTC:SkyCoordinate.latitude" dec=-2.10000/>
  <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
</GROUP>
</GROUP>
```

7.4 Attribute

An attribute is the role a value type plays in the definition of a structured type. The mapping of the attribute is therefore almost identical to that of its type itself; similar elements can be used and the utype attribute may contain the utype of the actual type (must do so to support polymorphism). To annotate an element as attribute one MUST use also the utype of the attribute itself. Furthermore the element representing the attribute MUST be contained in a GROUP that represents the containing structured type.
How all of this is achieved in practice is described in the following subsections.

7.4.1 FIELDref[@utype ⇒ Attribute [+ (PrimitiveType | Enumeration)]]

[NOTE the decision of telecom 2013-04-09 to *not* use concatenation of utypes, but a child PARAM to indicate specialized type casting implies that FIELDref-s, PARAMref-s and PARAM-s can *not* be explicitly casted.]
A FIELDref contained in a GROUP with a VO-DML annotation, MUST through its @utype declare that the FIELD it refers to stores an Attribute. [DEPRECATED It MAY (MUST?) add the utype identifying the actual type (a PrimitiveType or an Enumeration) that the FIELD represents separated by a '+' from the utype of the Attribute.]

Restrictions

- The Attribute MUST be available to the structured type represented by the containing GROUP.
- The Attribute MUST have an atomic type.
- The datatype of the referenced FIELD must be compatible with the declared datatype of the Attribute, as well as the actual detatype indicated by the relevant part of the utype combination.
-

Example

See example in TBD.

7.4.2 PARAM[@utype ⇒ Attribute [+ (PrimitiveType | Enumeration)]] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩]

When defined inside of a GROUP that represents a structured type, a PARAM MAY represent an Attribute defined on the type. Which Attribute it represents it declares through its @utype. It MAY (MUST?) add the utype identifying the actual type (a PrimitiveType or an Enumeration) that the FIELD represents separated by a '+' from the utype of the Attribute.

Restrictions

- The Attribute must have an atomic data type.
- The PARAM's @datatype SHOULD be a compatible, valid serialization type for the type of the Attribute. (For reasons of translational semantics: It must be possible to create an instance of the atomic type from the value stored in the PARAM's @value).
- …

Example
See example in TBD.

7.4.3 PARAMref[@utype ⇒ Attribute [+(PrimitiveType | Enumeration)]]

Inside a GROUP a PARAMref identifies a PARAM defined inside the same RESOURCE or TABLE where the GROUP is defined. Using its @utype the PARAMref can annotate the PARAM as holding the value of an Attribute.
Restrictions

- The Attribute must be available to the structured type represented by the containing GROUP.
- Attribute's data type must be atomic.
- The PARAM must obey the restrictions defined for the annotation of a PARAM by the Attribute's type, if a PrimitiveType see 7.2.3, if an Enumeration see 7.2.4.

Example
TBD

7.4.4 GROUP[@utype ⇒ Attribute(DataType)] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩]

A GROUP that is defined inside parent GROUP representing a structured type can be mapped to an Attribute with a (structured) DataType. The annotation will be a concatenation of the utype of the Attribute and of the actual DataType it represents.
Restrictions

- The Attribute represented by the child GROUP must be available on the structured type represented by the parent GROUP.
- …

Example
```
<GROUP utype="SAMPL:source/Source">
  ...
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
...
</GROUP>
```

7.5 ObjectType

7.5.1 GROUP[@utype ⇒ ObjectType]

A GROUP's @utype may identify an ObjectType. Depending on the context, i.e. the location of the GROUP within other VOTable elements, such a mapping indicates the existence of one or more instances of the ObjectType. These may

be directly represented by the GROUP, i.e. the GROUP *is* the instance, or the GROUP may indicate their existence when it is used to annotate a TABLE. These two cases are treated in the subsections below.
Much of the definition here mirrors that of the mapping of GROUP to DataType. The main difference is that having a GROUP annotated directly mapped an ObjectType, rather than to a role the ObjectType plays (say as reference or collection) is always meaningful. For the existence of instances of ObjectTypes cannot be inferred from the definition of the type.

Restrictions

- The components of a GROUP mapped to an ObjectType MUST be mapped to properties defined on that ObjectType. The following options are available:
  - FIELDref ∈ GROUP ⇒ Attribute ∈ ObjectType with atomic datatype (see)
  - FIELDref ∈ GROUP ⇒ ID ∈ ObjectType with atomic datatype (see
  - PARAM ∈ GROUP ⇒ Attribute ∈ ObjectType with atomic datatype (see)
  - PARAM ∈ GROUP ⇒ ID ∈ ObjectType with atomic datatype (see)
  - PARAMref ∈ GROUP ⇒ Attribute ∈ ObjectType with atomic datatype (see)
  - PARAMref ∈ GROUP ⇒ ID ∈ ObjectType with atomic datatype (see)
  - GROUP ∈ GROUP ⇒ Attribute ∈ ObjectType with structured datatype (see)
  - GROUP ∈ GROUP ⇒ ID ∈ ObjectType with atomic datatype (see)
  - GROUP ∈ GROUP ⇒ Collection ∈ ObjectType (see)
  - GROUP ∈ GROUP ⇒ Reference ∈ ObjectType (see)
  - GROUP ∈ GROUP ⇒ Container ∈ ObjectType (see)
  - GROUP ∈ GROUP ⇒ Extends ∈ ObjectType

7.5.1.1 GROUP[@utype ⇒ ObjectType] ∈ RESOURCE

GROUP as instance on RESOURCE

GROUP cannot have FIELDref-s, not can any of possible child GROUPs. Hence GROUP indicates an instance of the indicated ObjectType.

Example

```
<RESOURCE>
<GROUP utype="bSTC:SkyCoordinateFrame" ID="_icrs">
  <PARAM name="name" datatype="char" utype="bSTC:SkyCoordinateFrame.name" value="ICRS"/>
  <PARAM name="equnox" datatype="char"
utype="bSTC:SkyCoordinateFrame.equinox+bSTC:Equinox" value="J2000.0"/>
</GROUP>
...
</RESOURCE>
```

7.5.1.2 GROUP[@utype ⇒ ObjectType & ¬ FIELDref] ∈ TABLE

GROUP as instance on TABLE

¬ FIELDref means that the GROUP has not FIELDref-s directly under it, not in any GROUP *representing an Attribute or Reference*. Having no FIELDref, the GROUP represents an instance of the indicated ObjectType. Some of its child GROUPs may have a FIELDref, but only if they annotate an ObjectType.

It is allowed for a GROUP representing a Collection to have a FIELDref. In that case the GROUP represents an instance with ALL the TRs representing objects from the collection. There MUST NOT be more than 1 GROUP representing a Collection in this manner.

7.5.1.3 GROUP[@utype ⇒ ObjectType & FIELDref] ∊ TABLE

GROUP as annotation

A GROUP defined as child of a TABLE and having at least 1 FIELDref annotates the TRs in the TABLE, indicating that each row in the table represents at least part of an instance of the indicated ObjectType.

A child GROUP MAY contain a FIELDref, but *only* if that child GROUP is annotated to represent a collection defined on its GROUP. This represents one way of mapping a parent-child composition relation. The TABLE rows represent instances of the child ObjectType represented by the child GROUP. The parent GROUP represents the single instance of the container ObjectType.

Example

```
<TABLE>
<GROUP utype="SAMPL:source/Source">
  <FIELDref ref="_designation" utype="vo-dml:ObjectType.ID"/>
  <GROUP utype="SAMPL:source/Source.position+bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
...
</GROUP>
<FIELD name="designation" ID="_designation" utype="ivoa_1.0:stdtypes/string">
<DESCRIPTION>source designation formed from sexigesimal coordinates</DESCRIPTION>
</FIELD>
<FIELD name="ra" ID="_ra" unit="deg">
<DESCRIPTION>right ascension (J2000 decimal deg)</DESCRIPTION>
</FIELD>
<FIELD name="dec" ID="_dec"  unit="deg">
<DESCRIPTION>declination (J2000 decimal deg)</DESCRIPTION>
</FIELD>
...
```

7.6 vo-dml:ObjectType.ID and vo-dml:Identifier

It is assumed that *objects* (i.e. instances of an ObjectType) have an implicitly defined attribute that identifies instances uniquely; we will refer to this attribute as the ID. This attribute may be required for a serialization of an object, but is not modeled explicitly in VO-DML models. The reason is that it often depends on the particular physical representation of model instances how the ID is represented. For example a relational database might use one or more INTEGER or BIGINT columns in the table storing the instance, whereas an XML document might have an ID attribute, or a URI. In fact different serializations of the same object may use different ways to represent the ID.

Hence an instance of an ID can generally be serialized to one or more columns in a TABLE, or to one or more PARAMs in a GROUP or even some combination of the two. The ID may therefore be represented by a single FIELDref or PARAM

or PARAMref in the annotating GROUP, or possibly a child GROUP representing an ID with structure. Whatever the choice, an annotation must be provided that indicates the role of the element in the definition of the object serialization.
We may want to use the same logic as we apply for all such annotations, i.e. a utype MUST point to an explicitly defined model. To be able to do so we have created a special data model called VO-DML. This model is located in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/vo-dml/VO-DML.vo-dml.xml[5], has prefix "vo-dml" and is discussed further in 4.3.
We propose that the utype "vo-dml:ObjectType.ID" is used to annotate an attribute on an ObjectType that holds on to the ID. It's type is defined as "vo-dml:Identifier". This is a type that contains an array of strings named "field". Since it is a datatype, in the current proposal an ID attribute should be mapped to a GROUP. The special role that the VO-DML model plays in this proposal might be used to make simplifying annotations. For example the majority of all cases will be where a single field is used to hold the identifier. Hence a GROUP, as is otherwise mandated for DataType mappings, might seem cumbersome. Below we assume any FIELDref, PARAM(ref) or child GROUP might be annotated to be an ID. But there must be only one such annotated child element. This is TBD.

7.6.1 FIELDref[@utype ⇒ ID] ∈ GROUP[@utype ⇒ ⟨ObjectType⟩]

The GROUP representing the ObjectType has an ID consisting of a single FIELD. Each row in the TABLE has an instance identified by that FIELD.

7.6.2 PARAM[@utype ⇒ ID] ∈ GROUP[@utype ⇒ ⟨ObjectType⟩ & ¬ FIELDref]

A GROUP representing an ObjectType has a single PARAM to identify it. Parent GROUP SHOULD NOT be contained in TABLE, or have no FIELDref. Otherwise it would indicate multiple rows all with same ID, which would be odd. To be worked out (might this be used as a way to indicate replication of a parent object type in a table containing a complete collection? I'd say this should be modeled differently, using a CONTAINER annotation.) TBD

7.6.3 GROUP [@utype ⇒ ID] ∈ GROUP[@utype ⇒ ⟨ObjectType⟩]

GROUP represents a struct as ID. Containing objects can only be FIELDref, PARAMref and or PARAM. Order and data type of these is important as references must replicate structure.

7.7 Reference

A Reference is a relation between a structured type (the “referrer”, an ObjectType or DataType) and an ObjectType, the “target object”, or “referenced object”. The reference is a property of the referrer and the target object can be referenced by many referrers. The GROUP representing the referrer should hold

[5] This model still depends for its PrimitiveType on the IVOA_Profile model in https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/profile/IVOA_Profile.vo-dml.xml. We might decide to merge the two into a VO-DML Profile or so. That model would have to be part of a spec and be separately maintained.

on to an element representing the reference. The minimum requirement on this element MUST allow one to identify the target object.
In general one *cannot* assume that the target object can be found in the same element that contains the referrer. For example a standard Object Relational mapping will store both referrer and target objects in tables that will generally be different (unless referrer and target happen to have the same type). The referrer table will have a “foreign key” consisting of one or more columns that correspond to primary key columns in the target table. This kind of representation could be used in VOTable as well[6]; both referrer and target objects may be stored in TABLEs. This context requires that the mapping of the target objects includes an explicit mapping for their identifier. And the reference element must have a structure replicating the structure of that identifier.
This is the case where both referrer and target are indirectly represented by GROUP. Either or both may also be directly represented by a GROUP, which may offer different ways to represent the reference. An even more perverse possibility is that the target object is serialized in a different database or document altogether. For example one may wish to refer to a coordinate system formally defined in a document somewhere else, possibly officially registered in an IVOA registry. This object may be identifiable through some IVO identifier, and maybe our serialization can make use of this and not have to contain its own serialization of this reference data.
All in all we need to deal with the cases described in the following sections.

1. Both referrer and target directly represented by a GROUP.
2. Referrer directly in GROUP, refers to object indirectly represented by GROUP annotating TABLE.
   Referrer GROUP must be able to identify a row in the TABLE. The TABLE in which to look for the object MUST be annotated by a GROUP representing a type compatible with the type of the Reference. For it to be possible to identify the TR in the TABLE that stores the actual target object, the GROUP annotating the TABLE must define an ID. This may be through a single FIELDref, or possibly through a GROUP with multiple FIELDref-s in case the ID is structured. The referring GROUP MUST map the reference to a GROUP have a copy of this ID with values set. For each FIELDref there must be a PARAM referring (through its @ref) to the same FIELD as the ID’s FIELDref and with @value set to the value in the TR. Since there is only a single referrer object here, it SHOULD be possible to indicate in which TABLE the target object resides, indeed which GROUP represents it. This again should be done using a “GROUPref”.
3. Object in TR refers to object in GROUP.
   The object in the TR can in general only use TDs to refer to an object elsewhere. Hence an object in a GROUP will have to define an ID with its PARAMs. If there is a single referenced object, the
4. Object in TR refers to object in TR
5. Object in GROUP to external object not in document.

[6] See old note by Louys and Bonnarel, http://www.ivoa.net/Documents/Notes/DMVOTSer/ModelVOTSer-20040522.pdf

6. Object in TR to external object not in document.

In the titles of the sections below the target representation is located on the right of an →

7.7.1 GROUP[@utype ⇒ Reference & @ref] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩ & ¬ FIELDref] → GROUP[@utype ⇒ ⟨ObjectType⟩ & @id & ¬ FIELDref]

The title indicates that a GROUP inside of a GROUP representing a structured type directly (¬FIELDref) represents a reference to another GROUP that also represents an object directly, either as a type or as a role (must be a collection element).
We have to assume that each referenced object has an identifier and each referrer must somehow have a copy of that identifier. As the definition of identifiers is not exactly prescribed, neither can we prescribe the form that the reference will take.

Restrictions

- GROUP identified by @ref MUST represent an ObjectType compatible with the data type of the Reference, i.e. either the same type of some of its subtypes.

Example

```
<GROUP utype="SAMPL:source/PhotometryFilter" ID="_2massJ">
  <PARAM name="name" datatype="char" utype="SAMPL:source/PhotometryFilter.name"
value="2mass:J"/>
</GROUP>
...
  <GROUP utype="SAMPL:source/Source.luminosity;SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="_magJ" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <FIELDref ref="_errJ" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="_2massJ" utype="SAMPL:source/LuminosityMeasurement.filter"/>
  </GROUP>
...
```

7.7.2 GROUP[@utype ⇒ Reference + 'vo-dml:Identifier'] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩ & ¬ FIELDref] → GROUP[@utype ⇒ ⟨ObjectType⟩ & FIELDref & ID] ∊ TABLE

The title indicates an ObjectType or DataType instance directly represented by a GROUP (¬FIELDref) that contains a GROUP whose @utype is a concatenation of a utype identifying a Reference with the utype identifying the VO-DML ID concept, 'vo-dml:ObjectType.ID' (see 7.6). This indicates the GROUP represents a reference that identifies its target object using a copy of the identifier of that object. This is equivalent to the way foreign keys are used in the relational model to make links between rows in tables.

7.7.3 GROUP[@utype ⇒ Reference + 'vo-dml:Identifier'] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩ & FIELDref] → GROUP[@utype ⇒ ⟨ObjectType⟩ & ¬FIELDref & ID]

TBD

7.7.4 GROUP[@utype ⇒ Reference + 'vo-dml:Identifier'] ∊ GROUP[@utype ⇒ ⟨ObjectType | DataType⟩ & FIELDref] → GROUP[@utype ⇒ ⟨ObjectType⟩ & FIELDref & ID]

TBD

7.8 Collection

A collection is a composition relation between a parent ObjectType and a child, or *part*, ObjectType. For the same reasons as discussed for reference mapping, the fact that objects can be stored in different ways implies many different ways in which the relation may need to be expressed.
In XML serializations of a data model (such as used in VO-URP and the Simulation Data Model) one may choose to have the contained objects serialized *inside* the serialization of the parent. It is possible to do so in VOTable as well using child GROUP-s representing a complete child object embedded in the GROUP representing the parent object. This is natural for this relationship because a collection element is really to be considered as a *part* of the parent object.
This may be used to represent flattening of the parent-child relation. One or more child objects may be stored together with the parent object in the same row in a TABLE. Such a case is actually very common, for example when interpreting tables in typical source catalogues. These generally contain information of a source together with one or more magnitudes. The latter can be seen as elements form a collection of photometry points contained by the sources.
Hence a GROUP inside another GROUP MAY represent a collection. It can do so in different modes that are described in the following subsections.
Restriction

- The parent GROUP must represent an ObjectType that can contain the Collection.

See also the treatment of the Container reference in section 7.8.3.

7.8.1 GROUP [@utype ⇒ Collection+ObjectType] ∊ GROUP[@utype ⇒ ⟨ObjectType⟩]

Contained group represents an object of the indicated ObjectType that is an element of the indicated Collection. The GROUP MUST be contained inside of a GROUP that represents an ObjectType and is a valid Container of the Collection.

In this case the child GROUP represents a single object form the collection, not the collection as a whole. This can occur when annotating what is often called a flattened representation. One or more (must be a fixed number!) of elements of a collection are added to the container object and represented in the same TR in a TABLE.

Restriction

- The @utype MUST be a concatenation of the identifying utypes of both the Collection *and* the actual ObjectType.
- …

Example

```
<TABLE>
<GROUP utype="SAMPL:source/Source">
  <FIELDref ref="_designation" utype="SAMPL:source/Source.name"/>
  <GROUP utype="SAMPL:source/Source.position;bSTC:SkyCoordinate">
    <FIELDref ref="_ra" utype="bSTC:SkyCoordinate.longitude"/>
    <FIELDref ref="_dec" utype="bSTC:SkyCoordinate.latitude"/>
    <GROUP ref="_icrs" utype="bSTC:SkyCoordinate.frame"/>
  </GROUP>
  <GROUP
utype="SAMPL:source/Source.luminosity;SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="_magJ" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <FIELDref ref="_errJ" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="2massJ" utype="SAMPL:source/LuminosityMeasurement.filter"/>
  </GROUP>
  <GROUP
utype="SAMPL:source/Source.luminosity;SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="_magK" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <FIELDref ref="_errK" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="2massK" utype="SAMPL:source/LuminosityMeasurement.filter"/>
  </GROUP>
  <GROUP
utype="SAMPL:source/Source.luminosity;SAMPL:source/LuminosityMeasurement">
    <FIELDref ref="_magH" utype="SAMPL:source/LuminosityMeasurement.value"/>
    <FIELDref ref="_errH" utype="SAMPL:source/LuminosityMeasurement.error"/>
    <GROUP ref="2massH" utype="SAMPL:source/LuminosityMeasurement.filter"/>
  </GROUP>
</GROUP>
<FIELD name="designation" ID="_designation" utype="ivoa_1.0:stdtypes/string">
<DESCRIPTION>source designation formed from sexigesimal coordinates</DESCRIPTION>
</FIELD>
<FIELD name="ra" ID="_ra" unit="deg">
<DESCRIPTION>right ascension (J2000 decimal deg)</DESCRIPTION>
</FIELD>
<FIELD name="dec" ID="_dec"  unit="deg">
<DESCRIPTION>declination (J2000 decimal deg)</DESCRIPTION>
</FIELD>
```

7.8.2 GROUP [@utype ⇒ Collection & @ref] ∊ GROUP[@utype ⇒ ⟨ObjectType⟩]

If a child GROUP identifies a Collection, but *not* also an ObjectType, and it has a @ref identifying a GROUP elsewhere in the document, this should be interpreted as an indication that the referenced GROUP elsewhere in the document represents all or part of the identified Collection. This may be indirectly through a

TABLE, in which case it might represent multiple objects [TBD need Container pointer?], or it may be a single element identified by this "GROUPref".
During a "standard" object-relational mapping procedure, parent objects and the elements in a collection are generally mapped to different tables, with the child table having a pointer (foreign key in relational database) to the parent object. This can be handled by VOTable, which also provides the possibility for the parent object to be stored in a single GROUP (see).
In either case the child GROUP does *not* represent an element, but is a true pointer to a GROUP that annotates the TABLE that stores the collection objects. Actually, it may need to have multiple pointers, in case the collection is divided over multiple TABLEs. Each individual pointer to a GROUP must act as a kind of GROUPref. Since such an element does not exist we propose the following solution.

- The child GROUP representing the collection MUST have as @utype the utype of the Collection, *not* concatenated with another utype.
- The child GROUP MUST contain one PARAM for each TABLE storing (part of) the collection. This PARAM MUST have a @ref identifying the @id of the GROUP wrapping the TABLE.
- That referenced GROUP MUST represent a type compatible with the declared type of the collection.
- IF the parent object is stored in a TR (i.e. the parent GROUP wraps a TABLE), the child table MUST have a Container reference (see 7.8.3) to identify the parent instance.
- TBD how about parent in GROUP?

7.8.3 GROUP[@utype ⇒ Container+ID] ∊ GROUP[@utype ⇒ ⟨ObjectType ⟩]

Container is a special reference to remote object. But can only occur in ObjectType and container cannot be embedded. From the Container utype the possible type of the Container can be inferred.

7.9 Extends, inheritance

Object-relational mapping generally requires one to choose some approach to mapping inheritance hierarchies. One standard approach is to store a class distributed over multiple tables, linked using foreign keys. Each table stores the attributes etc. defined on one class in the hierarchy. To build a complete instance of the subclass requires one to join its contents to the contents of a table storing attributes etc. defined on its base class and so on. In VOTable one *may* encounter a similar mapping. The serialization (and annotation) of such a case requires a mapping of the "extends" relationship between subclass and base class.
An alternative mapping is to have one table per hierarchy, which then will store instances of different types. When using this in OR-mapping often a special column is used that identifies the type stored in a certain row. Note that such a

column is also often added to the table representing the root base class of the inheritance mapping above.
Finally (?) one may have separate tables for each concrete class in the hierarchy, containing columns for all the attributes etc. from its base classes.
All of these different possibilities to store objects form a common inheritance tree furthermore complicates the treatment of polymorphism in references. If a reference is defined whose data type is a type that has subclasses, to find the target instances may require looking through more than one target table.
TBD i.e. it's quite a mess.

7.9.1 GROUP[@utype ⇒ Extends+ID] ∊ GROUP[@utype ⇒ ⟨ObjectType ⟩]

...
TBC

8 Notable absences

The VOTable schema allows for redundancy in meta-data assignment. For example it allows assigning a ucd or utype to FIELDref-s, but also to the FIELD it references. How is one to interpret or use this? Our approach is at least for the use of utype attribute, to try to avoid this redundancy.
The design laid out in the previous subsections focuses utype assignments on the GROUP element and it components. The main reason is that in all but the most simplistic use cases we will not be able to void the use of GROUPs, and that at the same time they provide all functionality (and more) that TABLE and FIELD could provide. Choosing this approach implies client coders do not need to take the possibly conflicting assignments into account, they only need consider GROUPs.
Here we list a few possible assignments that we try to avoid, though they might seem valid.
TBD what more can we remove from section 7? FIELD annotation? All notation directly to PrimitiveType or Enumeration?

8.1 TABLE[@utype ⇒ ObjectType]

There might be some cases where a TABLE could be said to represent a structured type completely, and where the TABLE's @utype attribute could make that identification. However in probably most cases only part of the TABLE will correspond to the type, or multiple types will be stored inside a single row (see for example). In all of these cases one can (and MUST) use one or more GROUP elements contained by the TABLE to make the precise assignment. The extra cost of also doing so for TABLE-s where in principle the assignment could be made directly is minor. Client code only has to deal with GROUP-s, not also inspect the TABLE.

8.2 FIELD[@utype ⇒ Attribute] ∈ TABLE [@utype ⇒ ObjectType]

This assignment would only make sense inside of bigger context, which would have to be the TABLE here. But as we propose not to use TABLE to represent a DM element directly, consequently FIELD cannot be an attribute. We use FIELDref for that.

8.3 PARAM [@utype ⇒ Attribute] ∈ TABLE [@utype ⇒ ObjectType]

For the same reason that makes us avoid the assignment of Attribute to FIELD, we avoid assigning an attribute to a PARAM *that is contained in a TABLE*. The context (TABLE) is not used to indicate the type containing the Attribute. For this element a PARAMref inside a GROUP is to be used.

9 Utype: identifier or path expression?

Opposition has been raised against the explicit mapping of all structured types to GROUP elements with utype either the type, or the role(+type) the type plays in the definition of a parent GROUP. Some of the objections are that

1. It does not conform to existing usage.
2. It is complex and harder to parse using simple string matching tools such as SED
3. It cannot be applied to TAP_SCHEMA or in FITS files
4. ...

Another argument against the proposal could be that in particular attributes with a structured data type might not need such a hierarchy, which, depending on data model, might be deep indeed (see some STC examples). Instead if utypes had a formally defined path like structure, then from the structure, the hierarchical grouping of attributes belonging together might be inferred.

Existing utypes lists such as STC and Characterization seem to have taken this approach as some examples from the usages document show:

- stc:AstroCoords.Position2D.Value2.C1
- stc:AstroCoordSystem.SpaceFrame.CoordRefFrame.Equinox
- Char.SpatialAxis.Coverage.Location.Value

Some examples used in Aladin even go further towards an XPath-like syntax

- stc:AstroCoordArea/Region/reg:Sector/Center[1]
- ivoa:Characterization[ucd=pos]/Coverage/Bounds/min

The usages document[7] goes into more detail.

The proposal to use paths has not been developed (yet) as far as the proposal laid out in the sections above, in fact there is not a single proposal. In the current section, and as agreed in UTYPEs tiger team telecom 2013-03-26, we discuss

[7] https://volute.googlecode.com/svn/trunk/projects/utypes/current-usage/utypes-usage.html

each in more detail in section 9.2 below. First we present some thoughts on how one might derive paths form a VO-DML data model.

9.1 Deriving paths from VO-DML

The ant task run_vo-dml2paths-xml in the vo-dml build script[8] allows one to generate an XML file containing all possible paths one might wish to derive from a specified VO-DML data model. The XML file consists of elements of the following type:

```
<path expression="phot:PogsonZeroPoint.flux.unit">
  <path-element type="Model" utype="phot">
    <path-element type="ObjectType" utype="phot:PogsonZeroPoint">
      <path-element type="Attribute" utype="phot:ZeroPoint.flux">
        <path-element type="Attribute" utype="phot:PhysicalQuantity.unit" />
      </path-element>
    </path-element>
  </path-element>
</path>
```

A <path> element defines a path. It has a @expression attribute that gives the path expression. It contains nested <path- element>s that explicitly define the meaning of the path expressions. Each element in the path identifies the VO-DML element it represents. Its @type attribute indicates the VO-DML meta-model element type; the @utype element identifies precisely *which* element it is. The path expressions are generated to comply with the following grammar, which is a slightly altered version of the UTYPEs grammar originally proposed in SimDM[9] section 4.1. In particular the extends-utype is added and the separator between model and suffix is a ":" only:

```
utype := [model-utype | package-utype | class-utype |
            attribute-utype | collection-utype |
            reference-utype | container-utype | extends-utype |
            identifier-utype ]
model-utype := <model-utype>
package-utype := model-utype ":" package-hierarchy
package-hierarchy := <package-name> "/" [<package-name> "/"]*
class-utype := package-utype "" <class-name>
attribute-utype := class-utype "." attribute
attribute := [primitive-attr | struct-attr]
primitive-attr := <attribute-name>
struct-attr := <attribute-name> "." attribute
collection-utype := class-utype "." <collection-name>
reference-utype := class-utype "." <reference-name>
container-utype := class-utype "." "CONTAINER"
extends-utype := class-utype "." "EXTENDS"
identifier-utype := class-utype "." "ID"
```

This grammar is also used to define the structure of utypes generated by the XSLT pipeline for a VO-DML model. Note that class-utype is used for both ObjectTypes and DataTypes.

[8] https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/build.xml
[9] http://www.ivoa.net/Documents/SimDM/20120503/REC-SimulationDataModel-1.00-20120503.pdf

In boldface are indicated those aspects of the grammar that support the path like expressions one would need if one does not wish to use a GROUP hierarchy. When an attribute is a structured-attribute, i.e. when its datatype is a DataType, the expansion can continue into the datatype and its attributes can be appended. Of course, this is only a grammar. We need some more rules. Those we'll illustrate based on the Toy model in Figure 8

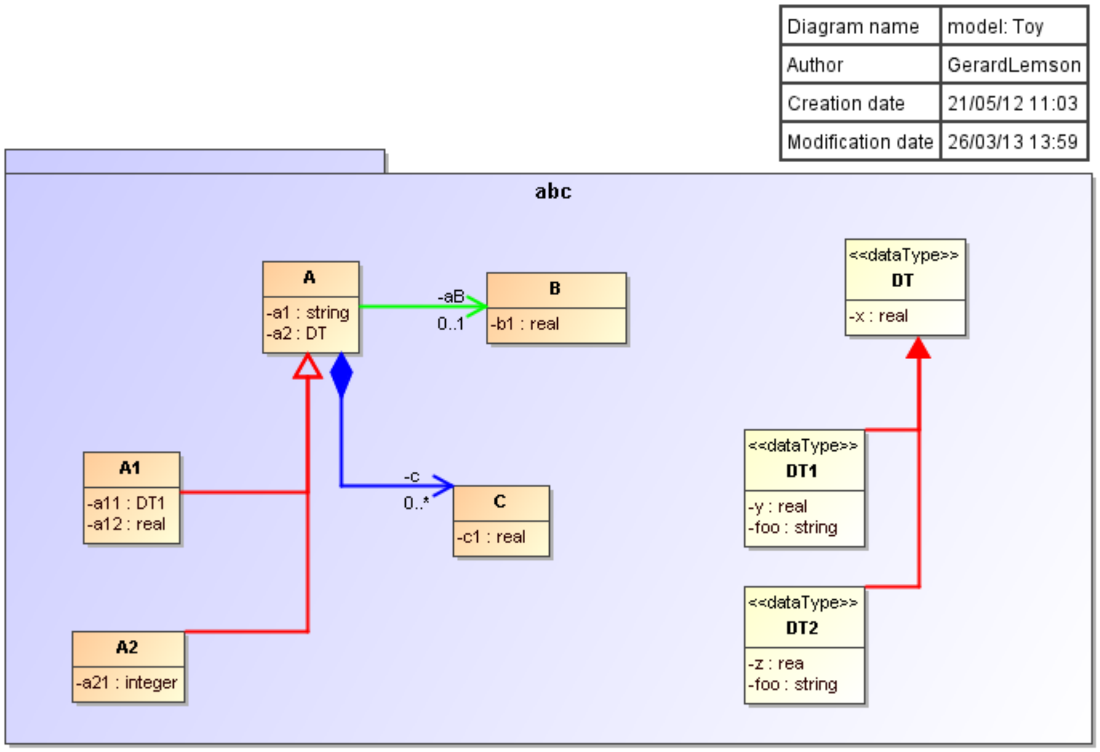


Figure 8 Toy model used to illustrate path generation.

9.1.1 Utype identifiers in VO-DML

Every (referencable) element in a VO-DML document has a utype identifier. When generating a value for this we start with the name of the element (or CONTAINER or EXTENDS) and walk up the hierarchy of elements, adding the name plus separator of the parent element recursively until the <model> element is reached.

Hence an imaginary configuration like the following,

```
<vo-dml:model><name>TOY</name>
...
  <package><name>abc</name>
  ...
    <objectType><name>C</name>
    ...
      <attribute><name>c1</name>
    ...
```

would lead to a utype TOY:abc/C.c1 for the attribute. Note, that these utype identifiers are also path expressions, but constrained. Only defined elements get utypes, and paths only follow explicit hierarchies. For the Toy model, the following table shows all utype identifiers generated this way:

TOY	TOY:abc/B

`TOY:abc/`
`TOY:abc/A`
`TOY:abc/A.a1`
`TOY:abc/A.a2`
`TOY:abc/A.aB`
`TOY:abc/A.c`
`TOY:abc/A1`
`TOY:abc/A1.EXTENDS`
`TOY:abc/A1.a11`
`TOY:abc/A1.a12`
`TOY:abc/A2`
`TOY:abc/A2.EXTENDS`
`TOY:abc/A2.a21`
	`TOY:abc/B.b1`
	`TOY:abc/C`
	`TOY:abc/C.CONTAINER`
	`TOY:abc/C.c1`
	`TOY:abc/DT`
	`TOY:abc/DT.x`
	`TOY:abc/DT1`
	`TOY:abc/DT1.EXTENDS`
	`TOY:abc/DT1.y`
	`TOY:abc/DT1.foo`
	`TOY:abc/DT2`
	`TOY:abc/DT2.EXTENDS`
	`TOY:abc/DT2.z`
	`TOY:abc/DT2.foo`

Figure 9 Table with utype identifiers generated for the Toy model

These rules are relaxed in the generic paths described next.

9.1.2 Generic path expressions

Here we follow a different set of rules, which one could characterize as walking in the opposite direction from the previous case. We always start at a structured type, either an ObjectType or a DataType. Paths identifying these are generated the same as in the previous section, i.e. model, zero or more packages and the type name itself.
Paths for attributes and so on are generated downwards. Paths are created for *any* role that is "available on"[10] the type. In particular this includes attributes, references etc *inherited* from a base class. Furthermore, if an attribute has a datatype that has subtypes, polymorphism indicates that any of the subtypes may be used to set the value of the attribute. Hence paths are also followed down all subclasses in such a case. In the Toy model this occurs for the attribute a2 defined on a. Its datatype is DT, which has subclasses DT1 and DT2, each with their own attributes. Note that this actually causes a problem for the current mechanism. It is legal for both DT1 and DT2 to define an attribute with the same name, here 'foo'. This will now lead to identical paths, indicated in common colors in the table below.
The following table lists all paths generated for the Toy model. Paths formatted in boldface correspond to an exact/pure utype identifier. All others are derived path expressions.

`TOY`
`TOY:abc/`
`TOY:abc/A`
`TOY:abc/A.a1`
`TOY:abc/A.a2`
`TOY:abc/A.a2.x`
`TOY:abc/A.a2.x`
`TOY:abc/A.a2.z`
`TOY:abc/A.a2.foo`
`TOY:abc/A.a2.x`
	`TOY:abc/A2`
	`TOY:abc/A2.EXTENDS`
	`TOY:abc/A2.a1`
	`TOY:abc/A2.a2`
	`TOY:abc/A2.a2.x`
	`TOY:abc/A2.a2.x`
	`TOY:abc/A2.a2.z`
	`TOY:abc/A2.a2.foo`
	`TOY:abc/A2.a2.x`
	`TOY:abc/A2.a2.y`

[10] As defined in the list of shorthand phrases in the start of section 7.

TOY:abc/A.a2.y	TOY:abc/A2.a2.foo
TOY:abc/A.a2.foo	**TOY:abc/A2.a21**
TOY:abc/A.aB	TOY:abc/A2.aB
TOY:abc/A.c	TOY:abc/A2.c
TOY:abc/A1	**TOY:abc/B**
TOY:abc/A1.EXTENDS	**TOY:abc/B.b1**
TOY:abc/A1.a1	**TOY:abc/C**
TOY:abc/A1.a2	**TOY:abc/C.CONTAINER**
TOY:abc/A1.a2.x	**TOY:abc/C.c1**
TOY:abc/A1.a2.x	**TOY:abc/DT**
TOY:abc/A1.a2.z	**TOY:abc/DT.x**
TOY:abc/A1.a2.foo	**TOY:abc/DT1**
TOY:abc/A1.a2.x	**TOY:abc/DT1.EXTENDS**
TOY:abc/A1.a2.y	TOY:abc/DT1.x
TOY:abc/A1.a2.foo	**TOY:abc/DT1.y**
TOY:abc/A1.a11	**TOY:abc/DT1.foo**
TOY:abc/A1.a11.x	**TOY:abc/DT2**
TOY:abc/A1.a11.y	**TOY:abc/DT2.EXTENDS**
TOY:abc/A1.a11.foo	TOY:abc/DT2.x
TOY:abc/A1.a12	**TOY:abc/DT2.z**
TOY:abc/A1.aB	**TOY:abc/DT2.foo**
TOY:abc/A1.c	

Figure 10 Table with path expressions generated for the Toy model

9.2 3 solutions

The discussion between the solution proposed here, sometimes called the "pure utypes" approach, and one using more traditional path-like expressions has been hampered by the fact that no complete solution for the latter has been proposed. Here we make an attempt at remedying this.
First we observe that there is no single proposal for how to approach an alternative, in the telecom at 2013-03-26 at least three different ones were identified. We list these here.

9.2.1 Paths iso GROUP-s for structured attributes.

The first proposal is mainly supported by Markus. ObjectType-s will still be represented by GROUPs, and so will collections (references will have to be dealt with yet). The main change is that one would not use child GROUPs for structured attributes. Instead its aim is to flatten these out, and annotate the resulting set of atomic "leaf" components (FIELDrefs, PARAMs and PARAMrefs) with the paths as defined in 9.1.2.
These paths are supposed to be parsable, i.e. *not* opaque strings. The reason for this is that with opaque strings one needs a pre-existing list form which one can infer the meaning. Unless this list can be kept short, it is better to be able to parse them.
The advantage of this approach is that one can use these annotations in cases where GROUP-like context is not available. The most extreme example of this is a part of the RegTAP solution where a part of the registry data model is flattened into a list of keyword-value pairs. If the keywords are to be represented by utypes,

the pure utypes may not give sufficient information. [TBD Investigate possible relevance of <<skosconcept>> in VO-DML for this problem.]

9.2.2 Paths (formal and custom) as aliases for "pure utypes"

Second there is a proposal by Jesus. He wishes to allow data model designers to add a list of explicitly defined custom utypes to the model and define their meaning in terms of paths in the model. I.e. the meaning does not come from parsing the string, but from an explicit definition. These custom utypes are to be interpreted as aliases utypes, representing pure utypes when those are used inside a particular path context. The GROUP based approach is still assumed [TBD], but at leaf nodes, on FIELDrefs etc, one is allowed to use an alias utype rather than a pure one. Supposedly one wants to insist on consistency of annotation. I.e. the path represented by the alias utype must be consisted with the GROUP hierarchy it is used in. [TBD Jesus hinted that maybe these alias utypes could be used on their own as well, i.e. outside of containing GROUPs. If so more work is to be done

The main advantage of this approach is the possibility for backwards compatibility. If we still want to support existing utype lists, the only thing added would be an interpretation of these as aliases of properly defined path expressions [TBD in a VO-DML model that may still have to be written]. Another advantage of this might be that one could use these aliases also in FITS keywords or TAP_SCHEMA.Columns.utype annotations.

9.2.3 Paths as complementary annotation on FIELDs and standalone PARAMs only

The third approach was proposed by Mireille, and apparently earlier also by Omar. It allows path-like expressions for utype attributes, but only on FIELDs and PARAMs outside GROUPs. I.e. precisely on those atomic elements that in the current proposal are *not* used. The proposal allows both the GROUP-based annotation with pure utypes, as well as the more traditional utypes. But the latter are *only* used on these atomic fields. They are *not* used in the GROUP based annotation.

These path-like utypes would be opaque as in case 2 above. I.e. a list of allowed utype-s must be given along with the data model and defined as path expressions [TBD is this correct, or would/could they be parsable?]

Main issues are: undesirability of two separate approaches and decision on which of two, or both, are mandatory.

Some possible generalizations:

- Cases 2 and 3: It need not be up to the data model designers to define the set of aliases. Also DAL protocols might define their own list.
- ..

9.2.4 Paths as "pure utypes" of Views

A fourth proposal that falls in this same area has been proposed independently in various forms by GL, also Omar, and Pierre. It hinges on a (possible) capability to define derived data models form a main one. The derived model would

generally be simpler in structure than the original to which it is still explicitly linked. Annotations could now follow the "pure utypes" approach, but with target these simpler models. Those will have their own annotation, but as utypes are opaque, existing, "legacy" utype values could be used to identify some of the elements. In fact, it may be possible to interpret some of these lists as providing a simplified model. Also the STC in VOTable proposal may be seen in these terms.
Main issue here is to define a language for linking derived to original model. GL has pointed to OQL, but that goes too far at the current stage. In fact one could argue that a VOTable annotated using the pure utype GROUP approach, could be seen as such a model. [TBD]
Advantage is that protocols could define these derived models formally and link them to the original. Can we interpret the original SSA+SpectrumDM like such a combination?

9.3 Discussion

NB Points below are mainly directed at proposal 9.2.1 above.
Possible arguments *against* the use of paths as defined in 9.1.2 for annotating VOTables might be:

1. Structure of data model only implicitly expressed, by path expressions based on a non-existing serialization. *But see counter argument in point 4 below.*
2. *Harder* to code against: a tool that wishes to use the data model structure to analyse/interpret a VOTable is most naturally coded in a language that easily expresses the data model, say Java or some other OO language. These naturally map attributes with structured types to fields with class representing the type. Such a class can easily be coded to expect a GROUP with structural components based on the definition of the type alone, NOT of some containing type. So a model importing STC, could also use an STC interpretation library as part of its own, does not need to write new code. Also, the interpreter only needs to see a GROUP, interpret its utype to decide whom to pass it on to. Does not first need to inspect all attribute, group some together to then pass it off to a component that does the same work anyway.
3. XPath expressions for inferring DM instances can be easily written against a hierarchy as well. Even easier, as the natural XML structure is more easily queried. Simply search for appropriate GROUPs.
   NOTE here it would have been better had UTYPE-s been element-s rather than attributes. These could have been given more structure that now must be encoded in @uype syntax.
4. Some Utype-s will only be defined implicitly. The data model itself will NOT contain an expression indicating explicitly the “attribute-of-attribute”. Hence it cannot be looked up. It could be derived, but that is unnatural and breaks OO paradigm. Hence path UTYPEs MUST be parsed.
   Counter: can derive all valid paths from VO-DML as shown in various use cases. DM designers might choose a subset of these that are valid.

5. GROUPs MUST be supported anyway for collections, 0..* attributes, references. Also, for custom serializations such as those resulting from queries, the utype-s of columns in TABLE may not be unique, hence a context (through GROUPs) is required to combine those columns that belong together. So coders must be able to deal with GROUP hierarchies anyway. Even in the most natural case: mapping a catalogue such as SDSS's PhotoObjAll, to the sample Source model.
6. Fact that FITS does not support GROUP-ing should not hold us back. FITS files SHOULD be wrapped by VOTable anyway in any VO context. [TBD DataLink?]. Its own metadata fields SHOULD NOT be used for this kind of annotation. We cannot be kept back in the expressiveness of our metadata by ancient formats as long as we already support wrapping FITS files anyway.
   And see point 5. GROUPs should be supported for other parts of the mapping, so a solution must be found anyway. That same solution can be applied to structured attributes.
7. Fact that TAP_SCHEMA does not support GROUPs should not be a problem either. Most obvious solution is to extend the TAP_SCHEMA. Alternative is to provide TAP_SCHEMA with a VOTable annotation similar to FITS. Each TAP service SHOULD be able to give a VOTable with TABLE definitions for each table in the database. There everything is available. Alternatively this could be added to VODataService or whatever other metadata format is desired. Note that querying a hierarchical data structure in tables is not supported naturally as it requires recursive queries. Since common table expressions are not part of ADQL this might be difficult. However see Appendix A for a possible schema extension that makes it still quite easy to find all groups in a table.
   And see point 5. GROUPs should be supported for other parts of the mapping, so a solution must be found anyway. That same solution can be applied to structured attributes.
8. Formally there should not be existing usage, as there is no spec dealing with utypes. That spec is what we're trying to define and existing usage may indeed have to be re-implemented. Hopefully we can find a more friendly solution. In particular the fact that in the solution proposed here utype-s do not have a prescribed syntax (apart from prefix-> model and possible concatenation), may allow some existing utypes to be interpreted as identifiers into a model in the prescribed way. Particularly since the models do not have a VO-DML representation that might be possible. One way might be to provide the utype-s list in these models a translation in terms of a GROUP structure. Note that this is only necessary for legacy implementations that should formally not have been there yet. SO another way to deal with existing utype-s lists is that utypes have to be interpreted in the context of the model that must be provided. IF the model (maybe extra meta-data field) does not have a VO-DML representation, in any case all spec is off. Special code only needs to be written against those models explicitly. We do not have to take them into account in the spec. A migration path exists to re-express all models in terms of VO-DML after which VOTables using these new versions of the model can also start using the formal version of the utype mapping spec.

9. ...

TBD

10 TBDs, Notes, issues, questions, comments

Serializing data models after a transformation contains many possible complications. Some have been treated above, others still need discussions. Here are a few:

1. Same VOTable component may be annotated in multiple ways. For example a “name” column (FIELD) can be given a utype of an Attribute, as well as of the ID of its ObjectType. (See example TBD). How shall we do that? Using multiple FIELDref-s, or using a concatenation of utype-s? FIRM proposal is to use multiple FIELDrefs.
2. Single attribute could be represented in multiple ways in the same serialization, say position in multiple coordinate systems, or in degrees *and* HMS. Should we allow that? How? Could we collect all distinct instances of the same role in a special GROUP that has task of GROUPing these representations?Or should we simply allow multiple instances with same attribute utype? But see case 8 below for a possible issue with that.
3. Some serialization (to VOTable) Map [x y z] to an array?
4. @unit: Some data models support the storage of quantities and have an explicit ‘unit’ attribute. The sample data model has this. Should there be special treatment of this? I.e. should we be able to say that a certain element is stored in a VOTable attribute?
5. Non-standard models. Can be easily accommodated, as long as there is a VO-DML representation of them. Only issue might be loss of control over supposed standard set of model prefixes. But if we release that, i.e let prefixes be free, then no problem.
6. Typical OR-mapping pattern is to store multiple subclasses in same table, but have a special column to indicate which type is stored. This is not yet treated here.
   One possibility is to allow the inheritance hierarchy to be mapped to a GROUP hierarchy. I.e the root GROUP represents the base class, sub GROUPs may be indicated with the role utype="vo-dml:Type.subClass+<utype of subclass>". This subclass-GROUP would define where its attributes etc are mapped. And it can have its own subclasses etc. This is *only* allowed on GROUPs wrapping a TABLE.
   It would be necessary somehow to indicate which column represents the identifier of the class that is stored in a row, and what value is used to do so. A possible way would be:
   FIELDref on root group with utype="vo-dml:Type.nameAlias" indicating the FIELD.
   On the GROUP representing a concrete type, a PARAM with utype="vo-dml:Type.nameAlias" and value the alias used for this type.
   Note, the root GROUP might also define a PARAM with utype="vo-dml:Type.nameAlias" and a value.
   This is TBD.

7. We may want to remove the option to use utype-s on FIELD and PARAMs-outside-GROUPs. I.e. *not* allow the patterns indicated in 7.2. Indicated this there using a DEPRECATED ...
8. Need to discuss explicitly how to deal with attributes (and references?) of cardinality > 1. Simplest is to simply allow multiple occurrences of elements with the same utype. Similar to how we support collection elements. Problem might be that this could interfere with case 2 above. It therefore might be important to
9. TBD Need a data model for definint Units that can be reused. In the data model units should be ObjectType-s ala the definition in PhotDM-alt. a standard document should be created listing all valid units. These should have a URI so they can be referenced explicitly, ala the way SKOS does this.
   Note, maybe also the PhysicalQuantity-s could be defined there.
10. PROPOSAL *inside* a VO-DML/XML document, the utypes *do not* have a prefix. Utyperef-s *do* have a prefix to indicate the model. But when looking up a utype inside a model the unprefixed version is to be used. This allows custom utypes to be handled more easily and allows variation of prefixes. Wherever a utype is use to reference an element, the prefix is mandatory. So also in VOTable @utype attributes for example.
11. PROPOSAL: When a utype identifies a role whose data type is defined in an external, imported data model, the type SHOULD be specified (using concatenation with '+' in current proposal). This increases interoperability of the VOTable, as it allows tools only familiar with the external model to be used in the interpretation of the annotation. Note, this is NOT necessarily advisable, but it allows for example a simple summary of a VOTable to indicate say all STC positions or regions that occur and could be extracted/plotted by a generic tool such as Aladin or TOPCAT that do not (need to) know what to do with the main model. And I think this is a proper use case.
12. (From Omar) TODO check that in the spec we do *not* insist that container objects are available for each contained object. Hence CONTAINER reference should not be obligatory.
13. ..

Appendix A. VO-DML/XML version of sample model TBD keep updating, or link to online version.

Here is the Sample model in the XML representation of VO-DML. Note that this model imports two other models, the IVOA Profile and a simple STC model. Those are not [TBD yet?] replicated here.
Note, this representation is likely out of date soon after updating it in this text. Hence it is better to to examine the model in
https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/sample/Sample.vo-dml.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vo-dml:model xmlns:vo-dml="http://volute.googlecode.com/dm/vo-dml/v0.9">
  <identifier id="eee_1045467100313_135436_1">
    <utype>SAMPL</utype>
  </identifier>
  <name>Sample</name>
  <description>This is a sample data model. It contains the IVOA UML Profile and imports
the IVOA_Profile data model with primitive types.
It has some sample relationships etc to be used in documentation etc.</description>
  <title>Sample VO-DML data model.</title>
  <version>0.x</version>
  <lastModified>2013-02-08T17:03:40</lastModified>
  <import>
    <identifier id="_12_1_1dfa04c4_1360333713314_831168_335">
      <utype>bSTC</utype>
    </identifier>
    <name>basicSTC</name>
    <url>https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/models/simple-
STC/bSTC.vo-dml.xml</url>
    <documentationURL>https://volute.googlecode.com/svn/trunk/projects/dm/vo-
dml/models/simple-STC/bSTC.html</documentationURL>
  </import>
  <import>
    <identifier id="_12_1_1dfa04c4_1352203901356_512837_228">
      <utype>ivoa_1.0</utype>
    </identifier>
    <name>IVOA_Profile</name>
    <url>http://volute.googlecode.com/svn/trunk/projects/dm/vo-
dml/models/profile/IVOA_Profile.vo-dml.xml</url>
    <documentationURL>http://volute.googlecode.com/svn/trunk/projects/dm/vo-
dml/models/profile/IVOA_Profile.html</documentationURL>
  </import>

  <package>
    <identifier id="_12_1_1dfa04c4_1354861906481_784028_160">
      <utype>SAMPL:source/</utype>
    </identifier>
    <name>source</name>
    <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
    <objectType>
      <identifier id="_12_1_1dfa04c4_1354024794825_962426_779">
        <utype>SAMPL:source/LuminosityMeasurement</utype>
      </identifier>
      <name>LuminosityMeasurement</name>
      <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      <container>
        <identifier id="_12_1_1dfa04c4_1354024794825_962426_779_CONTAINER">
          <utype>SAMPL:source/LuminosityMeasurement.CONTAINER</utype>
        </identifier>
        <collectionref idref="_12_1_1dfa04c4_1354024798700_361746_809">
          <utyperef>SAMPL:source/Source.luminosity</utyperef>
        </collectionref>
      </container>
      <attribute>
        <identifier id="_12_1_1dfa04c4_1354024679187_898957_712">
          <utype>SAMPL:source/LuminosityMeasurement.value</utype>
        </identifier>
        <name>value</name>
        <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
        <datatype idref="_12_1_1dfa04c4_1352203901371_896288_236">
          <modelUtypeRef>ivoa_1.0</modelUtypeRef>
          <utyperef>ivoa_1.0:stdtypes/real</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024689314_287467_716">
        <utype>SAMPL:source/LuminosityMeasurement.error</utype>
      </identifier>
      <name>error</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901371_896288_236">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/real</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024901425_195936_860">
        <utype>SAMPL:source/LuminosityMeasurement.description</utype>
      </identifier>
      <name>description</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901364_557823_229">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/string</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1355248682629_78694_413">
        <utype>SAMPL:source/LuminosityMeasurement.type</utype>
      </identifier>
      <name>type</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1355248550530_805055_254">
        <utyperef>SAMPL:source/LuminosityType</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1360336457973_924434_666">
        <utype>SAMPL:source/LuminosityMeasurement.unnamed1</utype>
      </identifier>
      <name>unnamed1</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1355248550530_805055_254">
        <utyperef>SAMPL:source/LuminosityType</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
    <reference>
      <identifier id="_12_1_1dfa04c4_1354024794871_921971_802">
        <utype>SAMPL:source/LuminosityMeasurement.filter</utype>
      </identifier>
      <name>filter</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1354024631268_183610_685">
        <utyperef>SAMPL:source/PhotometryFilter</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </reference>
  </objectType>

  <objectType>
    <identifier id="_12_1_1dfa04c4_1354024243598_413443_421">
      <utype>SAMPL:source/Source</utype>
    </identifier>
    <name>Source</name>
    <description>
        TODO : Missing description : please, update your UML model asap.
      </description>_12_1_1dfa04c4_1354024272942_181358_515
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024272942_181358_515">
        <utype>SAMPL:source/Source.name</utype>
      </identifier>
      <name>name</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901364_557823_229">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/string</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024284245_914170_519">
        <utype>SAMPL:source/Source.description</utype>
      </identifier>
      <name>description</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901364_557823_229">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/string</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024509379_61210_633">
        <utype>SAMPL:source/Source.position</utype>
      </identifier>
      <name>position</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1354024389551_634626_529">
        <modelUtypeRef>bSTC</modelUtypeRef>
        <utyperef>bSTC:SkyCoordinate</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354971231645_872982_332">
        <utype>SAMPL:source/Source.classification</utype>
      </identifier>
      <name>classification</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1354971184783_199418_302">
        <utyperef>SAMPL:source/SourceClassification</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <collection>
      <identifier id="_12_1_1dfa04c4_1354024798700_361746_809">
        <utype>SAMPL:source/Source.luminosity</utype>
      </identifier>
      <name>luminosity</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1354024794825_962426_779">
        <utyperef>SAMPL:source/LuminosityMeasurement</utyperef>
      </datatype>
      <multiplicity>0..*</multiplicity>
    </collection>
  </objectType>

  <objectType>
    <identifier id="_12_1_1dfa04c4_1354024631268_183610_685">
      <utype>SAMPL:source/PhotometryFilter</utype>
    </identifier>
    <name>PhotometryFilter</name>
    <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024640339_537847_704">
        <utype>SAMPL:source/PhotometryFilter.name</utype>
      </identifier>
      <name>name</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901364_557823_229">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/string</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024652194_944567_708">
        <utype>SAMPL:source/PhotometryFilter.location</utype>
      </identifier>
      <name>location</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1354024536493_847273_638">
        <utyperef>SAMPL:source/Quantity</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1359623493483_897112_274">
        <utype>SAMPL:source/PhotometryFilter.referenceURL</utype>
      </identifier>
      <name>referenceURL</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901371_161532_239">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/anyURI</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
  </objectType>

  <dataType>
    <identifier id="_12_1_1dfa04c4_1354024536493_847273_638">
      <utype>SAMPL:source/Quantity</utype>
    </identifier>
    <name>Quantity</name>
    <description>
```

```
        TODO : Missing description : please, update your UML model asap.
      </description>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024546380_605494_656">
        <utype>SAMPL:source/Quantity. value</utype>
      </identifier>
      <name> value</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901371_896288_236">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/real</utyperef>
      </datatype>
      <multiplicity>1</multiplicity>
    </attribute>
    <attribute>
      <identifier id="_12_1_1dfa04c4_1354024559196_962262_660">
        <utype>SAMPL:source/Quantity.unit</utype>
      </identifier>
      <name>unit</name>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
      <datatype idref="_12_1_1dfa04c4_1352203901364_557823_229">
        <modelUtypeRef>ivoa_1.0</modelUtypeRef>
        <utyperef>ivoa_1.0:stdtypes/string</utyperef>
      </datatype>
      <multiplicity>0..1</multiplicity>
    </attribute>
  </dataType>

  <enumeration>
    <identifier id="_12_1_1dfa04c4_1354971184783_199418_302">
      <utype>SAMPL:source/SourceClassification</utype>
    </identifier>
    <name>SourceClassification</name>
    <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
    <literal>
      <identifier id="_12_1_1dfa04c4_1354971208510_856613_324">
        <utype>SAMPL:source/SourceClassification.star</utype>
      </identifier>
      <value>star</value>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
    </literal>
    <literal>
      <identifier id="_12_1_1dfa04c4_1354971211833_964999_326">
        <utype>SAMPL:source/SourceClassification.galaxy</utype>
      </identifier>
      <value>galaxy</value>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
    </literal>
    <literal>
      <identifier id="_12_1_1dfa04c4_1354971215359_844385_328">
        <utype>SAMPL:source/SourceClassification.AGN</utype>
      </identifier>
      <value>AGN</value>
      <description>
        TODO : Missing description : please, update your UML model asap.
      </description>
    </literal>
    <literal>
      <identifier id="_12_1_1dfa04c4_1354971218541_706992_330">
```

```
          <utype>SAMPL:source/SourceClassification.planet</utype>
        </identifier>
        <value>planet</value>
        <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      </literal>
      <literal>
        <identifier id="_12_1_1dfa04c4_1354971250162_942097_336">
          <utype>SAMPL:source/SourceClassification.unknown</utype>
        </identifier>
        <value>unknown</value>
        <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      </literal>
    </enumeration>

    <enumeration>
      <identifier id="_12_1_1dfa04c4_1355248550530_805055_254">
        <utype>SAMPL:source/LuminosityType</utype>
      </identifier>
      <name>LuminosityType</name>
      <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      <literal>
        <identifier id="_12_1_1dfa04c4_1355248729653_893543_417">
          <utype>SAMPL:source/LuminosityType.magnitude</utype>
        </identifier>
        <value>magnitude</value>
        <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      </literal>
      <literal>
        <identifier id="_12_1_1dfa04c4_1355248735260_901139_419">
          <utype>SAMPL:source/LuminosityType.flux</utype>
        </identifier>
        <value>flux</value>
        <description>
          TODO : Missing description : please, update your UML model asap.
        </description>
      </literal>
    </enumeration>

  </package>

</vo-dml:model>
```

Appendix B. Mapping "grammar'

[GL Please ignore this for now]
The following grammar defines the structure of mapping expressions used in some of the titles in section 7

```
vot-mapping := element-mapping | relation-mapping
element-mapping := vot-element [ '[' '@utype' '⇒' utype-value [constraints] ']' [context]
]
relation-mapping := element-mapping '→' element-mapping
vot-element := annotated-vot-elemtn | basic-vot-element
basic-vot-element := 'GROUP' | 'FIELDref' | 'PARAM' |'TABLE' | 'FIELD' | '@ref' |'@utype'
utype-value := vo-dml-concept | vo-dml-role '+' vo-dml-type
vo-dml-concept := vo-dml-type | vo-dml-role | vo-dml-types | vo-dml-type-rep | 'Model'
vo-dml-type := 'PrimitiveType' | 'Enumeration' | 'DataType' |' ObjectType' | 'ID'
vo-dml-role := 'Attribute' | 'Reference' | 'Collection' | 'Container' | 'Extends'
vo-dml-type := '(' vo-dml-type ['|' vo-dml-type]* ')'
```

```
vo-dml-type-rep := '⟨' vo-dml-type ['|' vo-dml-type]* '⟩'
constraints := and-or constraint [constraints]
and-or := '&' | '|'
constraint := vot-element | '¬' vot-element
context := '∈' element-mapping [context]*
```

Appendix C. GROUP in TAP_SCHEMA

Following is a possible extension to TAP_SCHEMA that would add GROUP-like annotation to a table. To query for a complete GROUP hierarchy for a table requires resolution of the GROUP-s nesting. In absence of common-table –expressions, the parent_table foreign key should allow one easily to collect all groups for a table and build the nesting in code.

```
CREATE TABLE tap_schema.group (
  id varchar(128) not null -- unique id of this group
, parent_table varchar(128) not null -- foreign key to table ultimately owning this GROUP
, parent_group varchar(128)  -- if not null, id of GROUP owning this group
, utype varchar(128) -- if parent_group is not null, the role the group plays in the
parent,
-- possibly concatenated with a utype for the actual type represented by the GROUP.
-- HOWEVER that could be normalized by having a type_utype and a role_utype column
instead
)

CREATE TABLE tap_schema.group_field (
  groupId varchar(128) not null – ID of GROUP conainign this field-ref
, columnId varchar(128) not null -- column associated to the group, if null, this row is
a PARAM
, utype varchar(128) -- the utype of the role the field plays in group.
-- Possibly concatenated with type? Or normalized (see above)
, value varchar(8000) -- if this row represents a PARAM, this is its value
)
```

[NB, this schema implicitly supports stand-alone GROUP-s by not requiring a columnId, but all GROUP-s must be associated to some TABLE. I.e. standalone instances cannot be freely floating. This could happen if parent_table is allowed to be NULL.]