International
Virtual
Observatory
Alliance

Mango Mapping Syntax

Version 1.0

IVOA Note 2020-04-22

Working group
DM
This version
http://www.ivoa.net/documents/cab-msd/20200422
Latest version
http://www.ivoa.net/documents/cab-msd
Previous versions
This is the first public release
Author(s)
François Bonnarel, Gilles Landais, Laurent Michel, Jesus Salgado
Editor(s)
Laurent Michel

Abstract

???? Abstract ????

Status of this document

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

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

Contents

Acknowledgments

???? Or remove the section header ????

Conformance-related definitions

The words "MUST", "SHALL", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in IETF standard RFC2119 (**?**).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

There is no VO standard specifying how to map real data on a model. This shortcoming is a bottleneck for interoperability since there is no way to compare datasets from different origins. The model mapping is currently

done by using GROUPs. This is satisfactory in many cases but with some limitations however:

1. The GROUPs, as they are currently used, do not reflect the complete model structure. The are mostly used to group columns with a loose coupling with the model hierarchy. This prevents the restore complex data structures.

2. The identification of the GROUP elements is based on Utypes which are not clearly connected with the underlying model. Utypes are actually used to represent simple structures. In the current implementations, the client knows the data structures (e.g. coordinate system) and pick values within the group to fill them..

These limitations were unavoidable while we had no common way to represent models. It is indeed difficult to imagine a consistant model mapping mechanism if there is no standard serialization schema for those models. The situation changed with the usage of VODML. VODML proposes a robust way to make models machine-readable. With VODML, we can consider a mapping syntax directly derived from the model serialization. There are 2 approaches that might be complementary.

1. The first option is to hide the model complexity by flattenning its representation. This is the way GROUPs do work. This approach has the advantage of being compact but it has some strong limitations such as the impossibility of having different elements playing the same role (e.g. 2 position columns in the same table)

2. The second option consists in keeping the model structure even if this leads to more complex annotations.

Vodml-instance-vot is based on the second option with a particular concern for the process of archival data annotation. The purpose of the VODML mapping in VOTables is to provide a bridge between the data and the model it refers to. The goal if mapping processing is to make possible to model instances set with values taken out the data tables. In theory, a perfecly faith mapping should be capable of rendering any model feature. In our opinion this ambitious goal led to a mapping syntax difficult to manage. We prefer to discard this round-trip requirement and to keep focused on both client needs and easiness of the annotation process.

Our main use case is the annotation of pre-existing data. This means the annotation process must succeed whatever the way data are arranged even if they do not contain all quantities requested by the model. We consider thta we have to provide clients with both a data hierarchy view as simple as possibel and an accurate description of the used coordinate systems. For most of the usages (display, plot, match, computation), complex data hierarchies

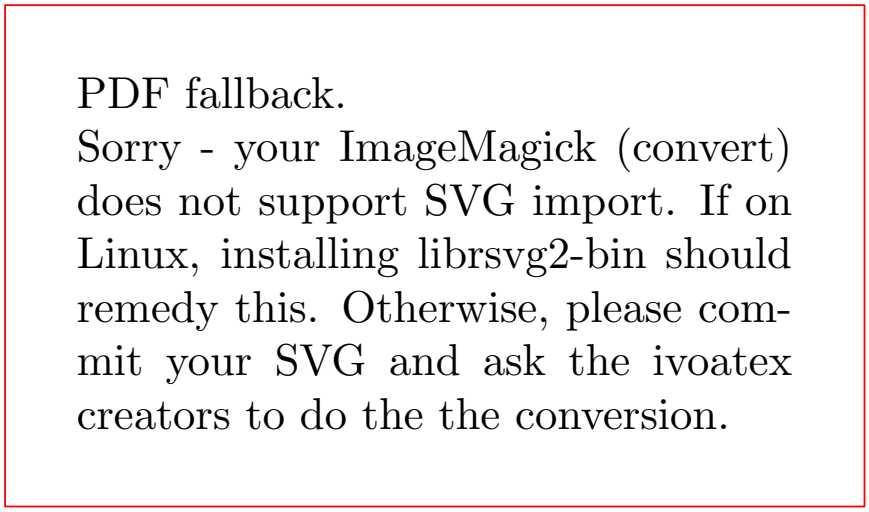

Figure 1: Architecture diagram for this document

can be wrapped in 3 basic types, the simple values, tuples and lists. Vodml-instance-vot has been built upon these 3 types with a few others annotations guiding the parser.

These ideas that led to this proposal were first tested in the framework of the TDIG on VOTABLEs containing time series provided by different missions such as Gaia or ZWICKI. Then the syntax has been refined to be used to validate the Mango model on real data.

1.1 Role within the VO Architecture

Fig. 1 shows the role this document plays within the IVOA architecture (**?**).

???? and so on, LaTeX as you know and love it. ????

2 Use Cases and Requirement

2.1 Use Cases

2.1.1 Client Side

The mapping is self consistant. The role of the mapping is to give the client all information it needs a reconstruct a datastructure similar to the model one. A model-aware client must be able to do this this without implementing any code specific to any particular model. The mapping syntax is independant of the model. The structure of the mapped model is given by the arrangement of the mapping elements, not by the elements them-self .

2.1.2 Server Side

We have identified 3 sorts of servers that could annotate data:

1. Mission data provider: the data annotation can be set once forever for each data product at the design phase.

2. Archival data provider: The data annotation must be done for each archived datat set. The curator has a little control, on the data format and he/she has to do his best to math data with the model(s)

3. TAP data provider: In case of TAP services, the annotation process in of charge of the TAP server that must match the queried data with the model quantities.

The goal of the version is to support 1) and 2) with a special attention to facilitate 2). 3) is still an experimental feature at the time of this specification.

The annotation process can represent a significant extra work for the curator team that must be limited as much as possible. To do so the mapping syntax is designed to facilitate the use of templates. The structure of the mappinfg DOM does not depends on the way mapped data are arranged. The connection between real data and mapping is done through XML elements attributes without changing the nature or the location of the mapping elements .

2.2 Requirements

- Shy Annotations: The data mapping must not affect the operation of the existing clients

- Faithful Annotations: The structure of the annotation must be faithful to any VODML compliant model.

- Different Usage Levels

  - The data mapping must be easy to be ignored by the client
  - The data mapping must allow clients to easily detect the model on which data are mapped
  - The data mapping must allow clients to easily get the metadata (e.g. coordinate systems)
  - The data mapping must allow clients to get full model instances for each table row

- Easy to Build

  - The mapping structure must be independant of the data structure

  - The mapping syntax should facilitate the building of mapping components and templates.

- Complex Data Mapping
  - The mapping syntax must support to retrieve data over several tables
  - The mapping syntax must be able to filter the data rows that are part of the current instance
  - The mapping syntax must be able to group data rows in an set of instances

The syntax specified in this standard gives rules to build consistant annotations for any model. However, it do not prevent to do foolish things, the same way that a programming langage grammar does not prevent against irrelevant software.

3 Syntax

3.1 Mapping Block Structure

The mapping block is outside of the data tables. Its scope is the whole VOTable. Its stucture is given below.

```
<VODML>
   <MODELS> ... </MODELS>
   <GLOBALS> ... </GLOBALS>

   <TEMPLATES tableref=...> ... </TEMPLATES tableref=...>
   <TEMPLATES tableref=...> ... </TEMPLATES tableref=...>
   ...
</VODML>
```

Listing 1: INSTANCE bloc example

The mapping construction rules are the same whatever the model or the data layout are.

- The mapping is located in a `<VODML>` block, child of `<VOTABLE>`.
- The mapping elements reflect the model structure.
- The `<VODML>` block starts with a list of implemented models.
- There is one `<TEMPLATES>` per mapped `<TABLE>`.
- There is one `<GLOBALS>` block containing data shared by the whole mapping.

3.2 MODELS

The models blocks contains the list of the models mapped in the block. Models referenced in `MODELS` are not necessary VO standards, but thet must be access through a VODML URI.

3.3 GLOBALS

Contains `INSTANCE`s with that can be used everywhere in the `VODML`.

```
<GLOBALS>
  <INSTANCE ID="SpaceCoordFrame" dmrole="">
    <INSTANCE dmrole="coords:SpaceFrame.refPosition" dmtype="coords:StdRefLocation">
      <VALUE dmrole="coords:StdRefLocation.position" dmtype="ivoa:string" value="NoS
    </INSTANCE>
    <VALUE dmrole="coords:SpaceFrame.spaceRefFrame" dmtype="ivoa:string" value="ICRS
    <VALUE dmrole="coords:SpaceFrame.equinox" dmtype="coords:Epoch" value="NoSet"/>
  </INSTANCE>
  <INSTANCE >
    ...
  </INSTANCE>
  ...
</GLOBALS>
```

Listing 2: INSTANCE bloc example

`INSTANCE`s contained in `GLOBALS` should have an `@ID` attribute so that they can be referenced from other instances

Child	Role
`INSTANCE`	Model instances with a scope covering the whole VOTable .

Table 1: Allowed `GLOBALS` children

`GLOBALS` has no attributes.

3.4 TEMPLATES

There is one `TEMPLATE` block for each mapped table in the VOTAble

3.5 INSTANCE

Mapping for either object type or a datatype instances.

```
<INSTANCE dmrole="ds:dataset.Dataset.dataID" dmtype="ds:dataset.DataID" ID="_ds_">
    <VALUE dmrole="ds:dataset.DataID.title" value="Gaia TS Mapping Test" />
```

Child	Role
INSTANCE	The table data are mapped on these instances.
ARRAY	There is one instance per table row. The structure of those instance is given by the ARRAY children
COMPOSITION	The table data are mapped on an instance list

Table 2: Allowed TEMPLATES children

Attribute	Requ. level	Role
@tableref	Mandatory	The @ID or the @name of the mapped table

Table 3: TEMPLATES attributes

```
    <VALUE dmrole="ds:dataset.DataID.datasetID" value="ivoa://gaia/ts/12345" />
    <VALUE dmrole="ds:dataset.DataID.creatorDID" value="ivoa://esa/gaia/" />
    <VALUE dmrole="ds:dataset.DataID.version" value="0.0" />
    <VALUE dmrole="ds:dataset.DataID.date" value="2018:11:11" />
    <VALUE dmrole="ds:dataset.DataID.creationType" value="LiteMappingTest" />
    <INSTANCE dmrole="ds:dataset.DataID.creator" dmtype="ds:dataset.Creator">
        <INSTANCE dmrole="ds:party.Role.party" dmtype="ds:party.Individual">
            <VALUE dmrole="ds:party.Party.name" value="VODML-Team" />
       </INSTANCE>
    </INSTANCE>
</INSTANCE>
```

Listing 3: INSTANCE bloc example

Child	Role
INSTANCE	Another embedded instance .
VALUE	Primitive attribute .
COMPOSITION	Composition with a limited set of INSTANCE e.g. author list
ARRAY	Composition with a set of INSTANCE corresponding each to one row of the data table.
FILTER	TbC

Table 4: Supported INSTANCE children

Attribute	Requ. level	Role
@dmrole	Mandatory	VODML role of the instance. May be empty for instances child of `GLOBALS`
@dmtype	Mandatory except for reference	VODML type of the instance.
@dmref	Mandatory for reference	reference to another instance in the mapping bloc.
@ID	Mandatory if the instance is referenced by other instances	Unique identifier of the instance.

Table 5: Supported attributes for `INSTANCE`

@dmrole	@dmref	@dmtype	use case
yes	yes		Reference to another instance. The element must have no child
yes		yes	Instance serialization The element must enclose the instance content

Table 6: Supported attribute patterns for `INSTANCE`

3.6 VALUE

Mapping for primitive attributes. `VALUE` are the model leaves that point onto real data.

```
<INSTANCE dmrole="model:value.example" dmtype="model:value.Example">
    <VALUE dmrole="model:preset.value" value="Preset Value" />
    <VALUE dmrole="model:ref.value" ref="fieldID" />
    <VALUE dmrole="model:reforpreset.value" value="Preset Value" ref="fieldID" />
 </INSTANCE>
```

Listing 4: VALUE examples

`VALUE`s have no children.

A Changes from Previous Versions

No previous versions yet.

Attribute	Requ. level	Role
@dmrole	MUST	VODML role of the instance attribute.
@dmtype	MUST	VODML type of the instance attribute.
@value	MUST if no `@ref` element attribute. MAY if `@ref` element attribute	Value of the instance attribute. If `VALUE` has also a `@ref`, `@ref` MUST be resolved first. `VALUE` MUST be taken when `@ref` cannot be resolved
@ref	MUST if no `@value` element attribute. MAY if `@value` element attribute	Reference of the data element (`FIELD` or `PARAM`). MUST refer to an element of the `TABLE` referenced by the current `TEMPLATE` The client MUST first look for a `FIELD` matching `@ref`. In case of failure, it MUST look for a `PARAM`

Table 7: Supported attributes for `VALUE`

@dmrole	@dmtype	@ref	@value	Role
yes	yes	yes		The instance attribute must take the value pointed by `@ref`
yes	yes		yes	The instance attribute must take the value set in `@value`
yes	yes	yes	yes	The instance attribute must take the value pointed by `@ref` and the this set in `@value` if `@ref` cannot be resolved

Table 8: Supported attribute patterns for `VALUE`