Data Model for Mapping

Version 0.51

IVOA DM WG Internal Draft

2004-04-10

Working Group: http://www.ivoa.net/twiki/bin/view/IVOA/IvoaDataModel

Probable Editors:

David Berry, Jonathan McDowell, Patrick Dowler, Brian Thomas

Authors:

IVOA Data Model Working Group

Abstract

This document defines the Mapping data model.

Status of this document

This is a Working Group Internal Draft only. It is inappropriate to reference this document.

Acknowledgments

Members of the IVOA Data Model Working Group, including representatives of the US NVO, Astrogrid, Starlink, the Canadian VO, and the AVO have contributed to the present draft.

Contents

1 Introduction and Scope

Here is a draft of a possible approach to Mapping. David Berry was the first to propose a detailed approach to Mapping, based on Starlink's AST. The present document was drafted by Jonathan McDowell, with a slightly different take, and does not yet represent a consensus with the other editors.

This document supplements the Quantity data model definition by specifying details of the Mapping interface model defined in that document.

Transformations may be needed to describe the relationships between different world coordinate systems and/or data value systems associated with a resource. For instance, the data values in a sub-mm data set may be described in different systems such as flux, effective antenna temperature, etc. Pixel positions in a CCD image can be described in terms of pixel coordinates, focal plane, coordinates, sky coordinates, etc. It would be advantageous for each data set to include descriptions both of the dimensionally distinct systems in which the values and positions in the data set can be represented, and also of the transformations between these different systems.

The scheme described below allows for the transformation of both scalar and vector values. It is based on the experience gained over the past 8 years or so with the Starlink AST library (see http://axp0.ast.man.ac.uk/ dsb/ast/ast.html).

2 The Mapping class

For the purposes of this document, a transformation is a machine-readable recipe for converting a vector representing a single position within some (unspecified) input coordinate system into a corresponding vector in another unspecified output coordinate system. The base class is referred to as a Mapping and represents a pair of transformations: one being the forward transformation (from input to output coordinate system) and the other being the inverse transformation (from output to input coordinate system). A Mapping must define at least one of these transformations - optionally one transformation may be undefined.

Note, an important distinction is made in this document between a Mapping, and the coordinate systems which describe the inputs and outputs of the Mapping. A Mapping is simply a recipe which describes a sequence of mathematical operations to be applied to a supplied N-dimensional vector, in order to create a corresponding output vector. A Mapping does not include any description of the input and output coordinate systems and does not make any assumptions about their nature. For instance, the recipe represented by a particular Mapping may be 'multiply all elements of the input vector by 3.5 and subtract 6 from the second element' of the input or output vectors.

However, a Mapping is only of any use when it is used to transform positions from some known coordinate system into another. So there must be some way of specifying the properties and nature of these coordinate systems, such as 'effective antenna temperature in units of Kelvins' or 'galactic

longitude and latitude in units of degrees'. Such descriptions are handled by the Frame class which is described in the Quantity Data Model document.

An important consequence of formally dissociating a Mapping from its input and output coordinate systems is that it is then possible to treat Mappings as 'black boxes' without any reference to what the inputs and outputs represent. Thus complex recipes can be formed by combining simple atomic steps.

The Mapping class encapsulates the properties which are common to all Mappings, but does not itself define any transformations. For this reason, it is an abstract class which cannot be instantiated. A wide range of subclasses of Mapping can be defined, each of which implements specific forms of transformation. It is these sub-classes which are instantiated.

In addition to the input and output values, any given mapping may take as input a set of parameters. Such parameterized mappings are common - for instance, a cosmological mapping might take the values of H_0, Ω_0 and Λ as parameters.

The interface to the general Mapping class gives access to a list of Parameters - possibly with Parameter a derived class from BasicQuantity. Having generic parameters, free of specific semantics, helps in applications which explore parameter space systematically (such as fitting algorithms). The implementation of the parameters and their serialization may be defined specifically for each type of mapping (recall that Mapping is abstract, so there is no generic implementation or serialization).

The properties of a Mapping are as follows:

Name	Type	Description
StandardName	string	Name of the mapping used for text output, e.g. "Matrix multiply"
Nin	integer	The length of the input vector (i.e. the number of axes in the input coordinate system).
Nout	integer	The length of the output vector (i.e. the number of axes in the output coordinate system).
inverted	boolean	True if the forward and inverse transform methods are swapped at invocation

Notes:

1. The StandardName is provided as a label for the mapping type; in user-output application we often want a more human-readable name than just the class type.

2. The Nin and Nout properties need not be equal. For instance, a Mapping which converts from spherical (longitude,latitude) to Cartesian (x,y,z) coordinates on a unit sphere will have 2 inputs but 3 outputs.

3. The Inverted property is used to modify the behaviour of the Mapping by inverting its normal forward direction. For each Mapping, the

'forward' and 'inverse' mappings are predefined (and specified in documentation). For instance, the ExpMap mapping by default calculates $\exp(x)$ as the forward transform and $\ln(x)$ as the inverse transform. By using the Invert method to toggle the Inverted attribute, we create a Mapping which uses $\ln(x)$ as the forward transform and $\exp(x)$ as the inverse transform.

The principal methods of a Mapping are:

Name	Purpose
fwdTransform(in: List): List	Use a Mapping to transform a set of input pos tions into corresponding output positions
invTransform(in: List): List	Apply the inverse transform; use the Mappi to transform a set of output positions into co responding input positions.
hasFwdTransform(): bool	The forward transform exists
hasInvTransform(): bool	The inverse transform exists
getStandardName(): string	Return name of mapping
getParams(): List	*Return parameters of mapping*
simplify(): Mapping	Create a new Mapping which is a simpler form a given Mapping
invert(): void	Invert a Mapping in place by toggling its Invert property
fwdDerivative(in: List, inno: int, outno: int): List	Optional method, returns the rate of change of output with respect to input at giv input
invDerivative(in: List, inno: int, outno: int): List	Optional method; returns rate of change of input with respect to output at giv output.

plus constructor and property accessor methods. We include definitions of derivative methods - giving the gradient of the mapping - as these can be useful.

In addition, we anticipate that methods will need to be added to propagate errors and uncertainties through the mapping. These would be similar to the fwdTransform and invTransform methods but would accept a list of accuracy values in addition to the supplied positions, and return a list of transformed accuracies in addition to the transformed positions.

2.1 Composite Mappings

An important sub-class of Mapping is a CompositeMap (or SeriesMap). A CompositeMap is a compound Mapping which allows two component Mappings (of any class) to be composed together in series (function composition) to form a more complex Mapping:

$$M(A \to B) = M_1(A \to C) * M_2(C \to B)$$

The methods and constructors associated with CompositeMap are

Name	Purpose
CompositeMap(m1: Mapping, m2:Mapping): CompositeMap	Constructor for composition
decompose(): List	Return list of individual mappings

My colleagues prefer the terms SeriesMap and ParallelMap to CompositeMap and AggregateMap respectively.

Thus to compose mappings M_1 and M_2 as above, we would do

```
M = composeMappings( M1, M2 )
```

Since a CompositeMap is itself a Mapping, it can be used as a component in forming further CompositeMaps. Mappings of arbitrary complexity may be built from simple individual Mappings in this way. It is easy to create overly-complex CompositeMaps, that is, CompositeMaps in which the effect of some of the Mappings cancel out. A simple example would be a CompositeMap which combined a (potentially complex) Mapping in series with its own inverse. The total effect of this CompositeMap would be equivalent to a unit Mapping. Whilst legal, this is bad practice for several reasons: the CompositeMap will take longer to evaluate, it will introduce greater rounding errors, and will require a larger description. For this reason, the Simplify method of the Mapping class is important: it creates a new Mapping from a supplied Mapping by removing any redundant steps in the supplied Mapping.

2.2 Aggregate Mappings

An AggregateMap (or ParallelMap) is a compound Mapping which combines two or more Mappings in parallel:

$$M((A_1, A_2, A_3) \to (B_1, B_2, B_3)) = (M_1(A_1 \to B_1), M_2(A_2 \to B_2, M_3(A_3 \to B_3))$$

Here one Mapping transforms some subset of the input coordinates for each position and the second and third Mappings simultaneously transform the remaining input coordinates; this supports a common case where an N-dimensional coordinate system is transformed using separable mappings on each axis.

The methods associated with AggregateMap are

Name	Purpose
new Mapping(maplist: List): Mapping	Create aggregate map
getMappingMembers(): List	Return list of individual mappings in aggregate

so that in the example above `M = Mapping( List( M1,M2 ))` and `M.getMappingMembers()` would return a list consisting of M1 and M2.

2.3 Atomic Mappings

Appendix A lists a large collection of concrete sub-classes of the Mapping class, each of which implements a specified simple mathematical operation or function, e.g.

$$x + y, x - y, x * y, x/y, x^y, \sin(x), \max(x, y)$$

Some of these atomic Mappings require extra properties - the parameters discussed earlier - to hold parameter values used in the mathematical operation.

2.4 Packaged Mappings

It is possible to represent many complicated transformations by combining the atomic Mappings listed in Appendix A into nested CompositeMaps and AggregateMaps. However, the evaluation of such transformations (e.g. using the transform method) could be made more efficient by having dedicated Mapping classes to represent commonly used specialised transformations. An example is matrix multiplication. A possible approach to providing a Mapping to do matrix multiplication would be to combine multiple AddMaps, MultMaps and PermMaps into a CompositeMap which would do the required matrix multiplication. Another approach would be to define a whole new sub-class of Mapping, a MatrixMap, which encapsulates the values of the matrix elements directly. The Transform method of the MatrixMap class would do the matrix multiplication directly, using the matrix elements stored in the MatrixMap. By comparison, the Transform method of the equivalent CompositeMap would invoke the Transform methods of its component Mappings recursively until the underlying atomic Mappings were reached. This interpretive process is bound to be slower than the direct approach implemented by the Matrixmap class.

So it would be advantageous to define a collection of packaged Mappings for commonly required complex operations, but at the same time leaving open the possibility of building up complex Mappings from atomic Mappings in order to cover more esoteric cases which are not common enough to justify their own dedicated Mapping class. Other examples of useful packaged Mappings would be various forms of spherical projections, conversion to and from different celestial, spectral or temporal coordinate systems, pin-cushion distortion, spherical to Cartesian conversion, etc. Of course, these packaged Mappings could themselves be combined together within a CompositeMap to form even more complex Mappings.

Some obvious packaged mappings we require are the Polynomial, and its very common special case, the degree 1 polynomial or linear mapping (LinearMap).

For any complicated mapping, one can implement it as an atomic mapping, or as a single packaged mapping combining many atomic mappings. As a general principle of software development, breaking things down into manageable chunks is a good idea - we recommend defining extremely complex mappings as packages of PackagedMappings of intermediate complexity.

As a policy question, we must decide whether different defined mappings can have overlapping functionality (what Brian Thomas calls the 'wild west approach') or whether the IVOA will organize the mappings in strictly exclusive sets (so that the "exp(x)" and "raise to power" mappings do not exist independently since they are special cases of the same math.)

2.5 Extensions

Many transformations in astrophysics cannot be constructed simply by composing atomic mappings; they require a full programming language to express. For example, cosmological transformations in the new standard picture, such as mapping flux to rest frame luminosity with redshift as a parameter, require the evaluation of elliptic integrals. We must provide ways to model and serialize user-defined mappings which can point to external code or services to perform the transformations. This is beyond the scope of the current draft.

2.6 Predefined list of Atomic and Packaged Mappings

In this section we present lists of concrete sub-classes of the Mapping class which implement various useful cases. Nin and Nout are the number of input and output values for the Mapping. * means that any positive number may be used. A Nout value of Nin means that Nout must be equal to Nin (in this case, each output will usually be a function of the corresponding input). The 'function' is the function carried out by the forward transform.

The first table gives packaged mappings covering useful cases which often arise in astronomical data, and the second table gives atomic mappings which implement basic mathematical operations.

The detailed definition of these packaged mappings is deferred to a later document. The special case of FITS WCS mappings is covered in a later section.

Name	Nin	Nout	Function	Parameters
linear-map	1	1	$y = y_0 + d(x - x0)$	x_0, y_0, d
linear-map	1	1	$y = ax + b$	(alternate constructor)
Polynomial	1	1	$y = \sum a_n x^n$	a_n
MatrixMult	n	n	Matrix multiplication	n * n
ProjMap	2	2	Individual FITS-WCS spherical projections	
SphRotate	2	2	Rotation on the sphere	
SkyMap	2	2	Convert between ICRS, galactic etc	
SpecMap			Spectral coordinate conversions	
GrismMap			From FITS-WCS paper 3	
TimeMap			Conversions between UTC, TT etc	

Table 1: Packaged Mappings

Note that we can have alternate constructors, for convenience, that implement the same internal composition of atomic mappings. For instance, the linear-map example above has parameters x_0 and y_0 which are degenerate in the equivalent polynomial representation, but which are often a useful representation.

Name	Nin	Nout	Function	Parameters
unit-mapping	*	*	Identity (unit) map	-
PermMap	*	*	Re-arrange input values into a different order, optionally adding or removing inputs or outputs.	List: Description of the axis rearrangement
LutMap	*	*	Calculate output values from a look-up table.	List: The look-up table values
AddMap	*	1	Adds all input values together	
SubMap	2	1	Input 2 minus input 1	
MultMap	*	1	Multiply all input values together	
DivMap	2	1	Input 2 divided by input 1	
PowMap	2	1	Input 2 is raised to the power of input 1	
ShiftMap	*	nin	Add a given value onto each input value	double: The shift for each input
ZoomMap	*	nin	Multiply each input by a specified value	double: The scale for each input
RecipMap	*	nin	Take the reciprocal of each input	
SqrMap	*	nin	Square each input	
SqrtMap	*	nin	Square root of each input	
RaiseMap	*	nin	Raise each input to a given power	double: The power for each input
CosMap	*	nin	Cosine of each input	
SinMap	*	nin	Sine of each input	
TanMap	*	nin	Tangent of each input	
LogMap	*	nin	Natural logarithm of each input	
ExpMap	*	nin	Exponential of each input	
MaxMap	*	1	Maximum of all inputs	
MinMap	*	1	Minimum of all inputs	
TestMap	3	1	Output is equal to input 2 if input 1 is non-zero. Otherwise output is equal to input 2.	
GtMap	2	1	Output is 1 if (input 1 ¿ input 2) and zero otherwise.	
LtMap	2	1	Output is 1 if (input 1 ¡ input 2) and zero otherwise.	
EqMap	*	1	Output is 1 if all inputs are equal and zero otherwise.	double: tolerance for equality
OrMap	*		Output is 1 if any of the inputs are non-zero.	
AndMap	*		Output is 1 if all of the inputs are non-zero.	

Table 2: Atomic Mappings

2.7 Packaged mappings and FITS-WCS

The highest priority practical example of a Mapping to be supported in the VO is the FITS celestial WCS mapping.

The general FITS-WCS family of mappings has N inputs and M outputs, with N often, but far from always, equal to 2, and M forced to be equal to N by the addition of degenerate axes when needed.

The parameters to the base N-dimensional FITS-WCS mapping are:

Name	No. values	Type	Meaning
CTYPE	1	string	Name of mapping type (usually last 4 chars of CTYPEn)
CUNIT	N	string	Units of CRVAL and CDELT
CRPIX	N	double	Reference input coordinates
CRVAL	N	double	Reference output coordinates
CDELT	N	double	Reference scale, output units per input unit
PC	N * N	double	Rotation matrix
NPV	1	integer	Number of projection parameters
PV	NPV	double	Projection parameters
NPS	1	integer	Number of string parameters
PS	NPS	string	String parameters
CRDER	N	double	Random error on mapping
CSYER	N	double	Random error on mapping

Each value of CTYPE has associated values of NPV and NPS, the number of projection parameters, both of which are usually zero. The PC matrix is sometimes represented by an angle (CROTA). Note that FITS-WCS also specifies CUNIT values and a WCSNAME value which are contained in our Frame rather than in Mapping, as are the individual coordinate names from the first 4 characters of CTYPE. The CRDER and CSYER parameters were added to FITS-WCS in the 2002 paper (Greisen and Calabretta A&A 395,1061) and have not been widely used (if at all). They provide a single characteristic error for the mapping independent of coordinate values.

The most common form of FITS-WCS mapping is one in which the first two dimensions are a celestial FITS-WCS (CelWCS) and the remaining dimensions are linear mappings with no cross terms, so that the PC matrix is diagonal except for those first two dimensions. In our terminology, the FITS-WCS paradigm is an aggregate mapping in which no attempt is made to express the separability of the different axes. The clearest way to implement FITS-WCS may be to always convert it to an aggregation of linear-maps, spectral conversion mappings, and CelWCS when possible.

The CelWCS mapping has the following special parameters in addition to the ones listed above:

Name	No. values	Type	Meaning
LONPOLE	1	double	Longitude of pole
LATPOLE	1	double	Latitude of pole

In addition, several parameters are not required for Mapping but may need to be present in the Frame or CoordSystem:

- EQUINOX, which is only required if the coord names are RA and DEC or ELON and ELAT.
- RADESYS, a string giving the reference system, which is only required if the coord names are RA and DEC or ELON and ELAT.
- MJD-OBS, only required for geocentric apparent coordinates.

The functional definition of the FITS-WCS mapping is given in Greisen and Calabretta (2002) and Calabretta and Greisen (2002, A&A 395, 1077).

We note that spectral FITS-WCS conventions have recently been proposed as FITS-WCS Paper 3; they are not covered in the current draft of this document. The complication introduced by Paper 3 is that the first four letters of CTYPEn may signify a further linear transformation in addition to the non-linear transformation supported by the last four letters, and the interpretation of the parameters now involves further calculation involving CUNIT.

As already implemented in Starlink's AST, the FITS-WCS celestial mapping can be instantiated as a composition of mappings:

1. Linear map origin shift, as described by CRPIX
2. Scale and Matrix multiply (CDELT and CD)
3. Spherical projection map (CTYPE)
4. Spherical rotation (CRVAL, LONPOLE, LATPOLE)

I propose that the FITS-WCS celestial mapping represent a predefined Compound mapping, recognized as such. This means that when software deserializes the FITS-WCS mapping, it can instantiate it as a composition of the individual mappings enumerated above.

3 XML Serialization

3.1 Simple mappings

The general Mapping class is serialized with a `<mapping>` tag. Within the mapping, the specific mapping type is given.

- The trivial identity map is serialized by `<unit-mapping/>`.
- linear-map is serialized by `<linear-map>` with attributes ref, value, and step:

```
<mapping>
<linear-map ref=1.0 value=1483212.3 step=600.0/>
</mapping>
```

- Polynomial is serialized with an "nparams" attribute equal to the degree plus 1, and with "param" elements giving the values of the parameters. Example:

```
<mapping>
        <m:polynomial nparams="3">
                <m:param>131281.4</m:param>
                <m:param>-.00013</m:param>
                <m:param>4.823</m:param>
        </m:polynomial>
</mapping>
```

- The atomic mappings are serialized by the name of the mapping, with an nparams attribute, enclosing param elements, as for polynomial:

```
<mapping>
        <m:sqrtMap/>
</mapping>
<mapping>
       <m:shiftMap nparams=1>
                <m:param>-42.10</m:param>
       </m:shiftMap>
</mapping>
```

(for the atomic mappings, we may instead want to consider using attributes for the parameters and defining each one).

3.2 XML Serialization of FITS Celestial WCS

We serialize the CelWCS as `<wcsmap>` with an attribute `type` and the elements `<refvals>`, `<refpos>`, `<scales>` corresponding to CRVAL, CRPIX, CDELT and each containing two values. The PC matrix may be serialized as `<wcspc>` with 4 values, or (if non-skew) as a single angular value `<rotation>` expressed in degrees.

Example:

```
<mapping>
        <fitscelwcs type="TAN">
                <refvals>131.2181 -31.1284</refvals>
                <refpos>512.1 512.1</refpos>
                <scales>-0.0016 0.0016</scales>
                <rotation>48.3121</rotation>
        </fitscelwcs>
</mapping>
```

An alternate proposal is to use the literal FITS keys:

```
<mapping>
        <fitscelwcs>
               <CTYPE>"RA---TAN" "DEC--TAN"</CTYPE>
                <CRVAL>131.2181 -31.1284</CRVAL>
                <CRPIX>512.1 512.1</CRPIX>
                <CDELT>-0.0016 0.0016</CDELT>
                <CD>0.66507 -0.74678 0.74678 0.66507</CD>
        </fitscelwcs>
</mapping>
```

(Note that we might have a FITSCelWCS mapping on an axis quantity which involves the 3rd and 4th axes of a 5-dimensional FITS image. Having XML tags like `<CRVAL4>` which would depend on these axis numbers seems like a bad idea.)

A more complicated example:

```
<mapping>
        <fitscelwcs type="SIN" npv=2>
                <refvals>131.2181 -31.1284</refvals>
                <refpos>512.1 512.1</refpos>
                <scales>-0.0016 0.0016</scales>
                <wcspc>0.8660 0.5000 -0.5000 0.8660</wcspc>
           <lonpole>210.0</lonpole>
           <latpole>-90.0</latpole>
           <pv>0.12 -0.11</pv>
        </fitscelwcs>
</mapping>
```

3.3 XML serialization of a compound mapping

Consider a mapping between pixel coordinates X,Y,Z and world coordinates RA, Dec, Velocity, in which the Z-to-velocity mapping is a composite of a polynomial (wavelength calibration) and a rescaling.

We express aggregation (parallel combination) with mappings following one after another and composition (series combination) with the `<compose>` tag. The interpretation is that the aggregation members are in order of the axes, and the compositions are performed in the order of input to output.

```
<mapping>
            <wcsmap type="TAN">
                    <refvals>131.2181 -31.1284</refvals>
                    <refpos>512.1 512.1</refpos>
                    <scales>-0.0016 0.0016</scales>
                    <rotation>48.3121</rotation>
            </wcsmap>
            <compose>
             <polynomial nparams=2>
               <param>4000.2</param><param>1.5</param>
             </polynomial>
             <linear-map ref=4861.0 val=0.0 step=61.7/>
            </compose>
</mapping>
```