11 March 2002, version 0.94

VOTable: A Proposed XML Format for Astronomical Tables

Clive **Davenhall**, Royal Observatory Edinburgh, UK
Daniel **Durand**, Canadian Astronomy Data Centre, Canada
Pierre **Fernique**, Observatoire Astronomique de Strasbourg, France
Robert **Hanisch**, Space Telescope Science Institute, USA
David **Giaretta**, Royal Observatory Edinburgh, UK
Tom **McGlynn**, NASA Goddard Space Flight Center, USA
François **Ochsenbein**, Observatoire Astronomique de Strasbourg, France
Alex **Szalay**, Johns Hopkins University, USA
Andreas **Wicenec**, European Southern Observatory, Germany
Roy **Williams**, California Institute of Technology, USA

1. Introduction

The VOTable format is a proposed XML standard for representing a table. In this context, a table is an unordered set of rows, each of a uniform format, as specified in the table metadata. Each row is a sequence of table cells, and each of these is either primitive data types, or an array of such primitives. The format is derived from the Astrores format [1], itself modeled on the FITS Binary Table format [2].

1.1. Example

A simple example of a VOTable document is:

```
<?xml version="1.0"?>
<!DOCTYPE ASTRO SYSTEM "VOTable.dtd">
<ASTRO ID="v1.0">
    <DEFINITIONS>
        <COOSYS ID="myJ2000" equinox="2000." epoch="2000." system="eq_FK5"/>
    </DEFINITIONS>
    <RESOURCE>
      <PARAM ID="Observer" datatype="A" arraysize="*" value="William Herschel">
       <DESCRIPTION>This parameter is designed to store the observer's name</DESCRIPTION>
      </PARAM>
      <TABLE name="Stars">
        <DESCRIPTION>Some bright stars</DESCRIPTION>
        <FIELD ID="Star-Name" ucd="ID_MAIN" datatype="A" width="10"/>

        <FIELD ID="RA" ucd="POS_EQ_RA" ref="myJ2000"
                                        unit="deg" datatype="E" precision="F3" width="7"/>

        <FIELD ID="Dec" ucd="POS_EQ_DEC" ref="myJ2000"
                                        unit="deg" datatype="E" precision="F3" width="7"/>

        <FIELD ID="Counts" ucd="NUMBER" datatype="I" arraysize="2x3x*"/>
        <DATA>
            <TABLEDATA>
<TR><TD>Procyon</TD><TD>114.827</TD><TD> 5.227</TD><TD>4 5 3 4 3 2 1 2 3 3 5 6</TD></TR>
<TR><TD>Vega</TD><TD>279.234</TD><TD>38.782</TD><TD>8 7 8 6 8 6</TD></TR>
            </TABLEDATA>
        </DATA>
      </TABLE>
    </RESOURCE>
```

```
</ASTRO>
```

This table shows the positions of two stars, each with a name and two floating point numbers as coordinates, together with a variable-length, multidimensional array called "Counts". The star names have a fixed length of 10 characters (padded by trailing blanks). The floating-point numbers (RA and Dec) are in degrees, and assumed to have three decimal digits (`precision="3"`), irrespective of the number of digits presented in the data. The frame of the coordinate system is specified explicitly with the `COOSYS` element. Associated with the table is a parameter (`PARAM`), which is to be interpreted as a string, which in this example is the name of the observer (William Herschel).

1.2. Why VOTable?

Astronomers have always been at the forefront of developments in information technology, and funding agencies across the world have recognized this by supporting the Virtual Observatory movement, in the hopes that other sciences and business can follow their lead in making online data both *interoperable* and *scalable*.

VOTable is designed as a flexible storage and exchange format for tabular data, with particular emphasis on astronomical tables.

Interoperability is encouraged through the use of standards (XML); because physical quantities are tagged not only with units, but also through a Uniform Content Descriptor (UCD) that expresses the nature of the quantity (eg. Gunn J magnitude, declination). The XML fabric allows applications to easily validate an input document, as well as facilitating transformations through XSLT (eXtensible Style Language Transformation) engines.

Grid Computing

VOtable has built-in features for big-data and Grid computing. It allows metadata and data to be stored separately, with the remote data linked according to the Xlink model. Processes can then use metadata to 'get ready' for their input data, or to organize third-party or parallel transfers of the data. Remote data allow the metadata to be sent in email and referenced in documents without pulling the whole dataset with it: just as we are used to the idea of sending a pointer to a document (URL) in place of the document, so we can now send metadata-rich pointers to data tables in place of the data itself. The remote data is referenced with the URL syntax protocol://location, meaning that arbitrarily complex protocols are allowed.

When we are working with very large tables in a distributed-computing environment ("the Grid"), the data stream between processors, with flows being filtered, joined, and cached in different geographic locations. It would be very difficult if the number of rows of the table were required in the header – we would need to stream in the whole table into a cache, compute the number of rows, then stream it again for the computation. In the Grid-data environment, the component in short supply is not the computers, but rather these very large caches! Furthermore, these remote data streams may be created dynamically by another process or cached in temporary storage: for this reason VOTable can express that remote data may not be available after a certain time (expiration). Data on the net may require authentication for access, so VOTable allows expression of password or other identity information (the 'rights' attribute).

Data Storage: Flexible and Efficient

The data part of VOTable has three different formats. There is a pure XML format so that small tables can be easily handled in their entirety by XML tools. The FITS binary table format is well-known to astronomers, and VOTable can be used either to encapsulate such a file, or to re-encode the metadata; unfortunately it is difficult to stream FITS, since the dataset size is required in the header (NAXIS2 keyword), and FITS requires a specification up front of the maximum size of its variable-length arrays. The BINARY format is supported for efficiency and ease of programming: no FITS library is required, and the streaming paradigm is supported.

We hope that VOTable can be used in different ways, as a data storage and transport format, and also as a way to store metadata alone (table structure only). In the latter case, we can imagine a VOTable structure being sent to a server, which can then open a high-bandwidth connection to receive the actual data, using the previously-digested structure as a way to interpret the stream of bytes from the data socket.

Alternatively, the metadata can be sent alone as an implicit query to a server, which will respond with the data part of the table filled in.

VOTable can be used for small numbers of small records (pure XML tables), or for large numbers of simple records (streaming data), or it can be used for small numbers of larger objects. In the latter case, there will be software to spread large data blocks among multiple processors on the Grid. Currently the most complex structure that can be in a VOTable Cell is a multidimensional arrays; in a further release we will use XML modularity to allow users to put arbitrarily complex objects in the cells, each with their own XML schema and namespace.

Future

In future versions of the VOTable format, we expect to benefit from both usage and tool-building. Such tools include presentation and transformations of the metadata (and data too, when it is in XML), using XML transformation language and software: XSL and XSLT. We would like to migrate to the more powerful document validation provided by XSchema (rather than DTD).

We also expect XSchema to allow better modularization of the document schema, so that, for example, users can put whatever serialized objects they wish into the table cells. In this way, we expect to use VOTable for handling the flow of large data objects through Grid resources, objects such as FITS files or XDF [7] documents. It would also mean, for example, that the description of a table could contain arbitrary HTML instead of the current version – plain text with paragraph markers; or that an XML definition of non-standard astronomical coordinate systems could be seamlessly integrated.

VOTable is currently specified for acting not only as a way to write a data table, but also as a way to specify how to address a *request* to data tables. We expect to sharpen and formalize this dichotomy with the benefit of experience, adding more sophisticated querying mechanisms and protocols.

We expect to add features for efficiency in the future also: to specify that the data stream has a particular sort order, to specify that a column in one table is a key in to another table; to specify that one table is an index into another.

2. Data Model

In this section we define the data model of a VOTable, and in the next sections its syntax when expressed as XML. The data model of VOTable can be expressed as:

VOTable = hierarchy of **Metadata** + **Tables**

Metadata = **Parameters** + **Infos** + **Descriptions** + **Links** + **Fields**

Table = list of **Fields** + **Data**

Data = stream of **Rows**

Row = list of **Cells**

Cell = **Primitive**
or variable-length list of **Primitives**
or multidimensional array of **Primitives**

Primitive = integer, character, float, floatComplex, etc (see table below)

Metadata is divided into that which concerns the table itself (parameters), and the definitions of the fields (or column attributes) of the table. Each Field represents the metadata at the top of the column, for example, in the table above, a Field is “RA (degrees)”. The instantiations of this class are the Cells, in the example these are the numbers for particular stars (114.827 and 279.234).

A parameter (`PARAM`) is similar to a `FIELD`, except that it has a “value” attribute. Parameters can be used for storing FITS keywords or any other information pertaining to the table itself or its environment.

An informative parameter (`INFO`) is a restricted form of the `PARAM` – it has only the name / value pair of attributes.

The ordered list of Fields at the top of the table thus provides a template for a Row object, (also called a *record*). The template allows interpretation of the data in the Row. In VOTable, there is no advance specification of the number of rows in the table: this is to allow streaming of large tables, as discussed above. The record is a set of Cells, with the number of Cells the same for each Row, and the same as the number of Fields defined in the Metadata.

`datatype` **value**	**Type**	**Length (bytes)**
L	Logical	1
X	Bit	*
B	Byte	1
I	Short Integer	2
J	Integer	4
K	Long integer	8
A	ASCII Character	1
U	Unicode Character	2
E	Floating point	4
D	Double	8
C	Float Complex	8
M	Double Complex	16

2.1. Primitives

Each Cell is composed from Primitives, each of which is a datatype of fixed-length binary representation, as enumerated in the accompanying table. Cells may consist of a single Primitive (this is the default); or a multidimensional array of Primitives. (see next section).

The 12 primitive types listed in the table above all have the same length in bytes (The bit type uses the minimal number of bytes, so that b bits take the integer part of (r+7)/8 bytes). It is this fixed size that allows efficiency in storage, so that the memory used is minimized.

In VOTable, characters are Primitives, either one byte for an ASCII character or two bytes for a Unicode character. Strings can be represented as a fixed or variable length array of characters. We can represent a 1D array of strings as a 2D array of characters, but given the logic above, the Cell can contain a variable-length array of fixed-length strings (but not a fixed-length array of variable-length strings).

2.2. Multidimensional Arrays

A table cell can contain an array of a given primitive type. The array is specified by a sequence of dimensions, with the first dimension changing fasts, and the last dimension that may be variable in length. For example, a table cell may contain a set of images, each 64x64 bytes:

```
<FIELD ID=”thumbs” datatype=”B” arraysize=”64x64x10*”/>
```

The string in the “arraysize” attribute expressed these dimensions, each integer separated by the “x” character, except the last. The last (slowest-varying) subscript of a multidimensional array may have

variable length, meaning that the dimensionality of the final subscript may be different for different rows of the table. In this case, there may be just an asterisk, in which case the array may be arbitarily large; or a number followed by an asterisk, meaning that this subscript is guaranteed not to exceed this value.

Variable-length arrays are more efficient in storage and data transfer, but less efficient computationally, because extra pointer dereferencing is required to access the data. The reason that only the slowest-varying subscript can be variable is effectively a demand that the data provider pack the data into a small number of fixed-size containers as much as is possible, rather than a large number of small containers.

2.3. FITS Binary Tables

VOTable is closely compatible with the FITS Binary Table format. Henceforth, we shall abbreviate "FITS Binary Table and its Conventions" simply by the word "FITS". Given a FITS file that represents a binary table, the header may be converted to VOTable, with a pointer to the original file, or with the original file included directly in VOTable. Since the original file is still present, it is clear that no data has been lost. There is a PARAM element in VOTable which can be used to hold the FITS keyword, value, and comment string.

But we might ask two more significant questions, about how much of the FITS header and data can be represented in VOTable. The answer is that there is considerable overlap.

What can FITS do but not VOTable?

FITS has semantics for how data is to be represented when printed, as the non-mandatory TDISP keyword: for example F12.4 means 12 characters are to be used, and 4 decimal places. This has been converted in VOTable as the attributes width and precision which, connected with **datatype**, are semantically identical. Note that error estimation and the number of digits to print are rather different semantically.

FITS has a complex semantics (the "Substring Array" convention) for structuring a single string as a collection of substrings, and VOTable does not support this. VOTable allows fixed and variable-length strings, as well as variable-length arrays of fixed length strings.

What can VOTable do but not FITS?

VOTable supports separating of data from metadata and the streaming of tables, and other ideas from modern distributed computing. It bridges two ways to express structured data: XML and FITS. It tries (through UCD) to express formally what is the semantic content of a parameter or field. It has the hierarchy and flexibility of XML. FITS does not handle Unicode (extended alphabet) characters.

It should be noticed that the transformation of FITS to VOTable is meant to be reversible: the conversion of a FITS file into VOTable does not lose any information, and a transformation back into FITS is possible. It will however not be possible to transform any VOTable into a FITS file without losing some information.

3. XML

VOTable is constructed with XML (extensible Markup Language), a powerful standard for structured data throughout the Internet industries. It derives through simplification from SGML, which has been a standard in technical documentation for many years. XML consists of *elements* and payload, where an element consists of a *start tag* (the part in angle brackets), the payload, and an *end tag* (with angle brackets and a slash). Elements can contain other elements. Elements can also bear `attributes` (keyword-value combinations), such as the FIELD elements above.

The payload may be in two forms: parsed or unparsed character data. Examples are:

```
<text>Fran&#231;ois</text>
<text><![CDATA[ a <= (b & c) ]]></text>
```

In the first example, the sequence ç is interpreted as part of the ISO/IEC 10646 character set, and translates to an accented character, so that the text is "François". The second example uses the special CDATA sequence so that the characters <, >, and & can be used without interpretation; in this case, any

ASCII characters are allowed except the terminating sequence "]]>". For more information, see any book on XML.

3.1. XSchema Lists

A standard data representation in astronomy has been CSV (Column Separated Values), also supported for table exchange by many commercial programs, including widely used spreadsheets. While designed for simplicity in a limited context, it has become bloated; furthermore Microsoft and International Standards Organization have different standards.

Therefore we have decided to reject CSV (character-separated values) as a data representation. However, within a table cell, multiple Primitives can be formatted with "XSchema List". These lists are used to express compound primitives (complex numbers) and multidimensional arrays. Text tokens are separated by contiguous whitespace (ASCII space `0x20`, tab `0x9`, carriage-return `0xD`, newline `0xA`, vertical tab `0xB`). There are no null tokens, comments, quote characters, or separators. However, it is possible to include special characters through an escape mechanism as in HTML: for example the string "New York" would be encoded as "New York".

Thus a table cell that contains an array of three complex numbers could be represented as:

```
<TD>1.0 0.0   -0.5 0.866   -0.5 -0.866</TD>
```

3.2. Syntax policy

The element names are in uppercase in order to help the reading. The attribute names are preferably in lowercase (with an exception for the `ID` attribute). Element and attribute names are further distinguished in this paper by being in `fixed-width` font.

3.3. ID and name

The `FIELD` and `PARAM` elements provide both `ID` and `name` attributes: the ID is meant as a formal naming of the objects in the VOTable document, while the name is meant for presentation purposes. The ID's of the fields and parameters must be unique throughout the document – this is part of the XML specification, and we intend that eventually ID's can be used to tie together data sources and applications that read and write them. The name of an object, when not present, defaults to the ID value,and when it is present is intended for presentation to humans.

Currently, however, the ID attribute (as defined by Xpointer standard) is used in order to refer to other elements in the document. The attribute `ID` is a string beginning with a letter or underscore (_), followed by a sequence of letters, digits, or any of `.-_:`, and each ID *must be unique* in the XML document. For example `ref="apple"` refers to the element that contains `ID="apple"` in the current XML document. In principle any element may have an ID attribute; elements that support the `ref` attribute (and can point to those with ID) are: `FIELD`, `PARAMETER`, and `TABLE`.

The `ID` is different from the `name` attribute in that (a) the ID attribute must be unique (or else the document is considered invalid in the XML sense), whereas names need not be unique; and (b) There should be support in the parsing software to look up references and extract the relevant element with matching ID. It should be noted that this referencing mechanism will not work unless the parser uses a *validating* parser.

3.4. Xlink and STREAM

The STREAM element is used to point to remote table data, and as such it closely follows the W3C specification called "Xlink". The `STREAM` implements the interface defined by Xlink; in particular it is an Xlink with `type="locator"`. However, `STREAM` has more attributes than Xlink allows for: a `rights` attribute for authentication information; and `expires` attribute for when the link may cease to be valid; and an `encoding` attribute if the data is filtered, (for example compression or binary-to-ascii filtering). Therefore we will wait until a future release to formalize the relationship between Xlink and `STREAM`.

4. Parameters and Hierarchy

The VOTable document is a hierarchy of RESOURCE elements, each of which can contain other RESOURCEs. The root of the tree is the single ASTRO element that constitutes the entire XML document, and the leaves of the tree are TABLEs. The resource elements may also contain parameters (PARAM).

The following is a complere VOTable which contains no tables, only a hierarchy of parameters.

```
<?xml version="1.0"?>
<!DOCTYPE ASTRO SYSTEM "VOTable.dtd">
<ASTRO ID="v1.0">
    <RESOURCE ID="Stars">
        <PARAM ID="Mass" datatype="E" unit="solMass" value="1"/>
        <RESOURCE ID="BigStars">
            <PARAM ID="Mass-big" datatype="E" unit="solMass" value="10"/>
        </RESOURCE>
        <RESOURCE ID="SmallStars">
            <PARAM ID="Mass-small" datatype="E" unit="solMass" value="0.2"/>
            <RESOURCE ID="VerySmallStars">
                <PARAM ID="Mass-tiny" datatype="E" unit="solMass" value="0.05"/>
            </RESOURCE>
        </RESOURCE>
    </RESOURCE>
</ASTRO>
```

5. Table

The Table element is written in XML as a DESCRIPTION and LINK elements, that describe the nature of the data in the table, together with a collection of parameters (PARAM). The LINK element may be parsed (see section 4.5), and the description may be split into paragraph elements. The rest of the table metadata describes the FIELDs that together make up each row of the table.

A FIELD element may have several sub-elements, including the informational DESCRIPTION, and LINK, as well as VALUES, that can express limits and ranges of the values that the corresponding cell can contain, such as minimum, maximum, or enumeration of possible values.

The FIELD must contain a datatype attribute, which expresses the nature of the data that is in the cells of this column of the table. This determines how data is read and stored internally. If it is not present, an exception is thrown.

Each table cell may contain more than one of the specified datatype, and this is specified with the arraysize attribute, as explained above, inducing a multidimensional array structure on those table cells. Because strings are not primitives (but rather characters are), so a variable-length character string would have datatype=”A: and arraysize=”*”:

```
<FIELD ID=”A_String” datatype=”A” arraysize=”*”/>
```

Unicode is a way to represent characters that is an alternative to ASCII. It uses two bytes per character instead of one, it is strongly supported by XML tools, and it can handle a large variety of international alphabets. Therefore VOTable supports not only ASCII strings (datatype=”A”), but also Unicode (datatype=”U”).

For details of the exact meaning of these data types, please see section 8.

If the data is written as TABLEDATA, and a table cell contains an array or complex number, then it should be encoded as multiple numbers separated by white space.

A FIELD may also specify a “null” attribute through a VALUES element. For example: null=”-99”. When this value is found in that data, it is assumed that no data exists for that table cell. The parser may choose to use this also when unparsable data is found, and the null value will be substituted instead.

5.1. Numerical Accuracy

The VOTable format is meant for transferring, storing, and processing tabular data, it is not intended for presentation purposes. Therefore (in contrast with Astrores) we generally avoid giving rules on presentation, such as formatting. However, we retain the "`width`" attribute of the `FIELD`, which is meant as a hint to the presentation system about the number of characters to use for input or output of the quantity. Inevitably some at least of the data will have to be presented – either as actual tables, or in graphs, etc... In any case, some estimation of the accuracy has to be known.

But there is a semantic difference between a number written as "5.12" and one that is written "5.1200". In that the former implies three significant digits of accuracy, and the latter five digits. Therefore the number of digits to show is not purely a presentation matter, but part of the metadata content of the number.

VOTable therefore provides the `precision` attribute in the `FIELD` element to express the number of significant digits, or equivalently, the log of the implied error estimate of the numbers in the column. More control is available through an initial character: setting this to "`E`" rather than the default "`F`" implies that the `precision` measures the relative error (significant figures) rather than its absolute error (decimal places). Thus `precision="E5"` means an implied relative error 10^{-5}, and `precision="5"` or "`F5`" means an implied absolute error 10^{-5}.

5.2. Units

The quantities in a column of the table may have physical units, and this is specified by the `units` attribute of the `FIELD`. Examples are:

```
units="cm-2.s-1.keV-1"
units="erg.s-1"
```

The syntax of this string is defined in reference [3].

5.3. Unified Content Descriptors

The CDS in Strasbourg has used the metadata from thousands of astronomical tables to make a hierarchical glossary of the scientific meanings of the data in those tables [4]. Of 1600 entries in the glossary, here are a few typical examples.

`PHOT_INT-MAG_B`	Integrated total blue magnitude
`ORBIT_ECCENTRICITY`	Orbital eccentricity
`STAT_MEDIAN`	Statistics Median Value
`INST_QE`	Detector's Quantum Efficiency

The `ucd` attribute of the `FIELD` is to hold this information.

5.4. VALUES element

The `VALUES` element of the `FIELD` is designed to hold subsidiary information about the nature of the data in the field. It may have `MIN` and `MAX` elements, and it may contain `OPTION` elements. The latter contains name and value attributes, and may also contain more `OPTION` elements, so that a hierarchy of keyword-values pairs may be associated with each field.

There may also be a `null` attribute. If this is present, and a table cell takes this value, it is assumed to mean that no data is present. For example, there may be a convention that missing values in a table are expressed with –99, in which case the "missing" table cell would be set to this. Therefore any cell in this field with this value is assumed to have no data.

5.5. LINK Elements as URL Templates

The `LINK` element is to provide pointers to other documents or data servers on the Internet through a URL. In Astrores, the `LINK` element may be part of the `RESOURCE`, `TABLE` or `FIELD` elements. The `href` attribute of

the LINK is meant to provide a URL that is at least valid syntactically, even though there need be no assurance that the link will actually connect and deliver data. It may be that a strange protocol is implied that the parser does not know about, for example gridftp://server/file. However, parsers are expected to understand at least the file, http and ftp protocols.

The gref attribute is meant for a higher-level protocol of some type, perhaps a logical name for a data resource, perhaps a GLU reference [5].

In some cases, there is additional semantics for the LINK element, where the href and gref attributes are not a simple URL, but rather a template for creating URL's. Depending on the content-role attribute of the LINK, and the nature of the parent element, the ID tags from the table may be substituted into the template to create an implicit new column, as explained in the next section.

5.5.1. Pattern-matching and Substitution

When a LINK element appears within a TABLE, there is extra functionality implied. The href or gref attributes may not be a simple link, but instead a template for a link. For example, in the table of section 1.1, we might have:

```
<LINK href="http://us-vo.org/lookup?Star=${Star-Name}&RA=${RA}&DE=${Dec}"/>
```

The implication is that the text is seen in the context of a particular row of the table, and a substitution filter is applied. If the selected row of the table is the first one, the result of the substitution would be:

```
http://us-vo.org/lookup?Star=Procyon&RA=114.827&DE=5.227
```

Whenever the pattern ${…} is found in the original link, the part in the braces is compared with the set of name attributes of the fields of the table. If a match is found, then the value from that field of the selected row is used in place of the ${…}. If no match is found, no substitution is made. Thus the parser makes available to the calling application a value of the href and gref attributes that depends on which row of the table has been selected. Another way to think of it is that there is not a single link associated with the table, but rather an implicitly defined new column of the table. This mechanism can be used to connect each row of the table to further information resources.

The action attribute in this release of the standard is simply a string. In a future release, it may gain an implied string substitution filter as with href and gref.

The purpose of the link is defined by the content-role attribute. The allowed values are query, hints, and doc. The first implies that string substitution should be used as defined above, and the latter two imply first that no substitution is needed, and that the link points to either information for use by the application (hints) or human-readable documentation (doc).

5.6. Type Attribute

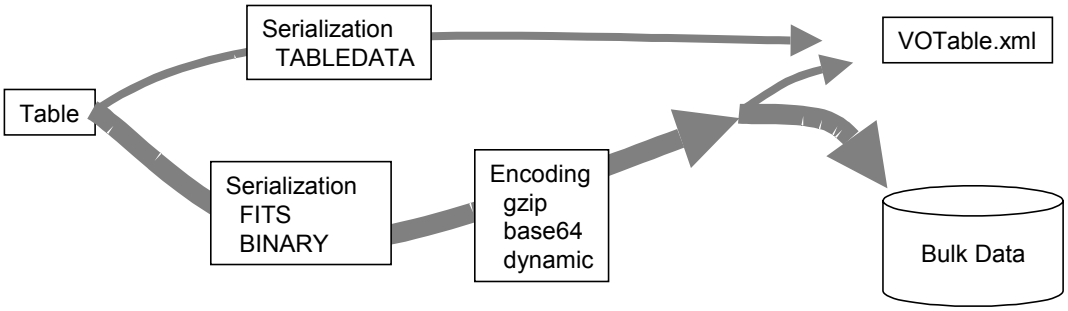
The type attribute of the FIELD may carry values that express the status of the field when the enclosing table describes the components for submitting a query, rather than a data document. If the value is "noquery", then the marked field is ignored in the creation of the action query – this field does not belong to the form described by the set of FIELDs. A computed column (value computed from other FIELDs) is a typical example of a FIELD which content is to be ignored in the generation of a query.

If type="trigger", then the marked field contains data necessary for correct LINK generation. If for instance only the columns "RA" and "Dec" are asked, but a link requires the knowledge of a "RecordNumber" to be operational, the result contains the additional column "RecordNumber" flagged as a "trigger" field.

6. Data Content

While the bulk of the metadata of a VOTable document is in the FIELD elements, the data content of the table is in a single DATA element. The data is organized in "reading" order, so that the content of each row

appears in the same order as the order of the FIELD tags, with each row having the same number of items as there are FIELD tags.



The data section of theVOTable document is created through a data pipeline. The abstract table is first serialized by one of several methods, then it may be Encoded for compression or other reasons. The result may be embedded in the xml file (*local* data), or it may be *remote* data.

The figure shows how the abstract table is rendered into the VOTable document. First the data is *serialized*, as XML or LIST (whitespace-separated text tokens), or a FITS binary table, or a simple binary format. This data stream may then be *encoded*, perhaps for compression or to convert binary to text. Finally, the data stream may be put in a remote file with a URL-type pointer in the VOTable document; or the table data may be embedded in the VOTable.

6.1. Data Serialization

The serialization elements and their attributes are:

6.1.1. TABLEDATA

This element is a way to build the table in pure XML, and is the only serialization method that does not allow an encoding or a remote data stream. It contains TR elements, which in turn contain TD elements. An example:

```
<TABLEDATA>
<TR> <TD>Procyon</TD> <TD>114.827242</TD> <TD>5.227506</TD>  </TR>
<TR> <TD>Vega</TD>    <TD>279.234106</TD> <TD>38.782992</TD> </TR>
</TABLEDATA>
```

While this serialization has a high overhead in the number of bytes, it has the advantage that XML tools can manipulate and present the table data directly.

Each item in the TD tag is passed to a reader that is implicitly defined by the datatype attribute of the corresponding FIELD, which attempts to read the object from it. If it reads a value that is the same as the NULL value for that field, then the cell will contain that value, and is therefore assumed to contain no data.

The reader may not succeed, for example if we try to parse the string 245.6h756 into a float. In this case, the parser may choose to insert the null value (no data available), or it may use a NaN (not a number), or it may throw an exception. It might however be useful if the data provider would warn that invalid data patterns could be used to designate non-existing data.

If a cell contains an array or complex number, it should be encoded as multiple numbers separated by whitespace. However in the case of character and Unicode strings, no separators are required. Here is an example of a table with a single row, that has arrays in the table cells:

```
<TABLE>
  <FIELD ID="String" datatype="A" arraysize="10"/>
  <FIELD ID="VarString" datatype="A" arraysize="*"/>
  <FIELD ID="Complex" datatype="C"/>
  <FIELD ID="VarFloats" datatype="E" arraysize="*"/>

  <DATA><TABLEDATA>
```

```
  <TR>
    <TD>Apple</TD>
    <TD>Orange</TD>
    <TD>0.0 1.0</TD>
    <TD>3.141 2.718 0.618</TD>
  </TR></TABLEDATA></DATA>
</TABLE>
```

The first entry is a fixed-length array of 10 characters; since the value being presented ("Apple") has 5 characters, this is padded with trailing blanks. The second cell is a variable-length array of characters, and will contain the 6 characters in "Orange". The complex number *i* is next, represented by its real and imaginary parts (0 and 1). The fourth cell is a variable-length array of floats, in this case there are 3 of them.

6.1.2. FITS

The FITS format for binary tables is well-used in astronomy [2], and its structure is a major influence on the VOTable specification. Metadata is stored in a header section, followed by the data. The metadata is substantially equivalent to the metadata of the VOTable format. One important difference is that VOTable does not require specification of the number of rows in the table, an important freedom if the table is being created dynamically from a stream.

The VOTable specification does not define the behavior of parsers with respect to this doubling of the metadata. A parser may ignore the FITS metadata, or it may compare it with the VOTable metadata for consistency, or other possibilities.

The following code shows a fragment that might have been created by a FITS-to-VOTable converter. Each FITS keyword has been converted to a PARAM, and the data itself is remotely stored and gzipped at an ftp site:

```
<RESOURCE>
    <PARAM name="EPOCH" datatype="E" value="1999.987">Original Epoch of the coordinates</PARAM>
    <PARAM name="TELESCOP" datatype="A" arraysize="*" value="VTel" />
    <INFO name="HISTORY">The very first Virtual Telescope observation made in 2002</INFO>
    <TABLE>
        <FIELD .....(insert field metadata here) >
        <DATA><BINARY><STREAM
            encoding="gzip"
            href="ftp://archive.cacr.caltech.edu/myfile.fit.gz"/></BINARY></DATA>
    </TABLE>
</RESOURCE>
```

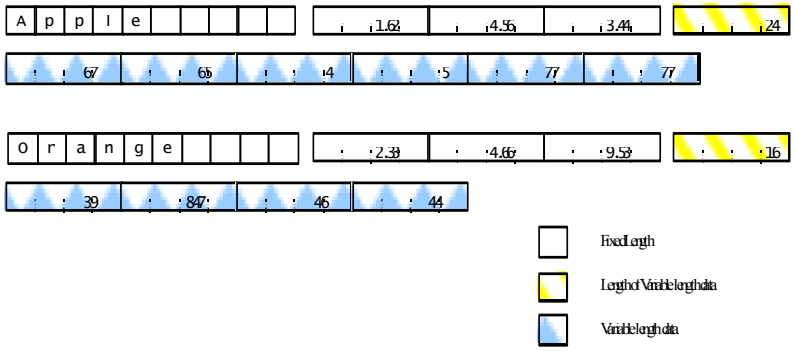
6.1.3. BINARY

The Binary format is intended to be easy to read by parsers, so that additional libraries are not required. It is just a sequence of byte strings, the length of each string corresponding to the `datatype` attributes of the `FIELD` elements in the metadata. The binary format consists of a sequence of records, with no header bytes, no alignment considerations, no block sizes.

Table cells may contain arrays of primitive types, each of which may be of fixed or variable length. In the former case, the number of bytes is the same for each instance of the item, as specified by the `arraysize` attribute of the `FIELD`. If all the fields have a fixed `arraysize`, then each record of the binary format has the same length, as the sum of `arraysize` times the length in bytes of the corresponding `datatype`.

In the case of variable-length arrays of primitives, however, the Binary format becomes more complex. Each record has first a part for the fixed-length fields, (as well as four bytes for each of the variable-length fields), followed by a section for the variable length fields. The four bytes for the variable-length field is interpreted as a four-byte integer with the length *in bytes* of the variable-length array, as shown in the figure. The parser can then compute the appropriate offset by adding the lengths of previous variable-length sections of the record, in order to read the data.

The figure shows the byte layout for this binary format for the same data as the example in section 6.1.1. Each record has a ten-byte character array, then an array of three four-byte floats, then a variable number of



four-byte integers, represented as the length in bytes (24 and 16 in the two records shown), then the corresponding number of bytes.

6.2. Data Encoding

As a result of the serialization, the table has been converted to a byte stream, either text or binary. If the TABLEDATA serialization is used, then those elements are directly in the XML document, and there is no possibility for encoding. However, if one of the other serializations is used, we might *encode* the result to compress it, or for other reasons.

In this version of VOTable, it is not possible to encode individual columns of the table: the whole table must be encoded in the same way.

In order to use an encoding of the data, it must be enclosed in a STREAM element, whose attributes define the nature of the encoding. The `encoding` attribute is a string that should indicate to the parser how to undo the encoding that has been applied. Parsers should understand and interpret these values at least:

- `encoding="gzip"` implies that the data following has been compressed with the gzip filter, so that gunzip or similar should be applied.
- `encoding="base64"` implies that the base64 filter has been applied, to convert binary to text.

The parser may also respond to the string `dynamic`, implying that the data is in a remote resource (see below), and the encoding will be delivered with the header of the data. This occurs with the http protocol, where the MIME header indicates the type of encoding that has been used. The default value of the encoding attribute is the null string, meaning that no encoding has been applied. In future releases, we will allow more complex strings in the encoding attribute, allowing combinations of encoding filters and a way for the parser to find the software needed for the decoding.

6.3. Remote Data

If the encoding of the data produces text, or if the serialization is naturally text-based, then it can be directly embedded into the XML document. However, if the data encoding produces binary, or if the data is very large, it may be preferable to keep the data separate from the metadata. The text contained in the STREAM element is then interpreted as the location of the data, rather than the data itself. The location is specified in a URL-type syntax, for example:

```
<STREAM href="ftp://server.com/mydata.dat"/>
<STREAM href="http://webserver.com/mydata.dat"/ expires="2002-01-31">
<STREAM href="gridftp://server.com/mydata.dat" actuate="onLoad" />
<STREAM href="file://mydata.dat"/>
```

The examples are the well-known anonymous ftp, and http protocols. Also is an example of a Grid-based access to data, and finally a local file, which is on the local file system.

There are further attributes of the STREAM element that may be useful. The expires tag is for when the VOTable is part of a pipeline of data processing, when data is being dynamically created and stored in temporary space, in which case it may be deleted after a certain time limit. The expires attribute expresses when a remote resource may cease to become valid, and is expressed in Universal Time in the same way as the FITS specification [2], for example:

```
<STREAM expires="2002-01-31T12:00:00">
```

The rights attribute expresses authentication information that may be necessary to access the remote resource. If the VOTable document is suitably encrypted, this attribute could be used to store a password.

The "actuate" attribute is borrowed from the XML Xlink specification, expressing when the remote link should be actuated. The default is "onRequest", meaning that the data is only fetched when explicitly requested (like a link on an HTML page), and the "onLoad" value means that data should be fetched as soon as possible (like an embedded image on an HTML page).

7. Definitions for a Query

Following Astrores [1], the details on the input parameters used to generate a query is currently assuming the ASU [6] procotol: PARAMETER and FIELD, paired in the form *name*=*value*, where *name* is the contents of the name attribute, and *value* represents a constraint written with the ASU conventions (e.g. <8 or 12.0..12.5 which denotes a range of values). Such pairs are appended to the action specified in the LINK element contained in the **RESOURCE**, separated by the ampersand (&) symbol – in a way similar to the FORM HTML syntax. PARAMETER or FIELD which have the attribute type=noquery are however ignored, and are never paired. A valid query could be:

```
http://server/asu-xml?-source=I/271/out&-out.max=99&-c=01:02:03-12:31:14&Rmag=12.0..12.5
```

and the corresponding VOTable document that generates it:

```
<?xml version="1.0"?>
<!DOCTYPE ASTRO SYSTEM "VOTable.dtd">
<ASTRO ID="v1.0">
<DEFINITIONS>
        <COOSYS ID="J2000" equinox="2000." epoch="2000." system="eq_FK5"/>
</DEFINITIONS>

<RESOURCE ID="GSC2.2.01" type="meta">
    <DESCRIPTION>The GSC 2.2 Catalogue (STScI, 2001,455851237 objects)</DESCRIPTION>

    <PARAM ID="MaxRec" name="-out.max" ucd="NUMBER" datatype="I" value="50">
        <DESCRIPTION>Maximal number of retrieved records</DESCRIPTION>
        <VALUES multiple="no" type="legal">
            <MIN value="1" inclusive="yes" /> <MAX value="9999" inclusive="yes" />
        </VALUES>
    </PARAM>

    <PARAM ID="SkyLocation" name="-c" ucd="POS_EQ" datatype="A" arraysize="*">
        <DESCRIPTION>Target position in the sky according to the ASU syntax in J2000</DESCRIPTION>
    </PARAM>

    <TABLE>
        <FIELD name="GSC2.2" UCD="ID_MAIN" datatype="A" arraywidth="14">
            <DESCRIPTION>Identification of the object</DESCRIPTION></FIELD>

        <FIELD name="RA(ICRS)" UCD="POS_EQ_RA_MAIN"
                                    ref="J2000" datatype="D" width="10" precision="6" unit="deg">
            <DESCRIPTION>Right Ascension in ICRS (J2000), at Epoch</DESCRIPTION> </FIELD>

        <FIELD name="DE(ICRS)" UCD="POS_EQ_DEC_MAIN"
                                    ref="J2000" datatype="D" width="10" precision="6" unit="deg">
            <DESCRIPTION>Declination in ICRS (J2000)</DESCRIPTION></FIELD>
```

```
        <FIELD name="Rmag" UCD="PHOT_PHG_R" datatype="E" width="5" precision="2" unit="mag">
            <DESCRIPTION>?Magnitude in F photographic band (red)</DESCRIPTION></FIELD>

        <FIELD name="e_Rmag" UCD="ERROR" datatype="F" width="5" precision="2" unit="mag">
            <DESCRIPTION>? Mean error on Rmag (2)</DESCRIPTION>
        </FIELD>
        <LINK content-role="query" action="asu-xml?-source=I/271/out&amp;" />
    </TABLE>
</RESOURCE>
</ASTRO>
```

Note that the `RESOURCE` displaying the parameters accessible for a query has the `type="meta"` attribute; it is also assumed that only one `LINK` having the `content-role="query"` attribute together with an `action` attribute exists within the current `RESOURCE`.

8. Document Structure

The VOTable document consists of a single all-containing element called `ASTRO`, which may contain a `DESCRIPTION` and a number of `PARAMETER` elements which contain strings, a `DEFINITIONS` element, and a `RESOURCE` element.

8.1. DEFINITIONS element

This element may contain a definition of a coordinate system, stored in a `COOSYS` element: there are attributes for equinox and epoch, as well as a specification of the coordinate system. We expect that future versions of VOTable may have a more formal structuring of the coordinate system definition which is currently extracted from Astrores. There may also be one or more `PARAMETER` elements that may contain user-specific data. Each of these may have an ID attribute, that can be referenced with the ref attribute of a field. Thus we can achieve grouping of fields (by having members of the group reference the same part of the definitions sections). We can also extend the definition of a field by adding user-specific data.

8.2. RESOURCE element

There may be multiple `RESOURCE` elements, and each of these may contain `PARAMETER` and `DESCRIPTION` elements, as well as `DEFINITIONS` (as above). There may be `LINK` elements to provide URL-type pointers that give further information.

The main ingredient of the `RESOURCE` element is one or more `TABLE`s. These are described in sections 3-5 of this document.

9. Adapted from FITS Binary Table Specification

Logical If the value of the `datatype` attribute specifies data type `L`, the contents of the field shall consist of ASCII `T` or `t` indicating true or ASCII `F` or `f`, indicating false. A 0 byte (hexadecimal 0) indicates an invalid value.

Bit Array If the value of the `datatype` attribute specifies data type `X`, the contents of the field shall consist of a sequence of bits starting with the most significant bit; the bits following shall be in order of decreasing significance, ending with the least significant bit. A bit array shall be composed of an integral number of bytes, with those bits following the end of the data set to zero.

Byte If the value of the `datatype` attribute specifies data type `B`, the field shall contain a sequence of zero or more members, composed of unsigned bytes. The most significant byte shall be first. Within each byte the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance. An array of bytes is also known as a “blob”, and can be used for storing general byte data.

Character If the value of the `datatype` attribute specifies data type `A`, the field shall contain a character string of zero or more members, composed of ASCII text. This character string may be terminated before the length specified by the repeat count by an ASCII NULL (hexadecimal code 00). Characters after the

first ASCII NULL are not defined. A string with the number of characters specified by the repeat count is not NULL terminated. Null strings are defined by the presence of an ASCII NULL as the first character.

Unicode Character If the value of the `datatype` attribute specifies data type `u`, the field shall contain a character string of zero or more members, composed of Unicode text. Each character is represented by two bytes, in order that many non-Latin alphabets can be represented.

16-Bit Integer If the value of the `datatype` attribute specifies datatype `I`, the data in the field shall consist of twos-complement signed 16-bit integers, contained in two bytes. The most significant byte shall be first. Within each byte the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance.

32-Bit Integer If the value of the `datatype` attribute specifies datatype `J`, the data in the field shall consist of twos-complement signed 32-bit integers, contained in four bytes. The most significant byte shall be first, and subsequent bytes shall be in order of decreasing significance. Within each byte, the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance.

64-Bit Integer If the value of the `datatype` attribute specifies datatype `K`, the data in the field shall consist of twos-complement signed 64-bit integers, contained in eight bytes. The most significant byte shall be first, and subsequent bytes shall be in order of decreasing significance. Within each byte, the most significant bit shall be first, and subsequent bits shall be in order of decreasing significance.

Single Precision Floating Point If the value of the `datatype` attribute specifies datatype `E`, the data in field shall consist of ANSI/IEEE-754 32-bit floating point numbers. All IEEE special values are recognized. The IEEE `NaN` is used to represent invalid values, unless the `null` attribute has been set, in which case it is used instead.

Double Precision Floating Point If the value of the `datatype` attribute specifies datatype `D` or `F`, the data in the field shall consist of ANSI/IEEE-754 64-bit double precision floating point numbers. All IEEE special values are recognized. The IEEE `NaN` is used to represent invalid values, unless the `null` attribute has been set, in which case it is used instead.

Single Precision Complex If the value of the `datatype` attribute specifies datatype `C`, the data in the field shall consist of a sequence of pairs of 32-bit single precision floating point numbers. The first member of each pair shall represent the real part of a complex number, and the second member shall represent the imaginary part of that complex number. If either member contains a `NaN`, the entire complex value is invalid.

Double Precision Complex If the value of the `datatype` attribute specifies datatype `M`, the data in the field shall consist of a sequence of pairs of 64-bit double precision floating point numbers. The first member of each pair shall represent the real part of a complex number, and the second member of the pair shall represent the imaginary part of that complex number. If either member contains a `NaN`, the entire complex value is invalid.

10.Sample VOTable Document

```
<!DOCTYPE ASTRO SYSTEM "VOTable.dtd">
<ASTRO version="v0.9" xmlns="http://vizier.u-strasbg.fr/VOTable">
    <DESCRIPTION>This is an example VOTable document</DESCRIPTION>
    <DEFINITIONS>
        <COOSYS ID="myJ2000" equinox="2000." epoch="2000" system="eq_FK5"/>
    </DEFINITIONS>
    <RESOURCE name="GSC1.2">
        <DESCRIPTION>
This is an excerpt of the HST Guide Star Catalog, Version 1.2 (Lasker+ 1996).  This version was re-reduced with PPM catalogue.
        </DESCRIPTION>
        <TABLE>
            <DESCRIPTION>  Default result of GSC1.2 Server around a target</DESCRIPTION>
            <FIELD ID="_r" name="_r" ucd="POS_ANG_DIST" unit="arcmin" datatype="E" width="7" precision="4">
                <DESCRIPTION>Distance from target NGC40</DESCRIPTION>
                <VALUES type="actual">
                    <MIN value="0.0"/>
                    <MAX value="10.0"/>
                </VALUES>
            </FIELD>
            <FIELD ID="gsc" name="GSC-Id" datatype="A" ucd="ID_MAIN" arraysize="10">
```

```
                    <DESCRIPTION>The GSC-Id is made of 10 digits, 5 representing the place number, and 5 the object number on the
plate.</DESCRIPTION>
                </FIELD>
                <FIELD ID="ra" name="RA(J2000)" ref="myJ2000" ucd="POS_EQ_RA" unit="deg" datatype="D" precision=" F5">
                    <DESCRIPTION>Right ascension in J2000, epoch of plate</DESCRIPTION>
                </FIELD>
                <FIELD ID="dec" name="Dec(J2000)" ref="myJ2000" ucd="POS_EQ_DE" unit="deg" datatype="D" precision=" F5" >
                    <DESCRIPTION>Declination in J2000, epoch of plate</DESCRIPTION>
                </FIELD>
                <FIELD ID="pos_err" name="PossErr" unit="arcsec" datatype="E" precision="1" ucd="ERROR">
                    <DESCRIPTION>Mean error on position</DESCRIPTION>
                </FIELD>
                <FIELD ID="Pmag" name="Pmag" ucd="PHOT_PHG_MAG" unit="mag" datatype="E" width="5" precision="2">
                    <DESCRIPTION>photographic magnitude (see n_Pmag)</DESCRIPTION>
                </FIELD>
                <FIELD ID="e_Pmag" name="e_Pmag" ucd="ERROR" unit="mag" datatype="E" width="4" precision="2">
                    <DESCRIPTION>Mean error on photographic magnitude</DESCRIPTION>
                </FIELD>
                <FIELD ID="class" name="Class" ucd="CLASS_CODE" datatype="I"  width="1" >
                    <DESCRIPTION>Class of object (0=star; 3=non-stellar)</DESCRIPTION>
                    <VALUES type="actual">
                        <OPTION name="star" value="0"/>
                        <OPTION name="galaxy" value="3"/>
                    </VALUES>
                </FIELD>
                <LINK content-role="doc" title="documentation" href="http://vizier.u-strasbg.fr/viz-bin/Cat?I/254"/>
                <DATA>
                    <TABLEDATA>
<TR><TD>0.0146</TD><TD>0430201297</TD><TD>4.7766</TD><TD>72.8474</TD><TD>3.6</TD><TD>8.59 </TD><TD>0.20</TD><TD>0</TD></TR>
<TR><TD>0.9704</TD><TD>0430200545</TD><TD>5.4576</TD><TD>72.6528</TD><TD>0.2</TD><TD>12.18</TD><TD>0.34</TD><TD>0</TD></TR>
<TR><TD>0.9730</TD><TD>0430200545</TD><TD>3.9867</TD><TD>72.9484</TD><TD>0.2</TD><TD>12.09</TD><TD>0.20</TD><TD>0</TD></TR>
<TR><TD>1.5843</TD><TD>0430202363</TD><TD>8.9587</TD><TD>72.6635</TD><TD>0.2</TD><TD>14.38</TD><TD>0.34</TD><TD>0</TD></TR>
<TR><TD>2.8586</TD><TD>0430200269</TD><TD>5.4847</TD><TD>72.8272</TD><TD>0.3</TD><TD>14.96</TD><TD>0.20</TD><TD>3</TD></TR>
<TR><TD>2.9198</TD><TD>0430200153</TD><TD>10.4746</TD><TD>72.4542</TD><TD>0.2</TD><TD>12.89</TD><TD>0.20</TD><TD>0</TD></TR>
<TR><TD>2.9215</TD><TD>0430200153</TD><TD>6.9484</TD><TD>72.1162</TD><TD>0.2</TD><TD>13.06</TD><TD>0.34</TD><TD>0</TD></TR>
<TR><TD>3.0487</TD><TD>0430202336</TD><TD>4.7586</TD><TD>72.9837</TD><TD>0.2</TD><TD>14.38</TD><TD>0.34</TD><TD>0</TD></TR>
<TR><TD>3.2247</TD><TD>0430200121</TD><TD>7.9585</TD><TD>72.5565</TD><TD>0.2</TD><TD>12.39</TD><TD>0.21</TD><TD>0</TD></TR>
<TR><TD>3.2269</TD><TD>0430200121</TD><TD>7.9484</TD><TD>72.5874</TD><TD>0.2</TD><TD>12.50</TD><TD>0.34</TD><TD>0</TD></TR>
                    </TABLEDATA>
                </DATA>
            </TABLE>
        </RESOURCE>
</ASTRO>
```

11. The DTD for VOTable

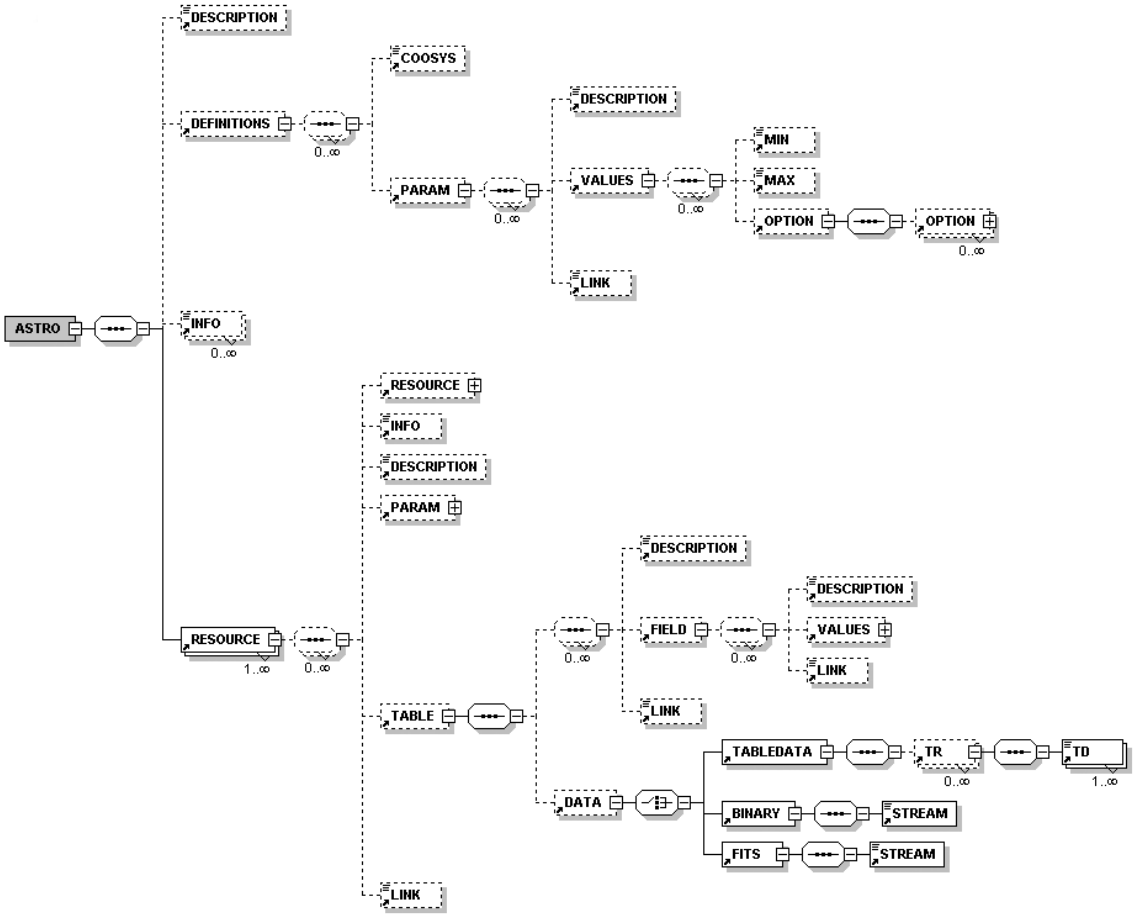
```
<!ELEMENT ASTRO (DESCRIPTION?, DEFINITIONS?, INFO*, RESOURCE+)>
<!ATTLIST ASTRO
      ID ID #IMPLIED
      version CDATA #IMPLIED
>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT DEFINITIONS (COOSYS?, PARAM?)*>
<!ELEMENT RESOURCE (RESOURCE?, INFO?, DESCRIPTION?, PARAM?, TABLE?, LINK?)*>
<!ATTLIST RESOURCE
      name CDATA #IMPLIED
      ID ID #IMPLIED
      type (results|meta) "results"
>
<!ELEMENT INFO (#PCDATA)>
<!ATTLIST INFO
                ID ID #IMPLIED
                name CDATA #IMPLIED
                value CDATA #IMPLIED

>


<!ELEMENT PARAM (DESCRIPTION?, VALUES?, LINK?)*>
<!ATTLIST PARAM
      ID ID #IMPLIED
      unit CDATA #IMPLIED
      datatype (L | X | B | I | J | A | U | E | D | C | M | K) #IMPLIED
      precision CDATA #IMPLIED
      width CDATA #IMPLIED
      ref IDREF #IMPLIED
      name CDATA #IMPLIED
      ucd CDATA #IMPLIED
      value CDATA #IMPLIED
      arraysize CDATA #IMPLIED
>
<!ELEMENT TABLE ((DESCRIPTION?, FIELD?, LINK?)*, DATA?)>
<!ATTLIST TABLE
```

```
      ID ID #IMPLIED
      name CDATA #IMPLIED
      ref IDREF #IMPLIED
>
<!ELEMENT FIELD (DESCRIPTION?, VALUES?, LINK?)*>
<!ATTLIST FIELD
      ID ID #IMPLIED
      unit CDATA #IMPLIED
      datatype (L | X | B | I | J | A | U | E | D | C | M | K) #IMPLIED
      precision CDATA #IMPLIED
      width CDATA #IMPLIED
      ref IDREF #IMPLIED
      name CDATA #IMPLIED
      ucd CDATA #IMPLIED
      arraysize CDATA #IMPLIED
      type (hidden | no_query | trigger) #IMPLIED
>
<!ELEMENT VALUES (MIN?, MAX?, OPTION?)*>
<!ATTLIST VALUES
      ID ID #IMPLIED
      multiple (yes | no) "no"
      type (legal | actual) "legal"
      null CDATA #IMPLIED
      invalid CDATA #IMPLIED
>
<!ELEMENT MIN (#PCDATA)>
<!ATTLIST MIN
      value CDATA #REQUIRED
      inclusive (yes | no) "yes"
>
<!ELEMENT MAX (#PCDATA)>
<!ATTLIST MAX
      value CDATA #REQUIRED
      inclusive (yes | no) "yes"
>
<!ELEMENT OPTION (OPTION*)>
<!ATTLIST OPTION
      name CDATA #IMPLIED
      value CDATA #REQUIRED
>
<!ELEMENT LINK (#PCDATA)>
<!ATTLIST LINK
      ID ID #IMPLIED
      content-role (query | hints | doc) #IMPLIED
      content-type CDATA #IMPLIED
      title CDATA #IMPLIED
      value CDATA #IMPLIED
      href CDATA #IMPLIED
      gref CDATA #IMPLIED
      action CDATA #IMPLIED
>
<!ELEMENT DATA (TABLEDATA | BINARY | FITS)>
<!ELEMENT TABLEDATA (TR*)>
<!ELEMENT TR (TD+)>
<!ELEMENT TD (#PCDATA)>
<!ATTLIST TD
      ref IDREF #IMPLIED
>
<!ELEMENT FITS (STREAM)>
<!ELEMENT BINARY (STREAM)>
<!ELEMENT STREAM (#PCDATA)>
<!ATTLIST STREAM
      type (locator | other) "locator"
      href CDATA #IMPLIED
      actuate (onLoad | onRequest | other | none) "onRequest"
      encoding (gzip | base64 | dynamic | none) "none"
      expires CDATA #IMPLIED
      rights CDATA #IMPLIED
>
<!ELEMENT COOSYS (#PCDATA)>
<!ATTLIST COOSYS
      ID ID #IMPLIED
      equinox CDATA #IMPLIED
      epoch CDATA #IMPLIED
      system (eq_FK4 | eq_FK5 | ICRS | ecl_FK4 | ecl_FK5 | galactic | supergalactic | xy | barycentric | geo_app) "eq_FK5"
>
```

12.Schema Diagram for VOTable



13.References

[1] Accomazzi *et. al, Describing Astronomical Catalogues and Query Results with XML*
http://vizier.u-strasbg.fr/doc/astrores.htx

[2] *FITS: Flexible Image Transport Specification*, specifically the Binary Tables Extension
http://fits.gsfc.nasa.gov/

[3] *Standards for Astronomical Catalogues: Units, CDS Strasbourg*
http://vizier.u-strasbg.fr/doc/catstd-3.2.htx

[4] *Unified Content Descriptors*
http://vizier.u-strasbg.fr/doc/UCD.htx

[5] *GLU: Générateur de Liens Uniformes, CDS Strasbourg*
http://simbad.u-strasbg.fr/glu/glu.htx

[6] *ASU: Astronomical Server URL, CDS Strasbourg*
http://vizier.u-strasbg.fr/doc/asu.html

[7] *XDF: Extensible Data format, ADC*
http://amase.gsfc.nasa.gov/xml/XDF_home.html