VOTable Format Definition
Version 1.09

IVOA Working Draft
2004-01-23

Previous versions:
1.0 (2002-04-15)

Authors:

François **Ochsenbein**	*Observatoire Astronomique de Strasbourg, France*
Roy **Williams**	*California Institute of Technology, USA*

with contributions from:

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

Abstract

This document describes the standards adopted for the version 1.1 of the VOTable format, and supersedes the previous version 1.0 of 15 April 2002. The differences between versions 1.0 and 1.1 are summarized in section 8. The main part of this document describes the adopted part of the VOTable standard; it is followed by appendices presenting extensions which have been proposed and/or discussed, but which are not part of the standard.

Status of this document

This is an IVOA Proposed Recommendation for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress". A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/Documents/.

Acknowledgments

This document is based on the W3C documentation standards, but has been adapted for the IVOA.

Contents

1 Introduction

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

1.1 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). 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. 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 tables themselves. 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 (`expires`). 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 in a VOTable may be represented using one of three different formats: TABLEDATA, FITS and BINARY. TABLEDATA 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 array.

1.2 XML Conventions

VOTable is constructed with XML[1] (extensible Markup Language), a powerful standard for structured data throughout the Internet industries. It derives from SGML, a standard used in the publishing industry and for 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).

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 `&#231;` 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.

1.3 Syntax policy

Following the general XML rule, element and attribute names are case-sensitive and have to be used with the specified capitalisation. For VOTable, we have adopted the convention that element names are spelled in uppercase and attribute names in lowercase (with an exception for the `ID` attribute). Element and attribute names are further distinguished in this paper by being shown in a `fixed-width` font.

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** + associated **TableData**
Metadata	=	**Parameters** + **Infos** + **Descriptions** + **Links** + **Fields** + **Groups**
Table	=	list of **Fields** + **TableData**
TableData	=	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 1 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 that can be found at the top of the column in a paper version of the table: in the example introduced in section 3.1 below, the first `FIELD` has its `name` attribute set to `"RA"`. The Field can be thought of as a class definition, and the table cells below it are the instances of that class.

A parameter (`PARAM`) is similar to a `FIELD`, except that it has a `value` attribute. Parameters can be seen as "constant columns", containing for instance FITS keywords or any other information pertaining to the table itself or its environment, as the `Epoch` parameter in the above example.

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

[1] http://www.w3.org/XML/

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.

From Version 1.1, columns may be logically grouped, so that it is possible to define table substructures made of column associations. Such an association is declared as a `GROUP`, which typically contains columns (`FIELD`) and associated parameters (`PARAM`).

2.1 Primitives

datatype	Meaning	FITS	Bytes
"boolean"	Logical	"L"	1
"bit"	Bit	"X"	*
"unsignedByte"	Byte (0 to 255)	"B"	1
"short"	Short Integer	"I"	2
"int"	Integer	"J"	4
"long"	Long integer	"K"	8
"char"	ASCII Character	"A"	1
"unicodeChar"	Unicode Character		2
"float"	Floating point	"E"	4
"double"	Double	"D"	8
"floatComplex"	Float Complex	"C"	8
"doubleComplex"	Double Complex	"M"	16

Table 1: List of the Primitives *(details in section 6)*

Each Cell is composed from Primitives, each of which is a datatype of fixed-length binary representation, as listed in Table 1. Cells may consist of a single Primitive (this is the default), or of a multidimensional array of Primitives (see section 2.2).

Except for the Bit type, each primitive has the fixed length in bytes given in Table 1. Bit scalars and arrays are stored in the minimum number of bytes feasible (so that b bits take the integer part of $(b+7)/8$ bytes). These primitives are described in more detail in section 6.

VOTables support two kinds of characters: ASCII 1-byte characters and Unicode 2-byte characters. 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="char"`), but also Unicode (`datatype="unicodeChar"`).

Note that strings are not a primitive type: strings are represented in VOTable as an array of characters.

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 fastest, and the last dimension that may be variable in length. For example, the following `FIELD` definition declares a table cell which may contain a set of up to 10 images, each 64x64 bytes:

```
<FIELD ID="thumbs" datatype="unsignedByte" 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.

Strings can therefore be represented in VOTable as a fixed- or variable-length array of characters:

```
<FIELD name="unboundedString" datatype="char" arraysize="*"/>
```

A 1D array of strings can be represented as a 2D array of characters, but given the logic above, it is possible to define a variable-length array of fixed-length strings, but not a fixed-length array of variable-length strings.

A convention to express an array of variable-length strings was proposed (see section C) but is not part of this standard.

2.3 Compatibility with 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. A `PARAM` element can be used to hold any FITS keyword with its value and comment string.

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.

For instance, the recommended formatting of the data for an edition of the data is expressed by 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 to the TDISP keyword.

What can FITS do but not VOTable?

FITS has a complex semantics (the "Substring Array" convention) for structuring a single string as a collection of substrings, and VOTable 1.1 does not support this *(see however the section C which proposes a compatible VOTable definition)*. The current version of 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 the UCD – see section 4.4) to express formally the semantic content of a parameter or field. It has the hierarchy and flexibility of XML: using `GROUP` elements introduced in version 1.1, a VOTable can represent structures of arbitrary complexity; and the ID attribute can be used in XML to enable what are essentially pointers.

FITS does not handle Unicode (extended alphabet) characters.

It should be noticed that the transformation of FITS to VOTable is meant to be reversible: any FITS table can be converted to a VOTable without loss of information and the resulting VOTable can be converted back to a FITS table also without loss of information. However, it is possible to create new VOTables which cannot be converted to FITS tables without loss of information.

3 The VOTable Document Structure

The overall VOTable document structure is described and controlled by its XML Schema[2] referenced at its top. This schema actually represents the VOTable definition, which means that documents claiming to represent VOTables should pass through W3C XML Schema validators without error. An illustration of the XML Schema is given in section 7.

An example is used here to illustrate the components of a VOTable document described in the following sections. Basically, a VOTable document consists of a single all-containing element called `VOTABLE`, which contains descriptive elements (`DESCRIPTION`, `DEFINITIONS`, `INFO`), followed by one or more `RESOURCE` elements. Each Resource element contains one or more `TABLE` elements, and possibly other `RESOURCE` elements.

The `TABLE` element, the actual heart of VOTable, contains a description of the columns and parameters (described in section 4) followed by the data values (described in section 5).

[2] http://vizier.u-strasbg.fr/xml/VOTable-1.1.xsd

3.1 Example

This simple example of a VOTable document lists 3 galaxies with their velocity with its error, and the estimated distance.

```
<?xml version="1.0"?>
<VOTABLE version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://vizier.u-strasbg.fr/xml/VOTable.xsd">
  <DEFINITIONS>
  <COOSYS ID="J2000" equinox="2000." epoch="2000." system="eq_FK5"/>
  </DEFINITIONS>
  <RESOURCE name="myFavouriteGalaxies">
    <TABLE name="results">
      <DESCRIPTION>Velocities and Distance estimations</DESCRIPTION>
      <PARAM name="Epoch" datatype="float" ucd="TIME_EPOCH"
             value="2003.875/">
      <FIELD name="RA"    ID="col1" ucd="POS_EQ_RA_MAIN" ref="J2000" datatype="float"
             width="6" precision="2" unit="deg"/>
      <FIELD name="Dec"   ID="col2" "POS_EQ_DEC_MAIN" ref="J2000" datatype="float"
             width="6" precision="2" unit="deg"/>
      <FIELD name="Name" ID="col3" ucd="ID_MAIN" datatype="char" arraysize="8*"/>
      <FIELD name="RVel" ID="col4" ucd="VELOC_HC" datatype="int"
             width="5" unit="km/s"/>
      <FIELD name="e_RVel" ID="col5" ucd="ERROR" datatype="int"
             width="3" unit="km/s"/>
      <FIELD name="R" ID="col6" ucd="PHYS_DISTANCE_TRUE" datatype="float"
             width="4" precision="1" unit="Mpc">
        <DESCRIPTION>Distance of Galaxy, assuming H=75km/s/Mpc</DESCRIPTION>
      </FIELD>
      <DATA>
        <TABLEDATA>
        <TR>
          <TD>010.68</TD><TD>+41.27</TD><TD>N  224</TD><TD>-297</TD><TD>5</TD><TD>0.7</TD>
        </TR>
        <TR>
          <TD>287.43</TD><TD>-63.85</TD><TD>N 6744</TD><TD>839</TD><TD>6</TD><TD>10.4</TD>
        </TR>
        <TR>
          <TD>023.48</TD><TD>+30.66</TD><TD>N  598</TD><TD>-182</TD><TD>3</TD><TD>0.7</TD>
        </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

This simple `VOTable` document shows a single `RESOURCE` made of a single `TABLE`; the table is made of 6 columns, each described by a `FIELD`, and has one additional `PARAM` parameter (the Epoch). The actual rows are listed in the `DATA` part of the table, here in XML format (introduced by `TABLEDATA`); each cell is marked by the `TD` element, and follow the same order as their `FIELD` description: *RA*, *Dec*, *Name*, *RVel*, *e_RVel*, *R*.

3.2 ID and name attributes

Most of the elements defined by VOTable may or have to bear *names*, like a `RESOURCE`, a `TABLE`, a `PARAM` or a `FIELD`. Naming an element is generally possible by means of one of or both `ID` and `name` attributes.

`ID` and `name` attributes have a different role in VOTable: the ID is meant as a *unique identifier* of an element seen as a VOTable component, while the name is meant for presentation purposes, and need not to be unique throughout the VOTable document.

The `ID` attribute is therefore required in the elements which *have to be referenced*, but in principle any element may have an `ID` attribute. According to the XML standard, the attribute `ID` is a string beginning with a letter or

underscore (_), followed by a sequence of letters, digits, or any of the punctuation characters . (dot), - (dash), _ (underscore), or : (colon).

In summary, the ID is different from the name attribute in that (a) the ID attribute is made from a restricted character set, and must be unique throughout a VOTable document whereas names are standard XML attributes and 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.

3.3 DEFINITIONS element

This element may contain a definition of a coordinate system, stored in a COOSYS element. The COOSYS element provides attributes for equinox and epoch, as well as a specification of the celestial coordinate system. The COOSYS element being the only astronomy specific part of VOTable, it may be deprecated in the future, as it is expected that a more formal structuring of the coordinate system will be designed, which would encompass conventions used in space science or solar physics. Its current definition is given below.

The DEFINITIONS element may also include one or more PARAM elements (section 4) that may contain user-specific data. Each of these may have an ID attribute, that can be referenced with the ref attribute of other elements.

The COOSYS element

This element defines a celestial coordinate system, to which the components of a position on the celestial sphere refer. It has an ID attribute — required if the COOSYS element has to be referred via the ref attribute of the position components, which is generally the case — a system attribute which specifies the coordinate system among "ICRS", "eq_FK5", "eq_FK4", "ecl_FK4", "ecl_FK5", "galactic", "supergalactic", "barycentric", "geo_app" and a user-defined "xy" value. equinox is the parameter required to fix the equatorial or ecliptic systems (as e.g. "J2000" as the default "eq_FK5" or "B1950" as the default "eq_FK4"), and epoch specifies the epoch of the positions if necessary.

As mentioned above, the COOSYS may be deprecated in the future in favor of a more generic way of describing the conventions used to define the positions.

3.4 RESOURCE element

A VOTable document contains one or more RESOURCE elements, each of these providing a description and the data values of some logically independent data structure.

Each RESOURCE may include the descriptive elements DESCRIPTION, INFO, COOSYS and PARAM; it may also contain LINK elements to provide URL-type pointers that give further information.

The main component of a RESOURCE is typically one or more TABLE elements – in other terms a RESOURCE is basically a set of related tables. The RESOURCE is recursive (it can contain other RESOURCE elements), which means that the set of tables making up a RESOURCE may become a complex structure.

A RESOURCE may have one or both of the name or ID attributes (see section 3.2); it may also be qualified by type="meta", meaning that the resource is *descriptive* only (does not contain any actual data in any of its sub-elements).

3.5 LINK element

The LINK element is to provide pointers to other documents or data servers on the Internet through a URL. In VOTable, the LINK element may be part of a RESOURCE, TABLE, GROUP or FIELD elements. The href attribute of the LINK element can comprise any arbitrary protocol, for example "http://server/file" or "bizarre://server/file". VOTable parsers are not required to understand arbitrary protocols, but are required to understand the following three common protocols: "file:", "http:" and "ftp:".

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 the Astrores format, from which VOTable is derived, there is additional semantics for the LINK element; the href attribute is used as a template for creating URL's. This behavior is explained in appendix A, and it represents a possible extension of VOTable.

In addition to the referencing href and gref attributes and to the naming name and ID attributes (see section 3.2), the LINK element may announce the mime type of the data it references with a content-type attribute (e.g. content-type="image/fits"), and specify the role of the link by a content-role attribute (e.g. content-role="doc" for an access to a documentation).

3.6 TABLE element

The TABLE element represents the basic data structure in VOTable; it is made of a description of the table structure (the *metadata*) essentially in the form of PARAM and FIELD elements (detailed in section 4), followed by the *values* of the described fields in a DATA element (detailed in section 5).

The TABLE element is always contained in a RESOURCE element: in other terms any TABLE element has a single father made of the RESOURCE element in which the table is embedded.

The TABLE element contains a DESCRIPTION element for descriptive remarks, followed by a mixed collection of PARAM, FIELD or GROUP elements which describe a parameter (constant column), a field (column) or a group of columns respectively. PARAM and FIELD elements are detailed in section 4, and the GROUP element is presented in section 4.7.

Furthermore the TABLE element may contain LINK elements that provide URL-type pointers, exactly like the LINK elements existing within a RESOURCE element (see section 3.5).

The last element included in a TABLE is the optional DATA element (see section 5): a table without any actual data is quite valid, and is typically used to supply a complete description of an existing resource e.g. for query purposes.

The TABLE element may have the naming attributes name and/or ID (see section 3.2). A TABLE may also have a ref attribute referencing the ID of another table previously described, which is interpreted as *defining a table having a structure identical to the one referenced*: this facility avoids a repetition of the definition of tables which may be present many times in a VOTable document.

4 FIELDs and PARAMeters

The atoms of the table structure are represented by FIELD and PARAM elements, where FIELD represents the description of an actual table column, while PARAM supplies a value which remains constant over the whole table, like the Epoch in section 3.1. A PARAM may therefore be viewed as a FIELD which keeps a *constant value* over all the rows of a table, and the only difference between the two elements is the existence of a value attribute in a PARAM which does not exist in a FIELD.

A FIELD or PARAM element may have several sub-elements, including the informational DESCRIPTION and LINK elements; it may also include a VALUES element that can express limits and ranges of the values that the corresponding cell can contain, such as minimum (MIN), maximum (MAX), or enumeration of possible values (OPTION).

4.1 FIELD attributes

The valid attributes of a FIELD or PARAM are:

- the name and/or ID (see section 3.2)
- the datatype, which expresses the nature of the data that is described as one of the permitted primitives (see Table 1 and their exact meaning in section 6). This attribute determines how data are read and stored internally; it is *required*, except when the ref attributes exists in which case the FIELD is just referenced (see section 4.7)
- the arraysize attribute exists when the corresponding table cell contains more than one of the specified datatype, as explained section 2.2. Note that strings are are not a primitive type, and have to be described as an array of characters.
- the width and precision attributes define the numerical accuracy associated to the data (see section 4.2)
- the unit attribute specifies the units in which the values of the corresponding column are expressed (see section 4.3)

- the ucd attribute supplies a standardized classification of the physical quantity expressed in the column (see section 4.4).
- the utype attribute, introduced in VOTable 1.1, is meant to express the role of the column in the context of an external data model (see section 4.5).
- the ref attribute defines the field as being a *reference* to a column having the referenced ID attribute. This attribute normally exists alone: if present, it precludes the existence of any other attribute except a utype attribute, and a value attribute for PARAM elements.
- The type is *not* part of this standard, but is reserved for future extensions. In Astrores it was used to express some pecularities of the column in the table as type="hidden" (see appendix B) and type="no_query" (see appendix B); an additional type="location" value is proposed to express columns containing parts of URIs (see appendix D). The type is not part of this standard, but is reserved for future extensions.

In addition, in the PARAM element only:

- the value attribute which exists only in the PARAM element; this attribute is moreover *required*, even when the PARAM contains the ref attribute.

4.2 Numerical Accuracy

The VOTable format is meant for transferring, storing, and processing tabular data, and is not intended for presentation purposes: therefore (in contrast to Astrores) we generally avoid giving rules on presentation, such as formatting. Inevitably however some at least of the data will have to be presented – either as actual tables, or in forms or graphs, etc... Two attributes were retained for this purpose:

- the width attribute is meant as a hint to the application about the number of characters to be used for input or output of the quantity.
- the precision attribute is meant to express the number of significant digits, either as a number of decimal places (e.g. precision="F2" or equivalently precision="2" to express 2 significant figures after the decimal point), or as a number of significant figures (e.g. precision="E5" indicates a relative precision of 10^{-5}).

The existence and presentation of the special *null* value of a field (when the actual value of the field is unknown) is another aspect of the numerical accuracy, which is part of the VALUES sub-element (see section 4.6).

4.3 Units

The quantities in a column of the table may be expressed in some physical unit, which is specified by the unit attribute of the FIELD. The syntax of the *unit* string is defined in reference [3]; it is basically written as a string without blanks or spaces, where the symbols . or * indicate a multiplication, / stands for the division, and no special symbol is required for a power. Examples are unit="m2" for m^2, unit="cm-2.s-1.keV-1" for $cm^{-2}s^{-1}keV^{-1}$, or unit="erg/s" for erg s^{-1}. The references [3] provides also the list of the valid symbols, which is essentially restricted to the *Système International* (SI) conventions, plus a few astronomical extensions concerning units used for time, angular, distance and energy measurements.

4.4 Unified Content Descriptors

The Unified Content Descriptors (UCD) can be viewed as a hierarchical glossary of the scientific meanings of the data contained in the astronomical tables. The initial version was created at CDS, but the UCD definition is currently evolving [4].
A few typical examples taken from the original UCD design:

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

4.5 The utype attribute

In some contexts, it can be important that FIELDs or PARAMeters are explicitely designed as being *the* parameter performing some well-defined role in some external data model. For instance, it might be important for an application to know that a given FIELD expresses *the* surface brightness processed by an explicit method. None of the existing name, ID or ucd attributes can fill this role, and the utype (usage-specific or *unique* type) attribute has been added in VOTable 1.1 to fill this gap.

In order to avoid name collisions, the data model identification should be introduced following the XML namespace conventions, as utype="*datamodel_identifier:role_identifier*".

4.6 VALUES element

The VALUES element of the FIELD is designed to hold subsidiary information about the *domain* of the data. For instance, in the example (section 3.1) we could rewrite the RA field definition as:

```
<FIELD name="RA" ID="col1" ucd="POS_EQ_RA_MAIN" ref="J2000" datatype="float"
       width="6" precision="2" unit="deg">
  <VALUES ID="RAdomain">
    <MIN value="0"/>
    <MAX value="360" inclusive="no"/>
  </VALUES>
</FIELD>
```

The VALUES element may contain MIN and MAX elements, and it may contain OPTION elements. The latter may itself contain more OPTION elements, so that a hierarchy of keyword-values pairs can be associated with each field.

All three MIN, MAX and OPTION sub-elements store their value corresponding to the minimum, maximum, or "special value" in a value attribute. MIN and MAX elements can have an inclusive attribute to specify whether the value quoted belongs or not to the domain, and the OPTION element can have a name attribute to qualify the "special" quoted value.

The VALUES element may also have a null attribute to define a non-standard value that is used to specify *"non-existent data"* – for example null="-32768". When this value is found in the corresponding 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. The default representation of a "null" value is an empty column in the TABLEDATA representation (i.e. <TD></TD>). For FITS and BINARY data, the *NaN* (not-a-number) patterns are recommended to represent floating-point "null" values. The "null" convention is therefore only necessary for primitive types that do not have a natural "null" value, such as int, short, byte, etc.

For fields containing arrays or complex numbers, the values specified in the value or null attributes have to be compatible with their datatype, and contain as many numbers separated by white space as implied by the datatype and arraysize of the parent FIELD.

The scope of the domain described by the VALUES element can be qualified by type="actual", if it is only applicable to the data enclosed in the parent TABLE. The domain of a valid RA in the example above has the default type="legal" qualification.

Finally the ref attribute of a VALUES element can be used to avoid a repetition of the domain definition, by referring to a previously defined VALUES element having the referenced ID attribute. When specified, the ref attribute defines completely the domain without any other element or attribute, as e.g. <VALUES ref="RAdomain"/>

4.7 GROUPing FIELDs and PARAMeters

The GROUP element was added in VOTable 1.1, to group together a set of FIELDS which are logically correlated, like a value and its error. Each field participating in a GROUP can be defined either *physically* (the FIELD contains a datatype field), or *logically* (the FIELD contains only a ref attribute referencing a field defined in the same parent TABLE). A physical field (i.e. a single column of the table) may therefore participate (logically) to several groups.

A straightforward example of a group, based on the example of section 3.1, can be to replace the definitions of columns 4 and 5 by the following:

```
<GROUP name="Velocity" ucd="VELOC_HC">
  <DESCRIPTION>Velocity and its error</DESCRIPTION>
  <FIELD name="RVel" ID="col4" ucd="VELOC_HC" datatype="float"
```

```
        width="5" unit="km/s"/>
  <FIELD name="e_RVel" ID="col5" ucd="ERROR" datatype="float"
        width="3" unit="km/s"/>
</GROUP>
```

A *logical* definition of this group could alternatively be achieved by inserting just before the DATA element the following:

```
<GROUP name="Velocity">
  <DESCRIPTION>Velocity and its error</DESCRIPTION>
  <FIELD ref="col4"/>
  <FIELD ref="col5"/>
</GROUP>
```

The GROUP element can have the name, ID, ucd, utype and ref attributes. It can include a DESCRIPTION, and any mixture of FIELDs, PARAMeters, and other GROUPs – this recursive grouping enabling a definition of arbitrary complex structures.

The possibility of adding PARAMeters in groups introduces also a possibility of associating parameter(s) to describe accurately the context of the data stored in the table: for instance, it is possible to associate the actual frequency of a radio survey with the following declaration:

```
<GROUP name="Flux" ucd="VELOC_HC">
  <DESCRIPTION>Flux measured at 352MHz</DESCRIPTION>
  <FIELD name="Flux" ucd="PHOT_FLUX_RADIO_400M" datatype="float"
        width="6" precision="1" unit="mJy"/>
  <PARAM name="Freq" ucd="OBS_FREQUENCY" unit="MHz" datatype="float" value="352"/>
  <FIELD name="e_Flux" ucd="ERROR" datatype="float" width="4"
        precision="1" unit="mJy"/>
</GROUP>
```

Similarly, the GROUP can be used to associate several parameters to one or several FIELDs: a filter may for instance be characterized by the central wavelength and the FWHM of its transmission curve; or several parameters of an instrument setup may be detailed.

5 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 having a datatype attribute, with each row having the same number of items as there are FIELD tags having a datatype attribute. Fields without a datatype attribute have a ref attribute, and represent references to "true" columns (see section 4).

Each DATA part of the VOTable document can be viewed as a stream coming out of a 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.

Figure 1 shows how the abstract table is rendered into the VOTable document. First the data is *serialized*, either as XML, a FITS binary table, or the VOTable 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.

The serialization elements and their attributes are described in the next sections.

5.1 TABLEDATA Serialization

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 — i.e. the same conventions as the familiar *HTML* ones. An example is contained in section 3.1, surrounded by in the <TABLEDATA> and </TABLEDATA> delimiters.

The number of TD elements should be in number equal to the number of FIELD elements having datatype attributes declaring the table; when there are less TD's than expected, the corresponding values are set to "null"s; superfluous TD's are ignored.

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.

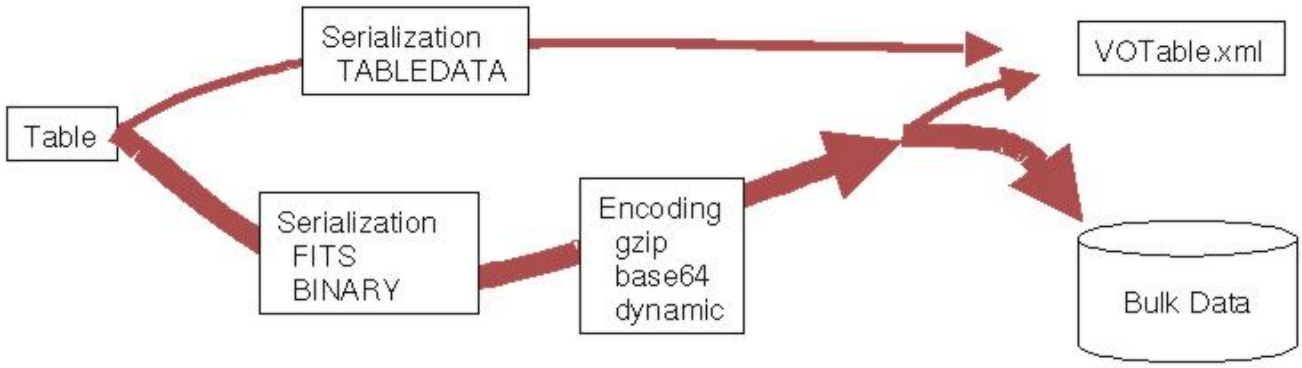


Figure 1: Data serialization

Valid representations of a number in a cell, depending on their `datatype`, are detailed in section 6.

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 two rows, that has arrays in the table cells:

```
<TABLE>
  <FIELD ID="aString" datatype="char" arraysize="10"/>
  <FIELD ID="Floats" datatype="float" arraysize="3"/>
  <FIELD ID="varComplex" datatype="floatComplex" arraysize="*"/>
  <DATA><TABLEDATA>
  <TR>
   <TD>Apple</TD><TD>1.62 4.56 3.44</TD>
   <TD>67 1.57  4 3.14  77 -1.57</TD>
  </TR><TR>
   <TD>Orange</TD><TD>2.33 4.66 9.53</TD>
   <TD>39 0  46 3.14</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 an array of three floats. The last cell contains a variable array of complex numbers, each complex number being represented by its real part followed by at least a blank and its imaginary part – hence 6 numbers for 3 complex numbers, or 4 numbers for 2 complex numbers.

5.2 FITS Serialization

The FITS format for binary tables [2] is in widespread in astronomy, and its structure has 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="float" value="1999.987">
  Original Epoch of the coordinates  </PARAM>
 <PARAM name="TELESCOP" datatype="char" arraysize="*" value="VTel" />
 <INFO name="HISTORY">
  The very first Virtual Telescope observation made in 2002
 </INFO>
 <TABLE>  <FIELD  (insert field metadata here) >
 <DATA><FITS extnum="2">
  <STREAM href="ftp://archive.cacr.caltech.edu/myfile.fit.gz"/>
 </FITS></DATA>
 </TABLE>
</RESOURCE>
```

The FITS file may contain many data objects (known as extensions, numbered from 1 up – the main header being numbered 0), and the `extnum` attribute allows the VOTable to point to one of these.

5.3 BINARY Serialization

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 bytes, the length of each sequence corresponding to the `datatype` and `arraysize` 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. The order of the bytes in multi-byte primitives (e.g. integers, floating-point numbers) is Most Significant Byte first, i.e. it follows the FITS convention.

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 (the sum of `arraysize` times the length in bytes of the corresponding `datatype`).

Variable-length arrays of primitives are preceded by a 4-byte integer containing the number of items of the array. The way the stream of bytes is arranged for the data of the example in section 5.1 is illustrated in Figure 2. The parser can then compute the number of bytes taken by the variable-length array by multiplying the size and number of the primitives.

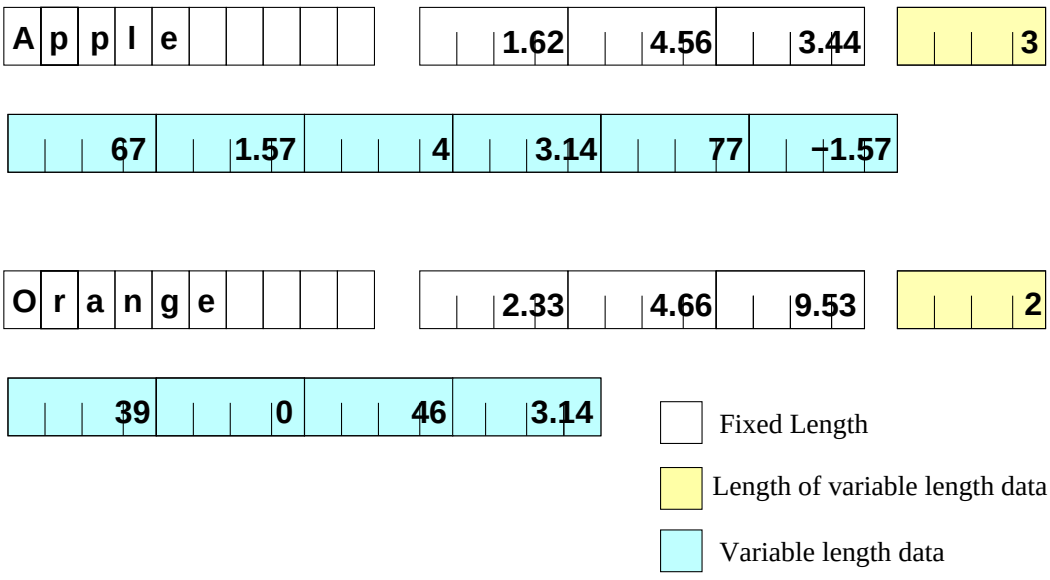


Figure 2: Data Storage in BINARY mode

5.4 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 the table is represented as XML tags directly embedded in the document, document, and conventional

tools can be used to encode the entire XML document. However, VOTable also provides limited encoding of its own. A VOTable document may point to a remote data resource that is compressed; rather than decompressing before sending on the wire, it can be dynamically decoded by the VOTable reader. We might also use the encoding facilities to convert a binary file to text (through base64 encoding), so that binary data can be used in the XML document.

In this version (1.1) of VOTable, it is not possible to encode individual columns of the table: the whole table must be encoded in the same way. The possibility of encoding selected table cells is however being examined for future versions of VOTable (see appendix E).

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 at the following values:

- `encoding="gzip"` [RFC1952] implies that the data following has been compressed with the *gzip* filter, so that *gunzip* or similar should be applied.
- `encoding="base64"` [RFC2045] implies that the *base64* filter has been applied, to convert binary to text.
- `encoding="dynamic"` implies 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 might 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.

5.5 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, as for instance:

```
<DATA><BINARY>
 <STREAM encoding="base64">
  AAAAAj/yVZiDGSSUwFZ6ypR4yGkADwAcQV0euAAIAAJBmMzNwZWZmkGle4tBR3jVQT9ocwAA
  ........................
 </STREAM>
</BINARY></DATA>
```

However, if the data is very large, it may be preferable to keep the data separate from the metadata. The `href` attribute of the `STREAM` element, if present, provides the location of the data in a URL-type syntax, for example:

```
<STREAM href="ftp://server.com/mydata.dat"/>
<STREAM href="ftp://server.com/mydata.dat" expires="2004-02-29T23:59:59"/>
<STREAM href="httpg://server.com/mydata.dat" actuate="onLoad"/>
<STREAM href="file:///usr/home/me/mydata.dat"/>
```

The examples are the well-known anonymous ftp, and http protocols. "httpg" is an example of a Grid-based access to data through httpg; "file" finally a reference to a local file. VOTable parsers are not required to understand arbitrary protocols, but are required to understand the three common protocols "file:", "http:" and "ftp:".

There are further attributes of the `STREAM` element that may be useful. The `expires` attribute indicates the expiration time of the data: this is useful when data are dynamically created and stored on some staging disk where files only persist for a specified lifetime and are then automatically deleted. The `expires` attribute expresses when a remote resource ceases to become valid, and is expressed in Universal Time in the same way as the FITS specification [2], itself conforming to ISO 8601 standard.

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).

6 Definitions of Primitive Datatypes

This section describes the primitives summarized in Table 1 and their representations in memory and in the `TABLEDATA` serialization (see section 5.1). In the following, the term "hexadigit" designates the ASCII numbers "0" to "9", or the ASCII lower- or upper-case letters "a" to "f" (i.e. a digit in an hexadecimal representation of a number).

- **Logical** If the value of the `datatype` attribute specifies data type "boolean", the contents of the field shall consist of ASCII "T", "t", or "1" indicating true or ASCII "F", "f", or "0" indicating false. The "null" value is indicated by a space (hexadecimal 20) or a question mark "?" (hexadecimal 3F). The acceptable representations in the `TABLEDATA` serialization include in addition any capitalisation variation of the strings "true" and "false" (e.g. "tRUe" or "FalsE"); the default representation of a *null* value is an empty cell (see section 4.6)
- **Bit Array** If the value of the `datatype` attribute specifies data type "bit", 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 field shall be composed of the smallest number of bytes that can accommodate the number of elements in the field. Padding bits shall be 0. The representation of a bit array in the `TABLEDATA` serialization is made by a sequence of ASCII "0" and "1" characters.
- **Byte** If the value of the `datatype` attribute specifies data type "unsignedByte", the field shall contain a byte (8-bits) representing a number in the range 0 to 255. In the case of an array of bytes (`arraysize="*"`), also known as a "blob", the bytes are stored consecutively. The representation of a Byte in the `TABLEDATA` serialization can be its *decimal* representation (a number between `0` and `255`) or its *hexadecimal* representation when starting by `0x` and followed by one or two hexadigits, (e.g. `0xff`), separated by at least one space from the next one in the case of an array of bytes; the default representation of a *null* value is an empty cell (see section 4.6)
- **Character** if the value of the `datatype` attribute specifies data type "char", the field shall contain an ASCII character. The `arraysize` attribute indicates a character string composed of ASCII text. Characters should be represented in the `TABLEDATA` serialization using the normal rules for encoding XML text: the ampersand (&) can be written `&` (symbolic representation) or `&#38` (decimal representation) or `&#x26` (hexadecimal representation); the less-than (<) and greater-then (>) symbols should be coded `<` and `>` or `&#3C` and `&#3D`; and a blank which would be interpreted by XML as whitespace (e.g. several consecutive blanks) should be coded ` ` or `&#x20`. The `BINARY` serialization follows the FITS rules, and a character string may be terminated by an ASCII NULL (hexadecimal code 00) before the length specified in the `arraysize` attribute: in this case characters after the first ASCII NULL are not defined; and a string having the number of characters identical to the `arraysize` value is not NULL terminated.
- **Unicode Character** If the value of the `datatype` attribute specifies data type "unicodeChar", the field shall contain a Unicode character. The `arraysize` attribute indicates a string composed of Unicode text. Each Unicode character is represented by two bytes, using the big-endian UCS-2 encoding (ISO-10646-UCS-2); no byte order mark should appear. This enables representation of text in many non-Latin alphabets. The representation of a Unicode character in the `TABLEDATA` serialization follows the XML specifications, e.g. the Cyrillic uppercase "Ya" can be written `&#x042F;` in UTF-8.
- **16-Bit Integer** If the value of the `datatype` attribute specifies datatype "short", 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. The representation of a Short Integer in the `TABLEDATA` serialization is either its decimal representation between -32768 and 32767 made of an optional `-` or `+` sign followed by digits, or its hexadecimal representation when starting by `0x` and followed by 1 to 4 hexadigits; the default representation of a *null* value is an empty cell (see section 4.6)
- **32-Bit Integer** If the value of the `datatype` attribute specifies datatype "int", the data in the field shall consist of twos-complement signed 32-bit integer, contained in four bytes. The most significant byte shall be first, and subsequent bytes shall be in order of decreasing significance. The representation of an Integer in the `TABLEDATA` serialization is either its decimal representation between -2147483648 and 2147483647 made of an optional `-` or `+` sign followed by digits, or its hexadecimal representation when starting by `0x` and followed by 1 to 8 hexadigits; the default representation of a *null* value is an empty cell (see section 4.6)
- **64-Bit Integer** If the value of the `datatype` attribute specifies datatype "long", 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. The representation of a Long Integer in the `TABLEDATA` serialization is either its decimal representation between -9223372036854775808 and 9223372036854775807 made of an optional `-` or `+` sign followed by digits, or its hexadecimal representation when starting by `0x` and followed by 1 to 16 hexadigits; the default representation of a *null* value is an empty cell (see section 4.6)
- **Single Precision Floating Point** If the value of the `datatype` attribute specifies datatype "float", the data in the field shall consist of ANSI/IEEE-754 32-bit floating point numbers. All IEEE special values are recognized. The IEEE NaN pattern is used to represent "null" values. The representation of a Floating Point number in the `TABLEDATA` serialization is made of an optional `-` or `+`, followed by the ASCII representation of a positive decimal number, and followed eventually by the ASCII letter "E" or "e" introducing the base-10 exponent made of an optional `-` or `+` followed by 1 or 2 digits. The number must be within the limits of the IEEE floating-point definition (around $\pm 3.4 \cdot 10^{38}$; numbers with absolute value less than about $1.4 \cdot 10^{-45}$ are equated to zero); the default representation of a *null* value is an empty cell (see section 4.6), and the special values "+Inf", "-Inf", and "NaN" are accepted.
- **Double Precision Floating Point** If the value of the `datatype` attribute specifies datatype "double", the data in the field shall consist of ANSI/IEEE-754 64-bit double precision floating point numbers. All IEEE special values

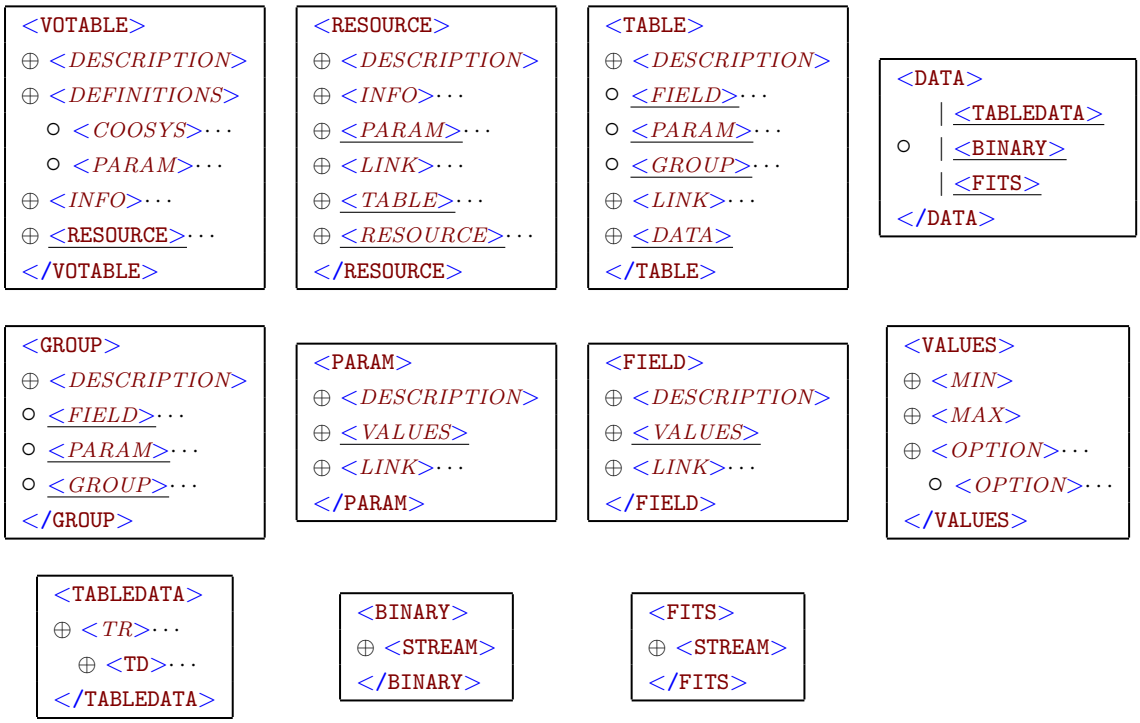
are recognized. The IEEE NaN pattern is used to represent "null" values. The representation of a Double number in the TABLEDATA serialization is made of an optional - or +, followed by the ASCII representation of a positive decimal number, and followed eventually by the ASCII letter "E" or "e" introducing the base-10 exponent made of an optional - or + followed by 1 or 2 digits. The number must be within the limits of the IEEE floating-point definition (around $\pm 1.7 \cdot 10^{308}$; numbers with absolute value less than about $5 \cdot 10^{-324}$ are equated to zero); the default representation of a *null* value is an empty cell (see section 4.6), and the special values "+Inf", "-Inf", and "NaN" are accepted.

- **Single Precision Complex** If the value of the datatype attribute specifies datatype "floatComplex", 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 "null". The representation of a Floating Complex number in the TABLEDATA serialization is made of two representations of a *Single Precision Floating Point* numbers separated by at least one blank, representing the real and imaginary part respectively; the default representation of a *null* value is an empty cell (see section 4.6)
- **Double Precision Complex** If the value of the datatype attribute specifies datatype "doubleComplex", 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 "null". The representation of a Double Complex number in the TABLEDATA serialization is made of two representations of a *Double Precision Floating Point* numbers separated by at least one blank, representing the real and imaginary part respectively; the default representation of a *null* value is an empty cell (see section 4.6)

7 A simplified view of the VOTable 1.1 Schema

The XML Schema [8] defining the VOTable document is available from
http://vizier.u-strasbg.fr/xml/VOTable-1.1.xsd

The illustration of the XML schema uses the following conventions: *italicized* text represents *optional* elements; ⊕ indicates that the order of the elements is mandatory, while the open bullet ○ indicates that the elements may occur in any order. The dots ··· indicate than an element may be repeated. The underlined elements are explained in a dedicated box.



8 Differences between versions 1.0 and 1.1

The differences between version 1.1 of VOTable and the preceding version 1.0 are:

the introduction of `GROUP` element (section 4.7)

the introduction of the `utype` attribute in the `FIELD` and `PARAM` elements (section 4.5)

generalisation of the description of a table as an unordered mixture of `FIELD`, `PARAM` and `GROUP` elements

`INFO` elements may exist in `TABLE` as well as `RESOURCE`

the `VALUE` element can have a `ref` attribute

9 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
See also Section 4 in Greisen and Calabretta 2002, A&A **395**, 1061; and the IAU Recommendations concerning Units from the *IAU Style Manual* by G.A. Wilkins (1989) available at http://www.iau.org/IAU/Activities/nomenclature/units.html

[4] *Unified Content Descriptors*
http://vizier.u-strasbg.fr/doc/UCD.htx (UCD1)
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaUCD

[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://xml.gsfc.nasa.gov/XDF/XDF_home.html

[8] *XML Schema: W3C Document*
http://www.w3.org/XML/Schema

Appendices

The definitions enclosed in these appendices are **not** part of VOTable 1.1, but are considered as candidates for VOTable improvements.

A VOTable LINK substitutions

The LINK *element in Astrores [1] contains a mechanism for string substitution, which is a powerful way of defining a link to external data which adapts to each record contained in the table* DATA*.*

When a LINK element appears within a RESOURCE or a TABLE element, extra functionality is implied. The href or gref attributes may not be a simple link, but instead a template for a link. If, in the example of section 3.1, we add the link

```
<LINK href="http://ivoa.net/lookup?Galaxy=${Name}&amp;RA=${RA}&amp;DE=${DE}"/>
```

a substitution filter is applied in the context of a particular row. For the first row of the table, the substitution would result in the URL

```
http://ivoa.net/lookup?Galaxy=N++224&RA=010.68&DE=%2b41.27
```

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 purpose of the link is defined by the content-role attribute. The allowed values are "query" (see section B), "hints" for information for use by the application, and "doc" for human-readable documentation.

The column names invoked in the pattern of the href attribute of the LINK element should exist in the document to generate meaningful links. In the common case where the VOTable was generated from a query of a database and contains only some of the columns in that database, it might be necessary to include columns additional to those requested in order to ensure that the LINKS in the VOTable are operational. Such a FIELD included "by necessity" is marked with by the attribute type="hidden". The primary key of a relational table is a typical example of a FIELD which would carry the type="hidden" attribute.

B VOTable Query Extension

The metadata part included in a RESOURCE *contains all the details necessary to create a* form *for querying the resource. The addition of a link having the* action *attribute can turn VOTable into a powerful query interface.*

In Astrores [1], the details on the input parameters available in queries are described by the PARAM and FIELD elements, and the syntax used to generate the actual query is described in the ASU [6] procotol: the FIELD or PARAM elements are paired in the form *name*=*value*, where *name* is the contents of the name attribute of a FIELD or PARAM, 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 quite similar to the HTML syntax used to describe a FORM.

A special type="no_query" attribute of the PARAM or FIELD elements marks the fields which are *not* part of the form, i.e. are ignored in the collection of *name*=*value* pairs.

The following is an example of a transformation of the VOTable in section 3.1 into a form interface:

```
<?xml version="1.0"?>
<VOTABLE version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://vizier.u-strasbg.fr/xml/VOTable.xsd">
  <DEFINITIONS>
  <COOSYS ID="J2000" equinox="2000." epoch="2000." system="eq_FK5"/>
  </DEFINITIONS>
  <RESOURCE name="myFavouriteGalaxies" type="meta">
    <PARAM name="-out.max" ucd="NUMBER" datatype="int" value="50">
      <DESCRIPTION>Maximal number of records to retrieve</DESCRIPTION>
    </PARAM>
    <LINK  content-role="query" action="myQuery?-source=myGalaxies&amp;" />
```

```
    <TABLE name="results">
      <DESCRIPTION>Velocities and Distance estimations</DESCRIPTION>
      <PARAM name="Epoch" datatype="float" ucd="TIME_EPOCH"
             value="2003.875/">
      <FIELD name="RA"   ID="col1" ucd="POS_EQ_RA_MAIN" ref="J2000" datatype="float"
             width="6" precision="2" unit="deg"/>
      <FIELD name="Dec"  ID="col2" "POS_EQ_DEC_MAIN" ref="J2000" datatype="float"
             width="6" precision="2" unit="deg"/>
      <FIELD name="Name" ID="col3" ucd="ID_MAIN" datatype="char" arraysize="8*"/>
      <FIELD name="RVel" ID="col4" ucd="VELOC_HC" datatype="int"
             width="5" unit="km/s"/>
      <FIELD name="e_RVel" ID="col5" ucd="ERROR" datatype="int"
             width="3" unit="km/s"/>
      <FIELD name="R" ID="col6" ucd="PHYS_DISTANCE_TRUE" datatype="float"
             width="4" precision="1" unit="Mpc">
        <DESCRIPTION>Distance of Galaxy, assuming H=75km/s/Mpc</DESCRIPTION>
      </FIELD>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

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. The PARAM with name="-out.max" has been added in this example to control the size of the result.

A valid query generated by this VOTable could be:

```
myQuery?-source=myGalaxies&-out.max=50&R=10..100
```

C Arrays of variable-length strings

Following the FITS conventions, strings are defined as arrays of characters. This definition raises problems for the definition of arrays of strings, which have then to be defined as 2D-arrays of characters – but in this case only the slowest-varying dimension (i.e. the number of strings) can be variable. This limitation becomes severe when a table column contains a set of remarks, each being made of a variable number of characters as it occurs in practice.

FITS invented the *Substring Array* convention (defined in an appendix, i.e. not officially approved) which defines a *separator* character used to denote the end of a string and the beginning of the next one. In this convention (*r*A:SSTR*w*/*ccc*) the total size of the character array is specified by *r*, *w* defines the maximum length of one string, and *ccc* defines the separator character as its ASCII equivalent value. The possible values for the separator includes the space and any printable character, but excludes the control characters.

Such arrays of variable-length strings are frequently useful e.g. to enumerate a list of properties of an observed source, each property being represented by a variable-length string. A convention similar to the FITS one could be introduced in VOTable in the arraysize attribute, using the **s** followed by the separator character; an example can be arraysize="100s," indicating a string made of up to 100 characters, where the comma is used to separate the elements of the array.

D FIELDs as data pointers

Rather than requiring that all data described in the set of FIELDs are contained in a single stream which follows the metadata part, it would be possible to let the FIELD act as a *pointer* to the actual data, either in the form of a URI or of a reference to a component of a multipart document.

Each component of the data described by a FIELD may effectively have different requirements: while text data or small lists of numbers are quite efficiently represented in pure XML, long lists like spectra or images generate poor performances if these are converted to XML. The method available to gain efficiency is to use a binary representation of the *whole data stream* by means of the STREAM element – at the price of delivering data in a totally non-human readable format.

The following options would allow more flexibility in the way the various FIELDs can be accessed:

- a FIELD can be declared as being a *pointer* with the addition of a type="location" value, meaning that the field contains a way to access the data, and not the actual data;
- a FIELD can contain a LINK element marked type="location" which contains in its href attribute the partial URI to which the contents of the column cell is appended in order to generate a fully qualified URI.

Note that the LINK is not required – a FIELD declared with type="location" and containing no LINK element is assumed to contain URIs.

An example of a table describing a set of spectra could look like the following:

```
<TABLE name="SpectroLog">
  <FIELD name="Target" ucd="ID_TARGET" datatype="char" arraysize="30*"/>
  <FIELD name="Instr" ucd="INST_SETUP" datatype="char" arraysize="5*"/>
  <FIELD name="Dur" ucd="TIME_EXPTIME" datatype="int" width="5" unit="s"/>
  <FIELD name="Spectrum" ucd="DATA_LINK" datatype="float" arraysize="*"
         unit="mW/m2/nm" type="location">
    <DESCRIPTION>Spectrum absolutely calibrated</DESCRIPTION>
    <LINK type="location"
        href="http://ivoa.spectr/server?obsno="/>
  </FIELD>
  <DATA><TABLEDATA>
    <TR><TD>NGC6543</TD><TD>SWS06</TD><TD>2028</TD><TD>01301903</TD></TR>
    <TR><TD>NGC6543</TD><TD>SWS07</TD><TD>2544</TD><TD>01302004</TD></TR>
  </TABLEDATA></DATA>
</TABLE>
```

The reading program has therefore to retrieve the data for this first row by resolving the URI

```
http://ivoa.spectr/server?obsno=01301903
```

The same method could also be immediately applicable to *Content-ID*s which designate elements of a multipart message, using the protocol prefix `cid:` [RFC2111]

Note that the *VOTable LINK substitution* proposed in section A fills a similar functionality: generate a pointer which can incorporate in its address components from the DATA part for the VOTable.

E Encoding individual table cells

Accessing binary data improves quite significantly the efficiency both in storage and CPU usage, especially when one compares with the XML-encoded data stream. But binary data cannot be included in the same stream as the metadata description, unless a dedicated coding filter is applied which converts the binary data into an ASCII representation. The base64 is the most used filter which does this conversion, where 3 bytes of data are coded as 4 ASCII characters, which implies an overhead of 33% in storage, and some (small) computing time necessary for the reverse transformation.

In order to keep the full VOTable document in a unique stream, VOTable 1.0 introduced the encoding attribute in the STREAM element, meaning that the data, stored as binary records, are converted into some ASCII representation compatible with the XML definitions. One drawback of this method is that the entire data contents become non human-readable.

The addition of the encoding attribute in the TD element allows the data server to decide, at the cell level, whether it is more efficient to distribute the data as binary-encoded or as edited values. The result may look like the following:

```
<TABLE name="SpectroLog">
  <FIELD name="Target" ucd="ID_TARGET" datatype="char" arraysize="30*"/>
  <FIELD name="Instr" ucd="INST_SETUP" datatype="char" arraysize="5*"/>
  <FIELD name="Dur" ucd="TIME_EXPTIME" datatype="int" width="5" unit="s"/>
  <FIELD name="Spectrum" ucd="SPECT_FLUX_VALUE" datatype="float" arraysize="*"
         unit="mW/m2/nm" precision="E3"/>
  <DATA><TABLEDATA>
    <TR><TD>NGC6543</TD><TD>SWS06</TD><TD>2028</TD><TD encoding="base64">
    QJKPXECHvndAgMScQHul4OCSLQ5ArocrQLxiTkC3XClAqOOWQKQIMUCblYFAh753QGij10BT
    Em9ARKwIQExqf0BqbphAieuFQJSOOUCJWBBAhcrBQJMzMOCmRaJAuRaHQLWZmkCyhytAunbJ
    QLN87kC26XlA1KwIQOu+d0DsWh1A5an8QNOm6UDOVgRAxO2RQM9Lx0Din75A3o9cQMPfOOC/
    dLxAvUeuQKN87kCXQ5ZAjFodQH0vG0B/jVBAgaHLQI7Ag0CiyLRAqBBiQLaXjUDYcrBA8p++
    QPcKPUDg7ZFAwcKPQLafvkDDlYFA1T99QM2BBkCs3S9AjLxqQISDEkCO6XlAmlYEQKibpkC5
    wo9AvKPXQLGBBkCs9cNAuGp/QL0euEC4crBAuR64QL6PXEDOTdNA2987QN9T+EDoMSdA8mZm
    QOZumEDDZFpAmmZmQGlYEEBa4UhAivGqQLel4ODgan9A4WBCQLNcKUCIKPZAk1P4QNWRaEEP
    kWhBKaHLQTkOVkFEan9BUWBCQVyfvg==
    </TD></TR>
  </TABLEDATA></DATA>
</TABLE>
```

When decoded, the contents of the last column is the binary representation of the spectrum, as defined in section 5.3; no length prefix is required here, the total length of the array being implicitely defined by the length of the encoded text.

F Additional TABLE attributes

The GROUP introduced in version 1.1 can be qualified by ucd and utype attributes. These attributes could similarly bring useful additional metadata to qualify the TABLE structure.

G A new XMLDATA serialization

In order to facilitate the usage of the standard XML query tools which are easier to handle when each parameter has its individual tag, the XMLDATA serialization introduces the designation of each FIELD by a dedicated tag. An example could look like the following:

```
<TABLE name="Messier">
  <FIELD name="M" ID="ID" ucd="ID_NUMBER" datatype="int" >
    <DESCRIPTION>Messier Number</DESCRIPTION>
  </FIELD>
  <FIELD name="R.A.2000" ID="RA" ucd="POS_EQ_RA_MAIN" ref="J2000" unit="deg"
     datatype="float" width="5" precision="1" />
  <FIELD name="Dec.2000" ID="DE" ucd="POS_EQ_DEC_MAIN" ref="J2000" unit="deg"
     datatype="float" width="5" precision="1" />
  <FIELD name="Name" ID="N" ucd="ID_ALTERNATIVE" datatype="char" arraysize="*">
    <DESCRIPTION>Common name used to designate the Messier object</DESCRIPTION>
  </FIELD>
  <FIELD ID="T" name="Classification" datatype="char" arraysize="10*"
     ucd="CLASS_OBJECT">
     <DESCRIPTION>Classification (galaxy, glubular cluster, etc)</DESCRIPTION>
  </FIELD>
  <DATA><XMLDATA>
    <TR>
      <M>3</M>
      <RA>205.5</RA>
      <DE>+28.4</DE>
      <N/>
      <T>Globular Cluster</T>
    </TR>
    <TR>
      <M>31</M>
      <RA>010.7</RA>
      <DE>+41.3</DE>
      <N>Andromeda Galaxy</N>
      <T>Galaxy</T>
    </TR>
  </XMLDATA></DATA>
</TABLE>
```

The full document would need an XML-Schema definition of the tags M, RA, DE, N and T; these being derived directly from the ID attribute of the FIELD element, their definition can be generated automatically from the set of FIELD definitions.