The GGobi XML Input Format

Duncan Temple Lang
Deborah F. Swayne

November 9, 2026

1 The Advantages of XML

GGobi's XML format allows a rich variety of data attributes and relationships to be specified in a single file, including

- missing values,
- how missing values are encoded (for the entire dataset or per variable),
- how many records there are,
- levels of a variable that are not observed but known to exist (e.g. ethnicities not encountered in a survey),
- the source of the data,
- the type of each variable (e.g. factor or numeric),
- graph topology,
- symbol types, sizes and colors (for the entire dataset or per observation).

There is no doubt that the XML format is verbose. However, its copious markup and rigid structure offer many compensating benefits to authors of the input files as well as to application programmers. For example, XML files can be validated externally: in other words, a well-formed file can be tested outside of the application for which it serves as input, which greatly helps in preparing and maintaining correct input files. XML parsers check whether the document is well-formed, that all obligatory sections are present, and that all sections are correctly placed. Identifiers can be specified for each row and validating parsers can check that they are unique.

To validate a ggobi data file, execute

```
xmllint -noout -dtdvalid ggobi.dtd flea.xml
```

The growing use of XML means that its structure is now familiar to many people, and there are editors and browsers to create and view XML files. Given the ability of R, S and Omegahat (and an increasing number of other statistical applications) to read XML, the dataset can be used in other applications with little or no additional code.

Additionally, it is easy to define new DTDs to represent different inputs such as property or resource files, descriptions of plots, layout specifications for multiple plots, graph descriptions, etc. This can leverage much of the same parsing setup and importantly provides a uniform and increasingly familiar interface for the user for specifying files.

XML offers support for reading compressed files. The XML parser we employ (Daniel Veillard's libxml) can parse XML directly from compressed files with very little speed penalty. You can try this feature by using GNU zip (gzip) to compress the file flea.xml in the **data/** directory and starting the ggobi application

```
ggobi data/flea.xml.gz
```

The parser automatically determines whether the file is compressed or not.

GGobi xml files allow one to specify default attributes (e.g., symbol type, size and color) for all records in the file, and to override those defaults for a single record at a time. This greatly simplifies experimenting with different parameter values.

The fact that the number of records and variables are specified in the file format means that only one pass of the file is needed to read the data. Additionally, it is easier to handle non-rectangular data, which may occur when data are sparse or when there is a variable number of values per observation (e.g in medical studies).

2 The File Format

The format of the file is described by the DTD (Document Type Definition) `ggobi.dtd`, though it may be easier to learn about the file format by looking at the examples in the data directory. Each file starts with the usual XML declarations that identify it as XML (and its version) and the particular document type and associated DTD.

```
<?xml version="1.0"?>
<!DOCTYPE ggobidata SYSTEM "ggobi.dtd">

<ggobidata>
```

The string *ggobidata* indicates that this is the top-level tag for the document, and this is what appears next. To specify that more than one dataset is included, use the *count* attribute:

```
<ggobidata count="2">
```

This tag must be terminated at the end of the file:

```
</ggobidata>
```

2.1 Data

This is followed by the *data* tag, which begins the entries for a dataset:

```
<data name="Flea beetles">
```

Here you can also specify the name which will appear in the titlebars of ggobi windows.

There can be multiple datasets within a file, and there can be two types of relationships among their elements:

- records in multiple datasets can represent different variables recorded for the same subject, as described in section 2.5.1, or
- one dataset can contain a description of edges which connect points in another, as described in section 2.5.2.

Both of these schemes depend on the *record id* to uniquely identify a record.

The remainder of the dataset is specified as sub-elements or sub-tags within this *data* element.

2.2 Color schemes

Included in the source code is a file called colorschemes.xml, which contains (as of this writing) 265 distinct color schemes.

To specify one that one of these schemes should be the default for a particular set of data, specify it inside the ggobidata element.

```
<ggobidata>
<activeColorScheme name="YlOrRd 9"/>
...
</ggobidata>
```

In case you have devised your own scheme you'd like to use, specify it as follows:

```
<activeColorScheme file="/full/path/name/mycolorschemes.xml"
                   name="MyColorScheme 7"/>
```

2.3 Description

The second of the sub-elements within the *data* tag is a description of the dataset.

```
<description>
Physical measurements on flea beetles.
</description>
```

This includes the source, any references, etc. This is currently free-format. A convenient attribute is *source* which indicates where it can be found.

2.4 Variables

The next section of the file contains the descriptions of the variables. It begins with the *variables* tag, which must include the number of variables:

```
<variables count="3">
</variables
```

Between the *variables* tags, the file lists the variables, which can be continuous (real or integer) or categorical. Continuous real variables are specified simply:

```
<realvariable>
 <name> tars1 </name>
</realvariable>
```

Integer variables use the tag `integervariable`. For categorical variables, some correspondance must be established between level values and data values, since GGobi stores and manipulates numbers. (These levels can also serve as a basis for linking records during brushing as described in 2.5.1.) The simplest way to specify the variable is:

```
  <categoricalvariable name="SEX" levels="auto" />
  <categoricalvariable name="SMOKER" levels="auto" />
```

In that case the record values must be individually tagged as strings, and all the rest of the values in the record should also be individually tagged. This is a sample record from tips.xml:

```
<record label="1">
 <real>1</real> <real>16.99</real> <real>1.01</real> <string>F</string>
 <string>no</string> <real>6</real> <real>1</real> <real>2</real>
</record>
```

If your data values are numbers instead of strings, and all levels of interest are present in the data, a convenient alternative is to let ggobi assign the levels:

```
<categoricalvariable name="fraudp">
  <levels count="3" />
</categoricalvariable>
```

By default, the level values in this case will be 1, 2, and 3, and the level names will be L1, L2 and L3. If you want level values that begin at some number other than 1, specify the variable range:

```
<categoricalvariable name="fraudp" min="0" max="2">
  <levels count="3" />
</categoricalvariable>
```

These methods are appropriate when the levels of the categorical variable have no natural ordering, and when all levels of interest are known to be present in the data. When you want all gaps to be filled in, and you want complete control over the correspondance between value and level name, a fully explicit specification can be used to establish the correspondance between levels and values:

```
<categoricalvariable name="fraudp">
  <levels count="3">
    <level value="0">low</level>
    <level value="1">medium</level>
    <level value="2">high</level>
  </levels>
</categoricalvariable>
```

For another example, see Shipman.xml. The levels of month are fully specified, because we want the correspondance between month name and number to be properly established; the levels of place and cause can be automatically assigned because no such restriction exists.

Two other variable types exist: `countervariable` `randomuniformvariable`. For these two variables, no data should be specified; they will be automatically populated when ggobi starts, the first with integers from 1 to N, the second with random real numbers on [0, 1].

The name of the variable can be specified as the text within the variable tag rather than as an attribute, and an optional "nickname" can be added, to be used for labelling variables in tight spaces:

```
<realvariable name="tars1" nickname="t1" />
```

(By default, the nickname is the first two letters of a variable, and that isn't always enough to make distinctions.)

Variable ranges can be specified for real variables as well:

```
<realvariable name="Proportion"  min="0.0"  max="1.0" />
```

2.5 Records

The next section of the file is the data itself. The individual `record` tags are contained within the `records` element, which must include the `count` attribute, specifying the number of records in the data.

```
<records count="74" color="2" glyphType="3" glyphSize="3" missingValue=".">
</records>
```

It may optionally include tags specifying the default color (the index in the color table), glyph type and size, or the character to interpret as a missing value.

Glyph type and size can also be specified in a single string:

```
glyph="fc 3"
```

GlyphSize can range from 0 to 7; glyphType from 0 to 6, with 0 representing a single-pixel point and 6 a filled circle, and the rest as shown in the table below. Legal values for the first string in the glyph tag are described in this table:

glyphType	glyph specifier	resulting glyph
0	.	point
1	plus	glyph resembling a "+"
2	x	glyph resembling an "x"
3	or	open rectangle
4	fr	filled rectangle
5	oc	open circle
6	fc	filled circle

The body or content of each `record` may take two forms:

- An ASCII listing of the values, with each value separated by white space (space character, tabs or new lines). (See flea.xml.)
- A fully specified listing of the values, with each value tagged as real, int, or string. (See tips.xml.) For the file to be valid, all values must be tagged if any one is. GGobi will read the file without complaint, but Rggobi will balk and the file will not validate.

A record can be considered "hidden". This is set via a logical value for the attribute `hidden`.

2.5.1 Linking Records

Each record can be given an *id* attribute value, which can be an arbitrary string. This is different from a label in that it is not used by ggobi instance when displaying plots. Instead, it is used only to uniquely identify a record within a dataset, and it has two purposes.

The *id* can be used in the case where different datasets contain different variables for the same subjects, or for some of the same subjects. For instance, dataset A may contain usage data for a set of customers, while dataset B contains demographic data for a subset of those customers. Those datasets will be linked for brushing and identification if they have the same value for *id*.

The levels of a categorical variable can also be used as the basis for linking records during brushing. If records are linked by a categorical variable (see the Brush control panel), then brushing one observation causes all observations in all displays with the same level for that variable to be painted as well.

2.5.2 Edges

The *id* is also a critical part of the specification of edges. In order to specify a set of edges, or line segments, between pairs of points in a dataset, it's necessary to define a second dataset whose records have *source* and *destination* tags, like this:

```
<record source="a" destination="b"> </record>
```

The values used for *source* and *destination* must correspond to the *id*s specified elsewhere.

A record which includes this edge specification can also include any other attribute or property of a record:

```
<record source="0" destination="2" color="3"> 4.2 6 9.6 </record>
```

If other data values are present, this dataset is like any other dataset, and the values can be displayed in scatterplots, parallel coordinate plots, and so on.

3 Conversion of Old Files

The distribution contains an application named xmlConvert that can be used to read datasets provided in the old file format to XML. This can be used by specifying the name of the file containing the old-style data in the same manner as ggobi expects. The output is written to standard output and can be redirected to a file using basic shell commands. For example,

```
xmlConvert data/flea > flea.xml
```

In the future, we will support writing the output to a file. (We need to process the command line arguments and look for a -o flag).

Note that this dynamically loads the libraries libGGobi.so and libxml.so. Thus the directories that contain these libraries must be referenced in the environment variable `LD_LIBRARY_PATH`. Alternatively, the makefile can be edited to statically link these libraries.

4 References

The XML Handbook, Charles F Goldfarb and Paul Prescod, Prentice Hall.

http://www.w3.org/XML