

The GGobi XML Input Format

Duncan Temple Lang
Debby Swayne

November 27, 2002

1 Using XML Input Formats

There is now support for reading the data from an XML file. This is an alternative to having the inputs for a dataset be contained in a collection of multiple but associated files that provide the different characteristics such as

- data
- observation labels
- glyph information
- color
- segments between points.

Instead, the XML format allows all the information to be specified in a single file. Additionally, different sections of information can be omitted.

One of the advantages of XML is that it can be validated externally. In other words, a well-formed file can be tested outside of the application for which it serves as input. This helps to prepare and maintain correct input 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. (Of course, `read.table()` and associated functions in R/S makes reading the old file formats easy.)

XML parsers can check whether the document is well-formed. This means that all obligatory sections are present, that sections are in the correct place. Additionally, identifiers can be specified for each row and validating parsers can check that they are unique.

Additionally, more useful information can be added to the data source. This includes items such as

- 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
- tooltips for use in describing the data
- the type of each variable (e.g. factor or numeric)

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 and no reallocation is needed as more observations than expected are located. Additionally, it is easier to handle non-rectangular data. For example, sparse data and variable number of values per observational unit (e.g in medical studies).

Additionally, the growing usage of XML means that there are editors and browser to create and view XML files.

There is no doubt that the XML format appears more verbose and indeed it is. However, its more rigid structure benefits the authors of the input files as well as the application programmers. It is more convenient and significantly less error-prone to have related information be in the same location. For example, specifying the color of a line segment connecting two points in one file and the actual connection in another means that one has to have a mechanism to link the two specifications. Frequently this is a simple order in which they appear in the different files. Removing individual lines of information commonly leads to lengthy searches for why the input does not produce the desired result. Often this is because of an “off-by-one” connection.

Note that the XML approach will also be used to generate and potentially read plot descriptions for persistence and exchange with other software systems.

A final benefit to using XML is that we have support for reading compressed files. The XML parser we employ (Daniel Veillard’s libxml) can parse XML directly from compressed files. For large datasets, this is convenient as we don’t have to uncompress the files before using them. 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 -x data/flea
```

This searches for a file named `data/flea.xml` and if this is not found, `data/flea.xml.gz` and then `data/flea.xmlz`. The parser automatically determines whether it is compressed or not. This support can be turned off. Some simple tests illustrate that the XML representation can be about 50% larger than their simpler ASCII equivalents (i.e. without the markup) but that the compression brings them to about 1/3 the size. The speed at which the compressed file is read is almost the same as the uncompressed XML, and is about 33% longer than the ASCII equivalents. (These were done on a 100000×5 matrix. Note also that all the color and glyph information was included in the XML and not provided in the ASCII, so the XML looks slightly better than the above numbers suggest.)

Note also that the XML parsing library has support for reading files via ftp and http.

```
ggobi -x http://www.ggobi.org/data/flea.xml
```

Being able to specify file-specific defaults allows one to easily change the characteristics of the plots without excessive editing of the contents of the file. For example, to use a different glyph type, we need only specify a different value for the `glyphType` attribute in the `records` tag. This greatly simplifies experimenting with different parameter values.

In contrast with the multiple input files for each dataset used by xgobi, the XML approach reduces the number of files to 1. This means that it is easier to distribute inputs to others. There is no need to send multiple files individually, or to combine them into an archive, etc.

An advantage of XML over a simpler but more specialized format is that people are somewhat familiar with the basic rules of HTML, and hence XML. Additionally, it is easy to define new DTDs to represent different inputs such as property or resource files, descriptions of plots (see `SVG/`), 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.

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.

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="Yl0rRd 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 or categorical. Continuous variables are specified simply:

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

For categorical variables, some correspondance must be established between level values and data values, since the data values in `record` tag are numbers. (These levels can also serve as a basis for linking records during brushing as described in 2.5.1.) The most explicit method for setting up this correspondance is to fully specify level values and names:

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

A convenient alternative when there are very many levels is to let ggobi assign them:

```
<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>
```

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" />
```

Additionally, instructions as to how to create the variable can be specified as a programming command via the Programming Instruction (PI)

```
<realvariable>
<?R rnorm(10)>
</realvariable>
```

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** is a simply ASCII listing of the values. Each value is separated by white space (space character, tabs or new lines).

A record can be considered “hidden”. This is set via a logical value for the attribute **hidden**.

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.

2.5.1 Linking Records

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 *Link by variable* is selected, and a categorical variable is highlighted in the **Variable manipulation** tool, then brushing one observation causes all observations in all displays with the same level for that variable to be brushed 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 ids 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 Using XML Files

To use data that is contained in an XML file, invoke ggobi with no command line flag, or with the command line flag `-x`.

In the future, ggobi could be smart enough to include the detection of the XML format.

4 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.

5 Compilation

To activate the XML mechanism, define the variable `USE_XML` in `local.config`.

When we use `autoconf`, this can be done by

```
./configure --with-xml
```

This requires the XML parsing library `libxml` (also known as `gnome-xml`) by Daniel Veillard. (Daniel.Veillard@w3.org). See <http://xmlsoft.org>

6 References

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

<http://www.w3.org/XML>