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
- connected segments.

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 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 characterstics 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 `ggobidata` 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 omre 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`

The 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">
```

The string ggobidata indicates that this is the top-level tag for the document, and this is what appears next. The attributes of this tag specify the number of records in the dataset. The identifier for a missing value can also be specified as an attribute (`missingValue`). (We might allow this to be overridden for each variable.)

Also, default values for different characteristics of the observations can be specified here. For example, the glyph type and size, the color of an observation, etc. can all be specified here. These are applied to the individual observations for which they are not specified. So, such an entry might look like

```
<ggobidata numRecords="" color="2" glyphType="fc" glyphSize="3">
```

The remainder of the dataset is specified as sub-elements or sub-tags within this `ggobidata` element. Thus, there must be a termination of this `ggobidata` at the end of the file.

2.1 Colormap

The user can specify rows of a color table in RGB format. (Perhaps different formats such HSV, etc. can be supported directly by GTK also.) The idea is that colors for records, etc. are specified as row identifiers for this table. The colormap section allows the user to specify the values for these rows. In this way, the data and color references can remain fixed and one need only change the contents of the colormap to which the correspond.

It is often convenient to use the same colormap for several datasets. Rather than encoding the data in each input file, the XML input file can be told to read color data from an external file. This is specified via the file and type The value of the file attribute identifies a URL or file. If not an absolute file name or URI, then this is located relative to the location of the document being currently read. That is, suppose we are running in the top-level distribution directory of ggobi and run with the input file **data/flea.xml**. Then, a reference to

```
<colormap file="stdColorMap.xml">
```

is found as **data/stdColorMap.xml**. Similalry, if a colormap file named **map.xml** is located in the directory **color/** parallel to **data/**, then

```
<colormap file="../color/map.xml">
```

would find it for the same input file **data/flea.xml**.

When an external colormap file is used in conjunction with local values, there are two size specifications. One is in the colormap of the external file and the other is local. Both refer to the number of local entries, i.e. entries under their control. By default, these are accumulated so that the size of the table is the sum of the two sizes. This can lead to oversized tables.

External color map files can be formatted using XML (see **data/stdColorMap.xml**) or as a simple rectangular array. In the latter case, each row should contain 3 values separated by white space.

To differentiate between the two formats, the type is used when specifying an external color map file to read. This can be either `xml` or `ascii`. (See **ggobi.dtd**)

Each entry can have an identifier attribute. This is usually an integer identifying the index by which one can reference this color in records and other files. The alternative values for the identifier attribute are "fg" or "bg".

The identifier is really only used to override other values read from a file. This is due to the fact that subsequent entries without identifiers occupy the next entry in the color table. In other words, the input

```
<color id="4" r=".5" g=".5" b="0" />
<color>0 0 1</color>
```

sets the 5 entry (in position 4) to blue (001) since the previous set entry was indexed explicitly as 4.

The attribute range value can be specified for the entire colormap or on a per-entry basis. If this is present, it is interpreted as a numeric value and each value in the entry (or all entries if specified for the entire colormap) is divided by that amount. This allows one to easily use different scales such as 0 to 1 or 0 to 100, etc. This is designed to assist when the software creating the file does not facilitate such rescaling.

3 Description

The second of the sub-elements is a description of the datasets. This includes the source, any references, etc. This is currently free-format. A convenient attribute is `source` which indicates where it can be found.

3.1 Variables

The next element of the file lists the variables.

```
<variables>
<variable name="A" group="1" />
<variable name="B" group="1" />
</variables
```

The name of the variable can be specified as the text within the variable tag rather than as an attribute.

The name of the transformed variable can be specified via the attribute `transformedName`.

The group attribute allows variables to be joined for the same purpose as ifentified by the .vgroups file in the old format. Variables within the same group are scaled "jointly".

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

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

3.2 Records

The next section of the file is the data itself. The individual `record` tags are contained within the `records` element. 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. This is different from a label in that it is not used by the ggobid instance when displaying plots. Instead, it is used only when reading the line segment information that connects different points. (See below). The id is a symbolic name or tag used to uniquely identify a record. These are mainted by the parser and can be used as alternatives to row numbers when specifying the source and destination records in a line segment.

3.3 Segments

Connections between points can be specified via the segments tag. Like records, the individual segment entries are contained within a segments entry. Also, the latter must specify the size attribute to identify how many entries to expect.

Each line segment can have characteristics such as line color, width, whether it is hidden or not, etc. Default values for these characteristics can be set via the attributes of the segments tag.

Each individual segment must have a source and destination attribute. The value for each of these is a record identifier. This can be simply the row number (starting at 1). Alternatively, as explained above, this can be the value of an id assigned to a record tag earlier in the input.

The different appearance characteristics of the line are specified via the attributes width, color, type and hidden

4 Using XML Files

To use data that is contained in an XML file, invoke ggobi with the command line flag `-x`.

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

5 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.

6 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 libray libxml (also know as gnome-xml) by Daniel Veillard. (`Daniel.Veillard@w3.org`) See http://xmlsoft.org

7 References

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

http://www.w3.org/XML