standard would be widely usable, just as such assurances are important for those programming in languages, such as C and C++, for which an agreed standard is available. Ongoing work in the Omegahat project will, we hope, encourage discussion of such a standard.

Summary

We have described very briefly some ongoing work, part of the Omegahat software, that provides facilities to programmers in R and S-Plus. Using these, you can connect software in the S language in an effective way to many other computing tools. Wherever possible, the resulting software should be directly usable in both R and S-Plus.

We encourage readers to get more information from the Omegahat website (http://www.omegahat.org/), and to try out the components that seem interesting for your applications. Because Omegahat is a joint, open-source project, we also particularly encourage suggestions and contributions to any of these efforts.

John M. Chambers
Bell Labs, Murray Hill, NJ, U.S.A.
jmc@research.bell-labs.com

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Using XML for Statistics: The XML Package

by Duncan Temple Lang

XML, the eXtensible Markup Language is fast becoming the *next big thing* in computer applications and the Web. Along with the usual hype, there really is some substance and XML will probably turn out to have non-trivial relevance for statistics. In anticipation of this, we in the Omegahat project (http://www.omegahat.org/) have added an S package (i.e., works with both R and S-Plus) to both read and write XML. In this short article, we will outline some of the problems for which XML is a natural solution, and then describe what XML is, and some other areas in which XML technology is being exploited. Then we will take a brief look at some of the aspects of the R/S-Plus **XML** package. More information on the functions can be found at http://www.omegahat.org/RSXML/.

Using XML for datasets

Many of us are familiar with receiving the contents or values of a dataset in one file and a description of the dataset and its format in another file. Another problem is when the dataset is given as a set of files where records in one file correspond to records in the other files. In either case, the user must figure out the exact format and appropriate commands to group the values so as to read the values into into the data analysis environment. For example, the user must know whether some of the (initial) lines are comments; whether the first row of data contains variable names or is an actual record; how are missing values represented; whether integer values in a column represent a factor, a real valued-variable, or actually an integer; etc. The separation of the auxiliary information such as the ones just listed and other details such the author, the version of the dataset, the contact address, etc. can make it harder to process and ensure that the data has been read correctly.

When the datasets are not simply tables of numbers or strings, things quickly get more complicated. For example, if each record contains a different number of observations for a particular variable (i.e., ragged rows), some representation must be encoded into the ASCII file indicating which values are associated with which variable. Similarly, if the data frame of interest contains actual S objects that are not merely simple scalars, but are made up of a collection of values (e.g., the first two values of a record define a time interval object), we need to process these in a separate step after reading all the values using `read.table`.

If we could provide structure and additional information to data, we could also combine different, related datasets. For example, we can have both data about computers in a network and also the topology of that network in different parts of the same file. Also, we could add auxiliary information such as the number of records, the number of elements in a list, etc. This makes it easier and more efficient to read into S.

In order to address these issues, we need to find a way to add this type of structural information to the values. To do this, we need to agree on a format and then develop tools to read and write data using this format. And we need to develop these tools for different systems such as R, S-Plus, Matlab, SAS, etc. Well this seems like a lot of work, so we should look for an existing format that allows us to add our own

types of information and which has existing tools to read and write data in this format. And this is exactly what XML is good for.

What is XML?

XML stands for eXtensible Markup Language. It is a general version of the familiar HTML. XML is "eXtensible" in the following way. While HTML has a fixed set of elements such as <H1>, <H2>, <TABLE>, <TR>, <A>, etc., XML allows one to define new tags or elements and the allowable attributes they support. In its simplest terms, XML is a general way of describing hierarchical data, i.e., trees. Nodes are represented as names with named attributes and subnodes, i.e.

```
<nodeName name="value" name="value">
  <subNode>...</subNode>
</nodeName>
```

How can we represent a dataset as a tree using XML? Simply by having the different elements such as row and variable names, the different records and the values within each record be introduced by XML tags. For example, we might markup the Motor Trend Car Data (mtcars) dataset in the following way:

```
<dataset numRecords="32">
  <variables number="11">
    <variable id="mpg">
      Miles Per U.S. Gallon
    </variable>
    <variable id="cyl">
      Number of cylinders
    </variable>
    ..
  </variables>
  <records>
    <record>
      <real>21</real>
      <int>6</int>
      <na/>...
    </record>
    ...
  </records>
</dataset>
```

With the ability to introduce new elements such as <dataset>, <variable>, etc., it is essential that software can interpret this so that it can convert this specification of a dataset to, for example, an S data frame. For this, S must expect certain tags and attributes an interpret them appropriately to create a dataset. We define these tags and the relationships between them in what is called a DTD—Document Type Definition. This is like a LaTeX style sheet, and characterizes a class of documents with the same structure. It specifies how the elements of a document are related and what are the valid attributes for each element.

Another difference between HTML and XML is that HTML is concerned with how things appear on a page or in a browser, whereas XML deals with the structure of data and the relationships between the different nodes. However, we often want to display XML documents such as the dataset above and have it be typeset in a nice way. This is where XSL, the eXtensible Stylesheet Language, enters the picture. XSL allows one to say specifies rules which control how XML elements are transformed to other XML elements, including HTML, and indirectly to TeX, PDF (via FOP), etc.

This separation of structure from appearance and the ability to easily transform XML documents to nicely rendered HTML, TeX, PDF, etc. files is one reason why XML is so useful. It allows us to put in real information about data that can be easily accessed from software. There are other reasons why XML is likely to be successful. The fact that is reasonably simple and builds on people's familiarity with HTML makes it likely that people will be able to use it. Since there is no company that controls the specification or has a singularly vested interest in controlling how it evolves, it has a good chance of being accepted widely and becoming a generic standard. Perhaps the most important reason that XML may succeed is the existence of a vast collection of applications and libraries for most common programming languages to parse XML, generate and edit XML, and convert XML to other formats using XSL. The support for different languages means that data in XML is essentially application independent and can be shared across different systems. Finally, XML is not a new technology. It is a simplification of SGML (Structured Generalized Markup Language) that has been in existence for about 20 years. Thus, XML has benefited from years of experience.

Other uses

We have seen how statisticians can use XML to share datasets across different applications. We can also use XML as an alternative format for saving R sessions and S objects in general. In other words, when S objects are written to disk, they can be stored using an XML format. This provides a convenient mechanism for sharing objects between R and S-Plus via a single copy of the object, reducing both the storage space and the work to convert them. Many types of these S language objects could also be read into XLisp-Stat, Octave and Matlab from their XML representation with little effort.

While we can use XML for our own purposes, it is important to think of XML as a way to exchange data with other disciplines too. XML is becoming very widely used for a variety of different applications. For example, many of the Microsoft applications use XML as one of their storage formats. Similarly, the

Gnumeric spreadsheet and other tools create output using XML. Businesses are using it to store information such as catalogs, documents, forms, etc. Many database systems now produce XML output as the result of queries. Also, business-to-business transactions are frequently defined by XML messages. And the Simple Object Access Protocol (SOAP, http://www.develop.com/soap/) is a way to use XML for distributed computing.

In addition to these more context specific uses of XML, there are several general purpose document type specifications (DTDs) that will prove to be relevant for statistics. These include:

MathML Mathematical formulae and expressions can be specified using MathML. The major browsers are adding facilities for rendering these within HTML documents. We might use MathML to represent formulas and mathematical functions which can be used in model fitting and trellis plots; contained in documentation; and passed to symbolic math environments such as Mathematica and Maple which can read and write MathML.

SVG The Scalable Vector Graphics (http://www.w3.org/Graphics/SVG/Overview.htm8) DTD provides a way to use XML to specify graphics, i.e., drawing, text and images. We might use this for describing plots generated in S, or rendering plots in S that were created in other systems. SVG supports dynamic and interactive plots, and even animation. There are already plug-ins for browsers that render SVG content.

GML The Geography markup language (GML) is used for representing "geographic information, including both the geometry and properties of geographic features". This includes maps, for example.

Bioinformatics Bioinformatic Sequence Markup Language (BSML) and a variety of other markup languages have been proposed for different aspects of genetic data.

There is a staggeringly large number of other domain-specific uses of XML listed at OASIS that may be worth browsing to see how people are using XML (http://www.oasis-open.org/cover/siteIndex.html#toc-applications).

XML schema is a new topic that will be interesting for us. It allows us to specify not only the structure of a document, but also give information about the types of different XML elements (e.g., <real> is a real number, <int> is an integer, etc.). Essentially, it allows type specification. This would allow us to read XML documents more efficiently and even to automatically generate C code to read datasets into S.

We have also started using XML for documenting S functions and datasets. This gives us an easier format to manipulate than the '.Rd' files so that we can add enhanced functionality to the documentation system (e.g., multiple, separate examples; output from commands; test code and output; etc.) in a more structured way. And we are working on ways to use XML for generating reports or statistical analysis output which are "live" documents containing text, code, output and plots that can be re-run with different datasets, etc.

The XML package

Now that we have discussed what XML is and what it can be used for, we should describe the basics of how to use the **XML** package in R. As we mentioned, XML is basically a way of representing trees. So, the basic functions of the **XML** package provide ways to read XML documents into R as a list of lists (i.e., a tree) and then access and manipulate the nodes in these trees using the familiar [and [[S operators.

The primary function of the package for reading XML documents is xmlTreeParse which returns the tree of *XMLNode* objects. Because we use Dan Veillard's C-level libxml parser, the XML can be read from a regular file, a compressed file, a URL or directly from a string. For example,

```
doc <-
  xmlTreeParse(paste("http://www.omegahat.org/",
                     "RSXML/examples/prompt.xml",
                     sep = "")
doc <-
  xmlTreeParse(system.file("data", "mtcars.xml"))
```

The result contains not just the tree of *XMLNode* objects, but also information from the document's DTD. We can extract the different elements of tree by tag name or index. For example, we can get the first node

```
root <- xmlRoot(doc)
root[[1]]
```

which gives the collection of 11 <variable> nodes. We can get the first record either as

```
root[[2]]
```

or

```
root[["record"]]
```

We will usually want to do something with the tree and convert its contents to another form. This can be done after the call to xmlTreeParse and the tree has been created. Alternatively, we can provide one or more functions to xmlTreeParse which act as "handlers" for the different XML elements. These can process the *XMLNode* object associated with that XML element as it is encountered in the parsing and the function can return any object it wants to be put into the tree (including NULL to remove it

from the tree). The use of handlers allows us to build quite simple filters for extracting data. For example, suppose we have measurements of height taken at monthly intervals for different subjects in a study, but that potentially different numbers of measurements were taken for each subject. Part of the dataset might look like

```
<record><id>193</id>
  <height unit="cm">
    <real>140.5</real>
    <real>143.4</real>
    <real>144.8</real>
  </height>
</record>
<record>
  <id>200</id>
  <height>
    <real>138.4</real>
  </height>
</record>
```

Now, suppose we want to get the records for the subjects which have more than one observation. We do this by specifying a handler for the `<record>` tags and counting the number of sub-nodes of the `<height>` element.

```
xmlTreeParse("data",
  handlers=list(record=function(x,...)
    ifelse(xmlSize(x[["height"]]) > 1, x, NULL)))
```

This will discard the second record in our example above, but keep the first. We should note we can also use XSL to filter these documents before reading the results of the filtering into S. More sophisticated things can be done in S, but XSL can be simpler at times and avoids a dependency on S. The good thing is that we have the luxury of choosing which approach to use.

The package also provides other facilities for reading XML documents using what is called event or SAX parsing which is useful for very large documents. Also, users can create XML documents or trees within S. They can be written to a file or a string by incrementally adding XML tags and content to different types of "connections".

The future

XML provides an opportunity to provide a universal, easy-to-use and standard format for exchanging data of all types. This has been a very brief introduction to some of the ways that we as statisticians may use XML. There is little doubt that there are many more. Also, regardless of how we can exploit XML for our own purposes, we will undoubtedly encounter it more as we interact with other disciplines and will have to deal with it in our daily work. Therefore, we should help to define formats for representing data types with which we work frequently and integrate the XML tools with ours.

Resources

The surge in popularity of XML has rivalled that of Java and hence the number of books on all aspects of XML has increased dramatically over the past few months. There are books on XML, XSL, XML with Java, etc. As with any dynamic and emerging technology, and especially one with so many subdomains with special interests, the Web is probably the best place to find information. The following are some general links that may help you find out more about XML:

- W3's XML page (http://www.w3.org/XML/)
- OASIS (http://www.oasis-open.org/)
- Apache's XML page (http://xml.apache.org/)

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Programmer's Niche

by Bill Venables

Welcome and invitation

Welcome to the first installment of *Programmer's Niche*.

This is intended to be the first of a regular series of columns discussing R issues of particular interest to programmers. The original name was to have been *Programming Pearls* with the idea of specialising in short, incisive pearls of R programming wisdom, but we have decided to broaden it to include any matters of interest to programmers. The gems are still included, though, so if you have any favourites, let's hear about them.

That brings me to the real purpose of this preamble: to invite readers to contribute. I have offered to edit this column and occasionally to present an article but for the column to serve its purpose properly the bulk of the contributions must come from readers. If you have a particularly fine example of R