XML

Jim Harner

10/4/2020

2.3 XML

Plain text files do not have information on the location of data. eXtensible Markup Language, XML, provides a formal way to provide labeling or "markup" for data. Data is stored in an XML document as characters.

The function `xmlParse` from the R package `XML` parses the XML file. The variables can then be extracted using the `getNodeSet` function and formed into a `data.frame`.

```
library(XML)
nemoDoc <- xmlParse("pointnemotemp.xml")
class(nemoDoc)
```

```
## [1] "XMLInternalDocument" "XMLAbstractDocument"
```

The R data structure is an `XMLInternalDocument`, which exactly reproduces the file `pointnemotemp.xml`.

We now use `getNodeSet` to pull out `temperature` and `date` as character vectors (after unlisting).

```
nemoDocTemp <- unlist(getNodeSet(nemoDoc,
                                 "/temperatures/case/@temperature"),
                                 use.names=FALSE)
nemoDocDate <- unlist(getNodeSet(nemoDoc, "/temperatures/case/@date"),
                      use.names=FALSE)
```

The second argument `/temperatures/case/@temperature` goes down the XML hierarchy until `temperature` is found and them extracts it. Note that `temperature` and `date` are attributes of `case`.

We then coerce `date` to a date format using `as.Date` and `temperature` to numeric using `as.numeric`. The resulting vectors are then formed into a data frame and named `date` and `temp`.

```
nemoDoc.df <- data.frame(date=as.Date(nemoDocDate, "%d-%b-%Y"),
                         temp=as.numeric(nemoDocTemp))
head(nemoDoc.df)
```

```
##         date  temp
## 1 1994-01-16 278.9
## 2 1994-02-16 280.0
## 3 1994-03-16 278.9
## 4 1994-04-16 278.9
## 5 1994-05-16 277.8
## 6 1994-06-16 276.1
```

The XML format labels every single data value. Thus, an XML file can be manipulated automatically by a computer. When we are storing data in XML format, we are writing computer code, which allows us to communicate more about the data to the computer.

Note: we will be transitioning to xml2 in future versions of these notes.

XML Syntax The XML format consists of two parts:

- the XML markup
- the data

For example, consider the XML *element*:

```
<latitude>48.8S</latitude>
```

It contains the start tag, the data, and the end tag.

The `case` elements are contained within the `temperatures` element along with other elements. It differs in that there is no end tag, i.e., it is empty. However, it has two attributes: `date` and `temperature`.

```
<case date="16-JAN-1994" temperature="278.9"/>
```

The first line of an XML document must be a declaration that the file is an XML document and which version of XML is being used.

The syntax of HTML (regular web pages) and XML are similar, but there are differences.

- XML is case sensitive.
- XML does not have a fixed number of elements.

The `xmllib` software together with a command-line tool called `xmllint` can be used to check the syntax, e.g., `xmllint xmlcode.xml` from the UNIX shell.

XML design We must decide how to *design* an XML document. For example, we could use `record` rather than `case` and change the attribute names.

We could also use a `cases` tag and put the raw data between `<cases> ... </cases>`. Since `<case ... />` is so repetitive, this may be more efficient. That is, the data above is given in attributes, but could also be represented using elements.

XML schema We need to write the design down so that we can check that an XML document follows the design. The computer must be able to understand the design.

The design can be specified by creating a `schema` for an XML document, which is a description of the structure of the document. A number of technologies exist for specifying XML schema, but we will focus on the Document Type Definition (DTD) language. A DTD is a set of rules for an XML document. It contains:

- element declarations `<!ELEMENT>`
- attribute declarations `<!ATTLIST>`

Case study: Point Nemo The DTD for the `pointmenttemp.xml` file:

```
<!ELEMENT temperatures (variable,
    filename,
    filepath,
    subset,
    longitude,
    latitude,
    case*)>
<!ELEMENT variable (#PCDATA)>
<!ELEMENT filename (#PCDATA)>
<!ELEMENT filepath (#PCDATA)>
```

```
<!ELEMENT subset (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT case EMPTY>

<!ATTLIST case
    date ID #REQUIRED
    temperature CDATA #IMPLIED>
```

The case data is `EMPTY`. The data of other elements are plain text indicated by `#PCDATA`.

The `temperatures` element contain other elements, which are specified. The `*` indicates zero or more elements.

`<!ATTLIST>` declarations in a DTD are used to specify which attributes each element is allowed to have. In this example, only the `case` elements have attributes, so there is only one <!ATTLIST> declaration.

The `date` attribute for `case` elements is compulsory (`#REQUIRED`) and the value must be unique (`ID`). The `temperature` attribute is optional (`#IMPLIED`) and, if it occurs, the value can be any text (CDATA).

The Document Type Declaration can be:

- inline
- external

An XML document is said to be *well-formed* if it obeys the basic rules of XML syntax. If the XML document also obeys the rules given in a DTD, then the document is said to be *valid*.

Many standard schemas exist, e.g., Statistical Data and Metadata eXchange (SDMX) format has been developed by several large financial institutions.

Advantages and disadvantages Advantages of XML:

- Self describing format
- Representing complex data structures
- Data integrity

Disadvantages of XML:

- Verbosity
- Costs of complexity