The RExcelXML package

URL: <http://www.omegahat.org/RExcelXML>

Depends: ROOXML, Rcompression, XML

Duncan Temple Lang ([duncan@wald.ucdavis.edu](mailto:duncan@wald.ucdavis.edu))

# Overview

The most recent version of Microsoft Office and most of the other office suites use XML and a ZIP archive to store the contents of spreadsheets, word processing documents and presentations. This makes them readily accessible on all platforms from within languages that have XML support. The RExcelML is the start of a package to provide functionality for R users to both query and modify the contents of Excel workbooks and worksheets. The GUI-nature of the spreadsheet, the way it recomputes dependencies when values change and the ability to add graphical displays, text and even interactive controls make them a useful way to display the results of statistical data analysis, along with Word documents, dynamic Web pages, Google Earth and Maps, etc.

This document attempts to describe some of the basic functionality and data structures/classes we have quickly developed. They may need to be changed, but not significantly we expect and hope.

We assume you start with an existing .xslx file. If not, you can copy an empty one from the templateDocs/ directory of this package. You pass the name of this file to the function excelDoc() and now you have an object which can be used to list the files within the .xslx archive and fetch the contents of any of the files.

The ExcelArchive object is of interest for several tasks, but users will typically want to work with a Workbook object as this is the entity that contains the worksheets and the data. So we can create a Workbook object in R with the function workbook(). This takes either the name of the .xslx file or the ExcelArchive object.

The Workbook class has methods for the names() and [[() functions. names() for a Workbook returns the names of the worksheets it contains. These are the names that appear in the tabs of the Excel GUI . The [[() method allows you to extract a individual worksheet, either by name or by index/number.

So let’s see how this works in practice. There is a sample worksheet within the RExcelXML package and it contains 4 worksheets. The file is found as

f = system.file(“templateDocs”, “sample3sheets.xslx”,   
 package = “RExcelXML”)

So we can create our Workbook object with

wb = workbook(f)

and the ExcelArchive object as

e = excelDoc(f)

Calling names on each of these gives

names(wb)

rId1 rId2 rId3

"Sheet1" "Sheet2" "Sheet3"

rId4

"Rectangle & Image"

and

names(e)

[1] "[Content\_Types].xml"

[2] "\_rels/.rels"

[3] "xl/\_rels/workbook.xml.rels"

[4] "xl/workbook.xml"

[5] "xl/worksheets/sheet4.xml"

[6] "xl/drawings/\_rels/drawing1.xml.rels"

[7] "xl/worksheets/\_rels/sheet4.xml.rels"

[8] "xl/worksheets/sheet2.xml"

[9] "xl/worksheets/sheet3.xml"

[10] "xl/media/image1.jpeg"

[11] "xl/theme/theme1.xml"

[12] "xl/styles.xml"

[13] "xl/sharedStrings.xml"

[14] "xl/drawings/drawing1.xml"

[15] "xl/worksheets/sheet1.xml"

[16] "xl/calcChain.xml"

[17] "docProps/thumbnail.jpeg"

[18] "docProps/core.xml"

[19] "docProps/app.xml"

The last of these shows all the files within the .xslx and illustrates that to interpret the data, one must work with multiple different files. The RExcelXML package attempts to hide this complexity and present you with the notion of a workbook and a collection of worksheets, which have rows and columns and cells and which are similar to data frames in R.

So how do we get a worksheet? The simplest way is with [[ on our Workbook:

wb[[1]]

wb[[“Sheet1”]]

What does this give us? By default, this gives us an object of class Workbook. This is an object with two slots: content which is an XML document, and name which is a vector of length 2 giving the name of the .xslx file and the name of the XML file within that which is associated with this worksheet. You typically don’t need to worry about this name field; it is used for other computations and remembering from whence the content came so that we can save any changes we make to the correct place.

What if we don’t want the XML content of the worksheet? Suppose instead that we just want to identify the file within the .xslx archive which contains the content. Perhaps we are interactively working with the worksheet in Excel and we want to get the contents later, but want a way to refer to that sub-file. In this case, we want a WorksheetFile. We can get this with

wb[[“Sheet1”, asXML = FALSE]]  
wb[[1, asXML = FALSE]]

That’s easy, but perhaps we don’t really need this.

If the [[ operator is not convenient, you can use the getSheet() function directly.

## Working with Worksheet objects

So what do we want to do with our worksheet object? First thing is to ask for its dimensions, e.g.

sh = wb[[1]]

dim(sh)

[1] 7 2

This tells us that there are 7 rows and 2 columns – of active cells. This function ignores the empty columns and rows before and after the main block of data. At present, it does not ignore the empty columns and rows within the populated region.

Okay, so how do we get the data into R? Given the worksheet, we call can call getSheetContents() with the Worksheet as the argument, or alternatively we can use the short-hand syntactic sugar sh[], i.e. give me everything. Similarly, sh[,] works. So

getSheetContents(sh)

sh[]

sh[,]

are all equivalent.

# Inserting values in a Worksheet

## Adding to a worksheet

We can add content to an existing worksheet by specifying the rows and columns at which the content is to be added.

We can add text to a cell with sh[i, j] = “my text”. The values for i and j here can be numbers or j can be the name of a column, e.g. “B”, “ABC”. We can similarly insert other scalars, e.g. integer, numeric, logical values.

Not only can we add the content to the XML document, we can also add the updated XML document to the .xslx archive. To do this, we can add the argument update = TRUE.

We can also add more complex objects. If we assign an non-scalar value to a particular cell, the values are added to the spreadsheet along an axis. The axis is either horizontal or vertical for a vector. For a 2-dimensional object, it is assumed to be both.

## Adding a worksheet

In addition to adding to existing worksheets, we can create new worksheets. We can do this with the [[ operator to create a “would-be” worksheet. So wb[[“new sheet”]] will create a Worksheet object in R, but not in the .xslx archive. We can work with this Worksheet as if it were a regular element of the overall document. However, we have to call save(), or explicitly add it to the .xslx archive (our variable e) or call updateArchiveFiles() to “commit” the changes to the .xslx document.

The call to wb[[“my sheet”]] creates a new Worksheet and a new XML document with empty content. It is a blank slate. We can add cells, rows, columns to it. If we use wb[[“my sheet”, update = TRUE]], we add the sheet to the archive.

## Specifying styles for rows, columns or cells

In addition to being able to add content, we can also specify its appearance. We can do this for cells, or groups of cells and rows and columns. We can do this by explicitly specifying the appearance of each cell. However, it is much, much better to use an extra layer of abstraction by which we assign one or more cells a style and then change the characteristics of that style. We need only make changes to the style definition to have all the associated cells appear differently. This is an important aspect of display, but unfortunately very underused.

# Adding a Plot/Image

We can create plots in R and then add them to the worksheet by placing them at particular locations. These plots are not updated when the spreadsheet is changed - unless there is a connection to R. However, we can also create plots in Excel from within R

# Creating Tables within Worksheets

Of course a worksheet is a rectangular table. However, within a worksheet, we can have “sub-tables” which operate as a “local” unit. We can add controls to sort columns, etc.

# Creating Pivot Tables

# Connecting back to R

When we create a spread/work-sheet from within R, the results are fixed. It is possible to add code to the worksheet, and workbook generally, which can make calls back to R to dynamically compute values within the workshop using the R engine. This is feasible on Windows via DCOM.

# Alternative approaches

On Windows, we can use DCOM (Distributed Component Object Model) to communicate with Excel, Word, PowerPoint and various other applications as they are running. With this connection, we can programmatically query and modify the spreadsheets, documents, presentations directly from within R in much the same way that we can here. For example, we can access sheets, cells and so on. The way we access these is very different, but the document model is very similar.

### Colophon

This document was written using Microsoft Word. This is a change from my typical tools which are either LaTeX or Docbook/XML/XSL. The jury is still out!