Read, write, format Excel 2007 files with package xlsx

Adrian A. Drăgulescu
adrian.dragulescu@gmail.com

Constellation Energy Group

Jul 24, 2010

Excel and R

Excel

- Has a good user interface
- Does not scale well
- Calculations in a spreadsheet are not easy to *read*
- People are willing to try it and spend time with the help system

R

- Opposite of the above points is true
- Increases your productivity, best for repetitive/reproducible tasks
- Gets the statistics right... (bugs are fixed quickly)
- Takes time to learn

Excel is everywhere in your organization. What do you do?

Existing packages

A soup of packages that interact with Excel, see the wiki page...

- `RDCOMClient` and `rcom`, give you ultimate control, Windows only
- `RODBC`
- `xlsReadWrite`, Windows only
- `WriteXLS`, `gdata`, requires Perl
- `dataframes2xls`, requires Python
- `RExcelXML`

Apache POI project

- A Java API for Microsoft Documents
- Goal: to read/write various OOXML and OLE2 files
- Well established project, 8+ years old, active development
- New version released about twice a year
- Comprehensive test suite, bug repository, and active mail list
- About 20 developers
- For Excel, Word, PowerPoint, Outlook, Visio, Publisher files

Use existing code!

Package xlsx

Use `rJava` with `POI` to control Microsoft documents from `R`

$$\text{R} \longleftrightarrow \text{rJava} \longleftrightarrow \text{Java (POI)} \longleftrightarrow \text{OOXML/OLE2}$$

Advantages

- Use a stable and tested Java API
- R code is just a thin wrapper, easy to maintain
- Don't have to deal directly with XML and Microsoft schemas
- Works on all operating systems where Java is available

High level API

Read the contents of an xlsx file with

```
> read.xlsx <- function(file, sheetIndex, sheetName=NULL,
+   rowIndex=NULL, colIndex=NULL, as.data.frame=TRUE, header=TRUE,
+   colClasses=NA, keepFormulas=FALSE, ...){}
```

Write a data.frame to an xlsx file with

```
> write.xlsx <- function(x, file, sheetName="Sheet 1",
+   formatTemplate=NULL, col.names=TRUE, row.names=TRUE,
+   append=FALSE){}
```

Low level API

Need to construct everything piece by piece ...

- Create a workbook

```
> wb <- createWorkbook()          # a java object
[1] "Java-Object{Name: /xl/workbook.xml - Content Type: application/vnd.ope
```

- Create a sheet in this workbook

```
> sheet <- createSheet(wb, sheetName="Sheet1")
```

- Create 10 rows

```
> rows <- createRow(sheet, rowIndex=1:10)
```

- Create 5 columns

```
> cells <- createCell(rows, colIndex=1:5)  # a matrix
```

- Set value to cell `cells[1,1]`

```
> setCellValue(cells[[1,1]], "A")
```

Low level API

- Get the value of one cell back

```
> x <- getCellValue(cells[[1,1]])  # returns "A"
```

- Or you can get a block of values (same kind)

```
> M <- getMatrixValues(sheet, 1:3, 1:4) # rows=1:3, cols=1:4
```

- When done, save the created workbook to file

```
> saveWorkbook(wb, file="C:/Temp/test.xlsx")
```

To see all available java methods for a sheet object `.jmethods(sheet)`. You can call them directly

```
> sheet$getLastRowNum()
[1] 9
```

Cell formatting

```
> createCellStyle <- function(wb, hAlign=NULL, vAlign=NULL,
+   borderPosition=NULL, borderPen="BORDER_NONE", borderColor=NULL,
+   fillBackgroundColor=NULL, fillForegroundColor=NULL,
+   fillPattern=NULL, font=NULL, dataFormat=NULL){}
```

- Create cell style objects

```
> cs1 <- createCellStyle(wb, dataFormat="#,##0.00")
> cs2 <- createCellStyle(wb, dataFormat="m/d/yyyy")
> cs3 <- createCellStyle(wb, borderPosition="RIGHT",
+   borderPen="BORDER_DASHED", fillBackgroundColor="yellow",
+   fillForegroundColor="tomato", fillPattern="BIG_SPOTS")
```

- Apply the cell style to a group of cells

```
> res <- lapply(cells, setCellStyle, cs3)
```

- Create a cell comment

```
> createCellComment(cells[[1, 1]], "Ho Ho Ho", author = "Santa")
```

Sheet formatting

- Set the page zoom to 200%

  ```
  > setZoom(sheet, 200, 100)
  ```

- Autosize column, size to fit first column

  ```
  > autoSizeColumn(sheet, 1)
  ```

- Merge cells, join 3 columns on first row

  ```
  > addMergedRegion(sheet, 1, 1, 1, 3)
  ```

- Create a freeze pane, fix first row and column

  ```
  > createFreezePane(sheet, 2, 2)
  ```

Other effects `setPrintArea`, `createSplitPane`, etc.

Other

- Control the print setup

```
> printSetup <- function(sheet, fitHeight=NULL,
+   fitWidth=NULL, copies=NULL, draft=NULL, footerMargin=NULL,
+   headerMargin=NULL, landscape=FALSE, pageStart=NULL,
+   paperSize=NULL, noColor=NULL){}
```

- Add images to sheets

```
> addPicture <- function(file, sheet, scale=1, startRow=1,
+   startColumn=1){}
```

Concluding remarks

Limitations

- Implementation is not fast
- Reading and writing "large" data.frames will trigger a jvm out of heap memory
- Not feature complete. You cannot create pivot tables, read/write password protected files, run macros

Good things

- Platform independent
- Can format the output the way you need
- Can use the same jars for processing Word, Powerpoint files, Outlook messages, Visio files

Contributions welcome!

Constellation Energy®

www.constellation.com