SchemaOnRead Manual

Argonne National Laboratory Technical Report ANL-15/16

Michael J. North

September 30, 2015

Introduction

SchemaOnRead provides tools for implementing the schema-on-read technique for R, including a single function call (e.g., `schemaOnRead("filename")`) that reads text (TXT), comma separated value (CSV), raster image (BMP, PNG, GIF, TIFF, and JPG), R data (RDS), HDF5, NetCDF, spreadsheet (XLS, XLSX, ODS, and DIF), Weka Attribute-Relation File Format (ARFF), Epi Info (REC), Pajek network (PAJ), R network (NET), Hypertext Markup Language (HTML), SPSS (SAV), Systat (SYS), and Stata (DTA) files. It also recursively reads folders (e.g., `schemaOnRead("folder")`), returning a nested list of the contained elements.

Example Uses

One way to use SchemaOnRead is to recursively load a folder. The result is a named list of elements for each entry in the folder's tree. The element names are converted to valid R variable names corresponding to the file or folder names. Sub-elements (e.g., files or subfolders) of a folder can be accessed using the R named list ("`$`") operator followed by the sub-element name.

Another way to use SchemaOnRead is to conveniently load a file without needing to handle the specifics of the file format. In this case the result is a variable containing the file contents.

An example showing how to read a folder tree starting in "`../inst/extdata`" is shown below. In this case, the contents of the "`dir1/Data.csv`" file within "`../inst/extdata`" is shown by accessing `results$dir1$Data.csv` as needed. The text file is as follows:

```
Name,Size,Weight
A,Small,1
B,Medium,2
C,Large,3
```

The results in R are as follows:

```
library(SchemaOnRead)
results <- schemaOnRead("../inst/extdata")
print(results$dir1$Data.csv)
```

```
##      Name   Size   Weight
## 1    A      Small  1
## 2    B      Medium 2
## 3    C      Large  3
```

Individual files can also be easily accessed. An example XML source file in the "../inst/extdata/data.xml" folder is as follows:

```
<example>
        <to>A</to>
        <from>B</from>
        <title>Important
                <subtitle>File</subtitle>
        </title>
        <text>Read me.</text>
</example>
```

The results in R are as follows:

```
library(SchemaOnRead)
xmlFile <- schemaOnRead("../inst/extdata/data.xml")
print(xmlFile)
```

```
## $to
## [1] "A"
##
## $from
## [1] "B"
##
## $title
## $title$text
## [1] "Important"
##  ## $title$subtitle
## [1] "File"
##
##
## $text
## [1] "Read me."
```

The `verbose` flag can be used to trace a call's progress or diagnose issues as shown here:

```
library(SchemaOnRead)
folder <- schemaOnRead("../inst/extdata", verbose = TRUE)
```

```
## [1] "schemaOnRead processing ../inst/extdata"
## [1] "schemaOnRead processing ../inst/extdata/arffexample.arff"
## [1] "schemaOnRead processing ../inst/extdata/data.xml"
## [1] "schemaOnRead processing ../inst/extdata/dir1"
## [1] "schemaOnRead processing ../inst/extdata/dir1/Data.csv"
## [1] "schemaOnRead processing ../inst/extdata/dir1/Data1.dif"
## [1] "schemaOnRead processing ../inst/extdata/dir1/Data1.xlsx"
## [1] "schemaOnRead processing ../inst/extdata/dir1/Data2.xls"
## [1] "schemaOnRead processing ../inst/extdata/dir1/dir3"
## [1] "schemaOnRead processing ../inst/extdata/dir1/dir3/data.xml"
## [1] "schemaOnRead processing ../inst/extdata/dir1/example.txt"
## [1] "schemaOnRead processing ../inst/extdata/dir1/spreadsheet.ods"
## [1] "schemaOnRead processing ../inst/extdata/dir2"
## [1] "schemaOnRead processing ../inst/extdata/dir2/data.xml"
## [1] "schemaOnRead processing ../inst/extdata/dir2/data1.dif"
## [1] "schemaOnRead processing ../inst/extdata/dir2/Example.net"
## [1] "schemaOnRead processing ../inst/extdata/dir2/Example.paj"
```

```
## [1] "schemaOnRead processing ../inst/extdata/dir2/example.rec"
## [1] "schemaOnRead processing ../inst/extdata/dir2/index.html"
## [1] "schemaOnRead processing ../inst/extdata/example.rds"
## [1] "schemaOnRead processing ../inst/extdata/image.gif"
## [1] "schemaOnRead processing ../inst/extdata/image.jpg"
## [1] "schemaOnRead processing ../inst/extdata/image.png"
## [1] "schemaOnRead processing ../inst/extdata/image.tiff"
```

Implementation

The SchemaOnRead package uses a recursive implementation. The initial user function call iterates over the given list of processors, invoking each in turn until one returns a non-null value. Processors are sequentially invoked in the order given by the input list, scanning from index number one upwards. Processing continues as long as each processor returns null. The results from the first processor to return a non-null value is stored as the content for the entry and processing of that entry stops. All of the results are stored in a named list. The order of the resulting list is given by the processing order. The list names are taken from the entry names (e.g., file or folder names). List names are converted to valid R variable names by replacing everything other than non-alphanumeric characters and dots with underscores. If the resulting variable name is already defined then "`_A`" is repeatedly appended to the name until it becomes unique.

Several special processors are defined. These include processors for nonexistent entries, directories, and entries of unknown types.

The `schemaOnReadProcessEntryDoesNotExist` processor returns null if the given entry exists and returns the value "Entry Does Not Exist" if entry file does not. It is meant to be the initial processor in most processor lists to intercept nonexistent entries before they waste execution time in other processors. In rare cases, special processing may need to be done on nonexistent entries so these unusual processors would run first.

The `schemaOnReadProcessDirectory` processor handles directories recursively as previously discussed. It is intended to be the second processor to run in normal lists.

The `schemaOnReadProcessDefaultFile` processor accepts all entries that exist. It returns the value "File Type Unknown" as a string when it succeeds. It is meant to be last processor to run to provide a default value for file types that are not otherwise recognized.

SchemaOnRead itself includes two processing lists. The predefined processor functions used in both lists and list assignments are shown in Table 1. The default list from `schemaOnReadDefaultProcessors()` is used for standard SchemaOnRead entry processing. The simple processing list from `schemaOnReadSimpleProcessors()` provides an easy to use starting point for fully customized user processor lists.

Function	List or Lists	Purpose
`schemaOnReadProcessDirectory(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default and Simple	Recursively Read Folders
`schemaOnReadProcessCSVFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Comma Separated Value Files
`schemaOnReadProcessTextFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Text Files
`schemaOnReadProcessRDSFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read R Data Files
`schemaOnReadProcessXMLFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Extensible Markup Language Files
`schemaOnReadProcessXLSandXLSXFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Simple	Read Microsoft Excel Files
`schemaOnReadProcessDIFFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Data Interchange Format Files
`schemaOnReadProcessODSFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Open Document Spreadsheet Files
`schemaOnReadProcessStata13File(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Stata Files
`schemaOnReadProcessBitmapsFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Bitmap Files
`schemaOnReadProcessGIFFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read Graphics Interchange Format Files
`schemaOnReadProcessTIFFFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)`	Default	Read TIFF Files
`schemaOnReadProcessNetCDandH5FFile(path = ".", processors =`	Simple	Read NetCDF

Function	List or Lists	Purpose
schemaOnReadDefaultProcessors(), verbose = FALSE)		and HDF5 Files
schemaOnReadProcessPajekFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default	Read Pajek Network Files
schemaOnReadProcessHTMLFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default	Read HTML Files
schemaOnReadProcessARFFFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default	Read Weka ARFF Files
schemaOnReadProcessEPIINFOFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default	Read Epi Info REC Files
schemaOnReadProcessSPSSFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default	Read SPSS Files
schemaOnReadProcessEntryDoesNotExist(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default and Simple	Accepts All Nonexistent Files and Returns "Entry Does Not Exist"
schemaOnReadProcessDefaultFile(path = ".", processors = schemaOnReadDefaultProcessors(), verbose = FALSE)	Default and Simple	Accepts All Existing Files and Returns "File Type Unknown"

Table 1: SchemaOnRead Predefined Processors

User-Defined Processors

New processors can be defined to support user-specified work. New processors are normally prepended to the front of the default list to let them to take precedence while still allowing the standard processors to work, if needed. Alternatively, a list of processors that just recursively scans folders can be found by calling the schemaOnReadSimpleProcessors function. User-specified processors can be added to this list to create a fully customized tool chain. An example showing how to define a new processor follows.

```
## Cite the needed library.
library(SchemaOnRead)
```

```
## Define a new processor.
newProcessor <- function(path, processors, verbose) {

         ## Check the given path.
         if ((file.exists(path)) &&
               (tolower(tools::file_ext(path)) == "tbd")) {

                 ## Return a new result...
                 return("Put Code for Processing TBD Files Here...")

          } else {

                 ## Note that the file does not exist.
                 return("Entry Does Not Exist")

          }

}

## Define a new processors list.
newProcessors <- c(newProcessor, schemaOnReadDefaultProcessors())

## Use the new processors list.
schemaOnRead(path = "../inst/extdata", processors = newProcessors)
```

Summary

SchemaOnRead provides tools for implementing the schema-on-read technique for R, including a single function call that reads a wide range of file types. It also recursively reads folders (e.g., `schemaOnRead("folder")`), returning a nested list of the contained elements.

Acknowledgements

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory "Argonne". Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. Argonne National Laboratory's work was supported under U.S. Department of Energy contract DE-AC02-06CH11357. Argonne National Laboratory's work was supported under U.S. Department of Energy contract DE-AC02-06CH11357.