Active Documents with Org-mode

Eric Schulte and Dan Davison

22 February 2011

Abstract

Org-mode is a simple, plain text, markup language for hierarchical documents allowing intermingled data, code and prose. An entire research project, including initial note taking, planning, task management, experimentation, analysis, and publication may take place within a single Org-mode document. This article introduces Org-mode with an overview of syntax, a working *reproducible* example of embedded data analysis, and a summary of the features that make Org-mode a particularly useful tool for the scientific researcher.

1 Introduction

Org-mode is implemented as a part of the Emacs text editor [7]. It was initially developed as a simple outlining tool intended for note taking and brainstorming, and was later augmented with task management tools—enabling notes to be transformed into tasks with deadlines and priorities—and with syntax for the inclusion of tables, data blocks, and active code blocks. Users new to Org-mode often start with its simple plain-text note taking system, then move on to increasingly sophisticated features as their comfort level permits.

Reproducible Research (RR) is the practice of publishing scientific results along with the software environment and data required for reproduction of all computational analyses presented in the publication [3]. Reproducibility is essential to peer reviewed research, however, scientific publications often lack the information required for reviewers to reproduce the analysis described in the work.

> An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the

> scholarship. The actual scholarship is the complete software development environment and complete set of instructions which generated the figures.
>
> (Donoho [1])

Org-mode supports RR with syntax for including in-line data and code, mechanisms for evaluating embedded code, and publishing functionality that may be used to automate the computational analysis and generation of figures. This article will focus on the features of Org-mode that support the practice of RR; information on other aspects of Org-mode can be found in the manual [2] [1] and the community wiki [2]. The plain text Org-mode source of this article is available for download [3]; a user with the requisite open-source software can execute the source code examples, which analyze a dataset and create graphics, and export the complete paper to one of several output formats.

2 Syntax

2.1 Outlines

Org-mode documents are organized using a hierarchical outline. The outline can be folded and expanded, hiding or exposing as much of the document as wanted. Using this facility, even very large documents can be comfortably navigated in a manner similar to that of a file system. Headlines are indicated by leading `*`'s as shown below in the folded view of this article from within Org-mode.

```
* Introduction...
* Syntax...
** Outlines...
** Code and Data...
* Evaluation...
* Example Application...
** Download External Data...
** Parsing...
** Analysis...
** Display...
```

[1] http://orgmode.org/manual/

[2] http://orgmode.org/worg/

[3] https://github.com/eschulte/CiSE/raw/master/org-mode-active-doc.org

```
* Conclusion...
* COMMENT How to Export this Document...
* Footnotes...
```

The ...'s at the end of each line indicate that the content of the heading is hidden from view. Notice that the heading beginning with the keyword `COMMENT` is not included in the exported document. Org-mode uses many such keywords for associating information with headlines.

2.2 Code and Data

Using a simple 'block' syntax, both code and data can be embedded in Org-mode documents, as follows.

```
First a data block.
#+begin_example
  raw textual data
#+end_example

Second a code block.
#+begin_src sh
  echo "shell script code"
#+end_src
```

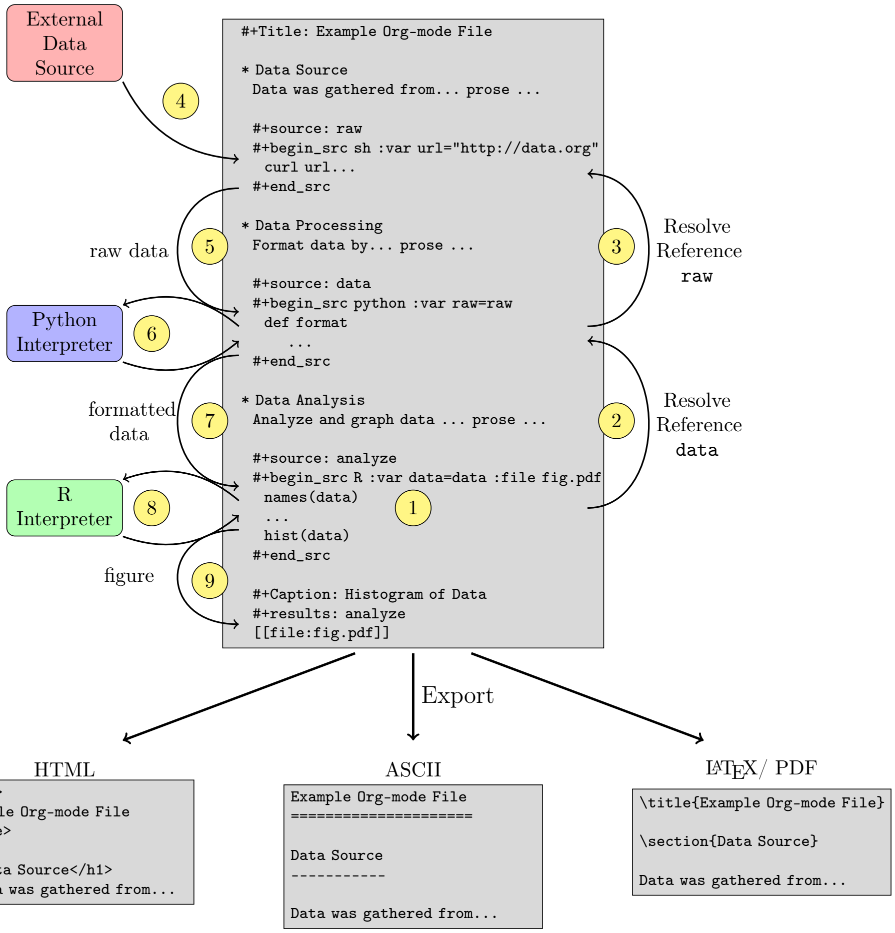
Code and data blocks can be named, allowing their contents to be referenced form elsewhere in the Org-mode file, as illustrated in the following example where the shell script references the contents of the data block.

```
First a data block.
#+results: raw-data
#+begin_example
  raw textual data
#+end_example

Second a code block.
#+begin_src sh :var text=raw-data
  echo $text|wc
#+end_src

#+results:
: 1       3      17
```

Cross references between the code and data elements of an Org-mode file turn Org-mode into a powerful multilingual programming environment, in which data and code expressed in many different programming languages may interact.

3 Evaluation

Code and data references make possible strings of *chained evaluation*. Figure 1 shows the series of actions which result when the `analyze` code block (fig. 1, 1) is evaluated interactively or during export.



```
#+Title: Example Org-mode File

* Data Source
  Data was gathered from... prose ...

  #+source: raw
  #+begin_src sh :var url="http://data.org"
    curl url...
  #+end_src

* Data Processing
  Format data by... prose ...

  #+source: data
  #+begin_src python :var raw=raw
    def format
        ...
  #+end_src

* Data Analysis
  Analyze and graph data ... prose ...

  #+source: analyze
  #+begin_src R :var data=data :file fig.pdf
    names(data)
    ...
    hist(data)
  #+end_src

  #+Caption: Histogram of Data
  #+results: analyze
  [[file:fig.pdf]]
```

HTML

```
<title>
  Example Org-mode File
</title>

<h1>Data Source</h1>
<p>Data was gathered from...
```

ASCII

```
Example Org-mode File
=====================

Data Source
-----------

Data was gathered from...
```

LaTeX/ PDF

```
\title{Example Org-mode File}

\section{Data Source}

Data was gathered from...
```

Figure 1: Active Org-mode Document

1. The `analyze` code block is evaluated. The `:var data=data` header argument causes Org-mode to evaluate the `data` reference.

2. To resolve this reference the `data` code block is located in the Org-mode file and is evaluated.

3. The `:var raw=raw` header argument causes Org-mode to resolve the `raw` reference.

4. The `raw` code block is evaluated, the `:var url="http://data.org"` header argument is evaluated as a literal value which is assigned to the `url` variable and passed to the shell script.

   The shell script then downloads data from the external url and makes these data available to Org-mode.

5. The results of the shell script are assigned to the `raw` variable, which is passed to the Python code in the body of the `data` code block.

6. This code is passed to an external Python interpreter which evaluates the Python code and returns its result to Org-mode.

7. The results of the `data` code block are then assigned to the `data` variable and passed to the R code in the body of the `analyze` code block.

8. This code is then passed to an external R interpreter, which generates a figure that is written to file specified in `:file fig.pdf`.

9. A reference to this figure is then passed from the `analyze` code block back to Org-mode, which inserts a link marked by double square brackets into the body of the Org-mode document. On export to HTML, ASCII, LaTeX, or another format supported by Org-mode, the linked figure will be embedded into the exported document.

4 Example Application

The application of Org-mode to RR is illustrated with an analysis of baseball statistics. The ordered nature of baseball games makes them particularly amenable to statistical analysis. The performance of baseball players, and the course of baseball games, are routinely captured in a small number of statistics that are comparable across space and time.

In this example we analyze the correlation of several common offensive statistics with the attendance at Major League Baseball (MLB) games in the

2010 season. We hypothesize what every baseball fan wants to believe, that large crowds spur the home team to superior levels of performance. The offensive statistic that has the largest correlation with high attendance is found and reported.

4.1 Download External Data

This example will show correlation of home team offensive statistics with attendance for the `2010` MLB season.

This first code block, named `url`, translates the numerical season shown above into the url for the `retrosheet.org` [4] website, a website devoted to the collection and curation of major league baseball statistics.

With the `raw-data` shell code block, the zip file of statistics located at the specified url is downloaded and its contents are unpacked into a local text file named `2010.csv`. The `:cache yes` header argument ensures that this code block is only run once and the data are not downloaded again every time the results of the code block are referenced.

Next the `stat-headers` Python code block returns a list of the names of the offensive statistics that will be tested for correlation with attendance.

4.2 Parsing

The next two shell code blocks, `offensive-stats` and `attendance`, collect the offensive statistics and the attendance from the raw data file produced by the `raw-data` code block.

4.3 Analysis

The `analysis` code block uses the `R` statistical programming language to calculate correlations between the outputs of the `offensive-stats` and `attendance` code blocks, whose values are saved into the `stats` and `attendance` variables respectively.

The most correlated column, namely `intentional walks`, can be mentioned in the text using an inline code block. The Org-mode syntax for an inline block can be seen below.

These results indicate that the fans' belief in the effect of large crowds is shared by the visiting team, which chooses to walk a dangerous home team

[4] The information used here was obtained free of charge from and is copyrighted by Retrosheet. Interested parties may contact Retrosheet at "www.retrosheet.org".

hitter rather than take the chance that the large crowd will spur him to a potentially damaging performance.

4.4 Display

Using gnuplot we can plot the number of forced walks and the attendance for the five games with the most forced walks (see Figure 2).

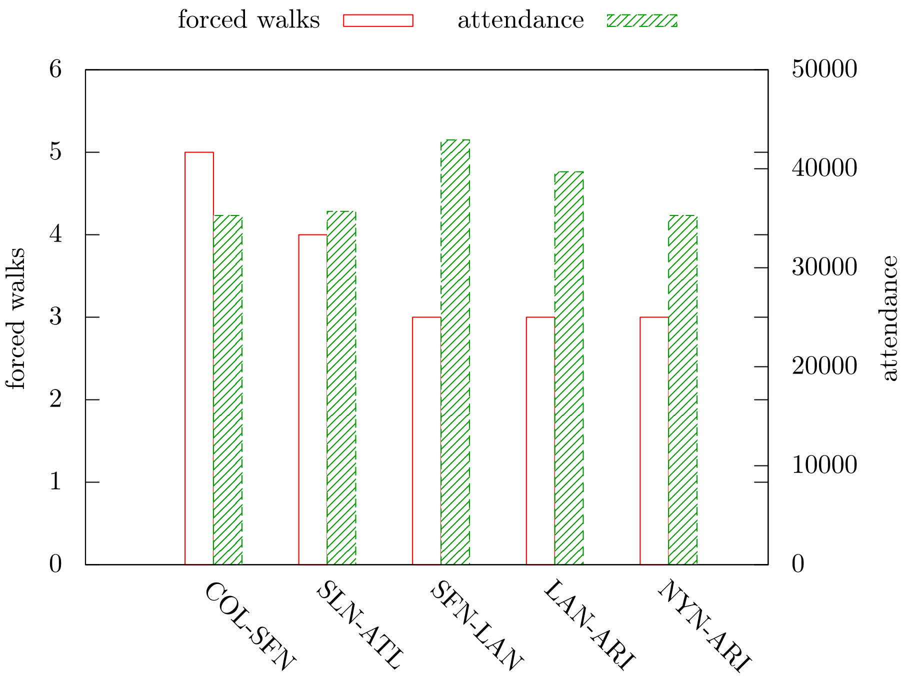


Figure 2: Top 5 games by forced walks, with forced walks and attendance shown.

Commingling code and prose, as demonstrated in this example, makes it possible for the author to collect all relevant information into a single place. This practice benefits the reader, who can reproduce the calculations performed in the work, and also extend the analysis, possibly within Org-mode itself. For example, the reader of this article can re-run the analysis for another season by simply changing the value of the `season` code block above and re-exporting the file.

5 Conclusion

There are a number of features of Org-mode that make it a good choice for reproducible research; some of these are *essential* for any RR tool, and others alleviate common burdens of practicing RR.

Of the *essential* properties, arguably the most important is that as part of Emacs, the Org-mode copyright is owned by the Free Software Foundation [6]. This ensures that Org-mode is now and will always be free and open source software. This is directly related to two of the goals of RR. First, Org-mode is available free of charge to install by any user on any system ensuring access to the software environment required for reproduction. Second, the source code specifying the inner workings of Org-mode is open to inspection, ensuring that the mechanisms through which Org-mode generates scientific results are open to review and verification.

In addition to its open source pedigree, Org-mode benefits in other ways from its development as part of Emacs. Emacs is one of the most widely ported pieces of software in existence, with versions that run on all major operating systems. This ensures that Org-mode documents can be incorporated into almost any computer working environment. Emacs is also widely used by the scientific community for editing both prose documents and source code. By leveraging existing Emacs editing support, Org-mode is able to offer its users a comfortable and familiar editing environment for all types of content. Finally, due to Org-mode's implementation in the Emacs extension language, *Emacs Lisp* [5], it is possible for users to customize the behavior of Org-mode to their particular needs and to add support for arbitrary new programming languages—Org-mode currently has support for over thirty programming languages.

Org-mode addresses many common problems in the practice of RR. Given that a single Org-mode document can be used for every stage of a research project from brain-storming, through software development and experimentation, to publication, the author is largely relieved of the burden of tracking resources required for reproduction of the work. Such large amounts of information can result in extremely large files, however the hierarchical folding of Org-mode documents enables users to comfortably read and edit such files. The files themselves are encoded in plain text, which enhances their portability and makes them easy to integrate well with version control systems, allowing for revision tracking and collaboration [4].

Org-mode documents can run the gambit from simple collections of plain-text notes, to complex laboratories housing data and analysis mechanisms, to publishing desks with facilities for the display and export of scientific results. There is a friendly community of Org-mode users and developers who communicate on the Org-mode mailing list [5]; through answering questions and helping each other to master Org-mode's many features, this community helps to solve one of the largest hurdles posed by any RR tool, namely learning how to use it.

References

[1] J.B. Buckheit and D.L. Donoho. *Wavelets and Statistics*, chapter WaveLab and Reproducible Research. Springer-Verlag, 1995.

[2] Carsten Dominik et al. *The Org Mode 7 Reference Manual*. Free Software Foundation, 2010.

[3] Sergey Fomel and Jon F. Claerbout. Guest editors' introduction: Reproducible research. *Computing in Science and Engineering*, 11:5–7, 2009.

[4] Konrad Hinsen, Konstantin Läufer, and George K. Thiruvathukal. Essential tools: Version control systems. *Computing in Science and Engineering*, 11(6):84–91, 2009.

[5] Bil Lewis, Dan LaLiberte, and Richard Stallman. *GNU Emacs Lisp Reference Manual*. Free Software Foundation, Boston, MA, third edition, 2010.

[6] Richard Stallman. Free software foundation (fsf). In *Encyclopedia of Computer Science*, pages 732–733. John Wiley & Sons, Chichester, UK, 2003.

[7] Richard M. Stallman. Emacs the extensible, customizable self-documenting display editor. *SIGPLAN Not.*, 16(6):147–156, 1981.

[5] http://lists.gnu.org/mailman/listinfo/emacs-orgmode