



ACTIVE DOCUMENTS WITH ORG-MODE

By Eric Schulte and Dan Davison

Org-mode is a simple, plain-text markup language for hierarchical documents that allows the intermingling of data, code, and prose.

Org-mode is implemented as a part of the Emacs text editor.¹ It was initially developed as a simple outlining tool intended for note taking and brainstorming. It was later augmented with task management tools—letting researchers transform notes 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.

In *reproducible research* (RR), researchers publish scientific results along with the software environment and data required to reproduce all computational analyses in the publication.² Reproducibility is essential to peer-reviewed research, but scientific publications often lack the information required for reviewers to reproduce the analysis described in the work. As Jonathan Buckheit and David L. Donoho noted,³

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.

Org-mode supports RR with syntax for including inline data and code,

mechanisms for evaluating embedded code, and publishing functionality that might be used to automate the computational analysis and generation of figures. Here, we focus on the Org-mode features that support the practice of RR; information on other aspects of Org-mode can be found in the manual (<http://orgmode.org/manual>)⁴ and in the community wiki (<http://orgmode.org/worg>).

The plain text Org-mode source of this article is available for download at <https://github.com/eschulte/CiSE/raw/master/org-mode-active-doc.org> (see the sidebar “How to Download this Document” for more information). Readers with the requisite open-source software can execute the source code examples—which analyze a dataset and create graphics—as well as export the complete paper to one of several output formats.

Syntax

Org-mode documents are plain text files organized using a hierarchical outline defined by a number of simple syntactical rules.

Outlines

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 in the folded view of this article in Figure 1.

The ellipses at the end of each line indicate that the heading's content 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.

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 from elsewhere in the Org-mode file. Figure 2 shows an example, in which the shell script references the data block's content.

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

Evaluation

Code and data references make *chained evaluation* strings possible.

HOW TO EXPORT THIS DOCUMENT

This article was originally composed as an Org-mode document; the raw plain-text version is available for download at <https://github.com/eschulte/CiSE/raw/master/org-mode-active-doc.org>. All of the examples presented in this article can be interactively recreated from the original document.

Requirements

The first step is to ensure that you have installed on your system recent versions of Emacs (www.gnu.org/software/emacs) version 23 or greater and Org-mode (<http://orgmode.org>) version 7.5 or greater. To evaluate the code blocks in our article, you also need the following programming languages installed on your system:

- Python (www.python.org),
- R (www.r-project.org),
- Emacs speaks statistics (ESS; <http://ess.r-project.org>)
- gnuplot (www.gnuplot.info), and
- gnuplot-mode (www.emacswiki.org/emacs/GnuplotMode)

Configuration

Next, you can evaluate the emacs-lisp code block to configure Org-mode to export our article (see Figure A).

Export

After installing all required software, you can export the article to several different back ends in three steps.

First, open this document in Emacs. Second, evaluate the “Configuration” emacs-lisp code block immediately previous in this document. This can be done with `C-c C-v p` to jump to the previous code block, then `C-c C-c` to evaluate the code block where `C-c` means press “c” while holding the control key, `C-v` means press “v” while holding the control key, and so forth.

Finally, use `C-c C-e` to open the Org-mode export dialog, which displays a number of backend options and the key which should be used to export to that backend, for example, press “d” to export this document to a .pdf and open the resulting file in your document reader, or press “b” to export this document to .html and open the resulting file in your Web browser.

```
#+source: configuration
#+begin_src emacs-lisp :results silent
;; first it is necessary to ensure that Org-mode loads support for the
;; languages used by code blocks in this article
(org-babel-do-load-languages
 'org-babel-load-languages
 '((sh . t)
  (org . t)
  (emacs-lisp . t)
  (python . t)
  (R . t)
  (gnuplot . t)))
;; then we'll remove the need to confirm evaluation of each code
;; block, NOTE: if you are concerned about execution of malicious code
;; through code blocks, then comment out the following line
(setq org-confirm-babel-evaluate nil)
;; finally we'll customize the default behavior of Org-mode code blocks
;; so that they can be used to display examples of Org-mode syntax
(setf org-babel-default-header-args:org '(:exports . "code"))
#+end_src
```

Figure A. The emacs-lisp code block to configure Org-mode to export this article.

Figure 3 shows the series of actions that result when the `analyze` code block is evaluated interactively or during export.

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 causing the `:var url=http://data.org` header argument to be
5. The results of the shell script are assigned to the `raw` variable, which is passed to the Python

evaluated as a literal value that's 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.

```

* Introduction...
* Syntax...
** Outlines...
** Code and Data...
* Evaluation...
* Example Application...
** Download External Data...
** Parsing...
** Analysis...
** Display...
* Conclusion...
* COMMENT How to Export this Document...
* Footnotes...

```

Figure 1. The folded view of this article. Headlines are indicated by leading *'s.

```

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

```

Figure 2. The shell script referencing the data block's content. By naming code and data blocks, you can reference their contents from elsewhere in the Org-mode file. data block's content.

- 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 `data` code block's results 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 the 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 Org-mode supported format, the linked figure will be embedded into the exported document.

Example Application

To illustrate Org-mode's application to RR, we use an analysis of baseball statistics. The ordered nature of baseball games makes them particularly amenable to statistical analysis. Baseball players' performance and the course of baseball games are routinely captured in a few statistics that are comparable across games and seasons.

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 performance levels. We found and report on the offensive statistic that has the largest correlation with high attendance.

Download External Data

Our example correlates home team offensive statistics with attendance for the 2010 MLB season (see Figure 4).

As Figure 5 shows, this first code block, named `url`, translates the numerical 2010 season into the url for the retrosheet.org website (we obtained this copyrighted information free of charge; see www.retrosheet.org). The website is devoted to collecting and curating MLB statistics.

As Figure 6 shows, 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 run only once and the data aren't downloaded again every time the code block's results are referenced.

Next, the `stat-headers` Python code block returns a list of the names of the offensive statistics that we'll test for correlation with attendance (see Figure 7).

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.

Analysis

The `analysis` code block uses the R statistical programming language to calculate correlations between the

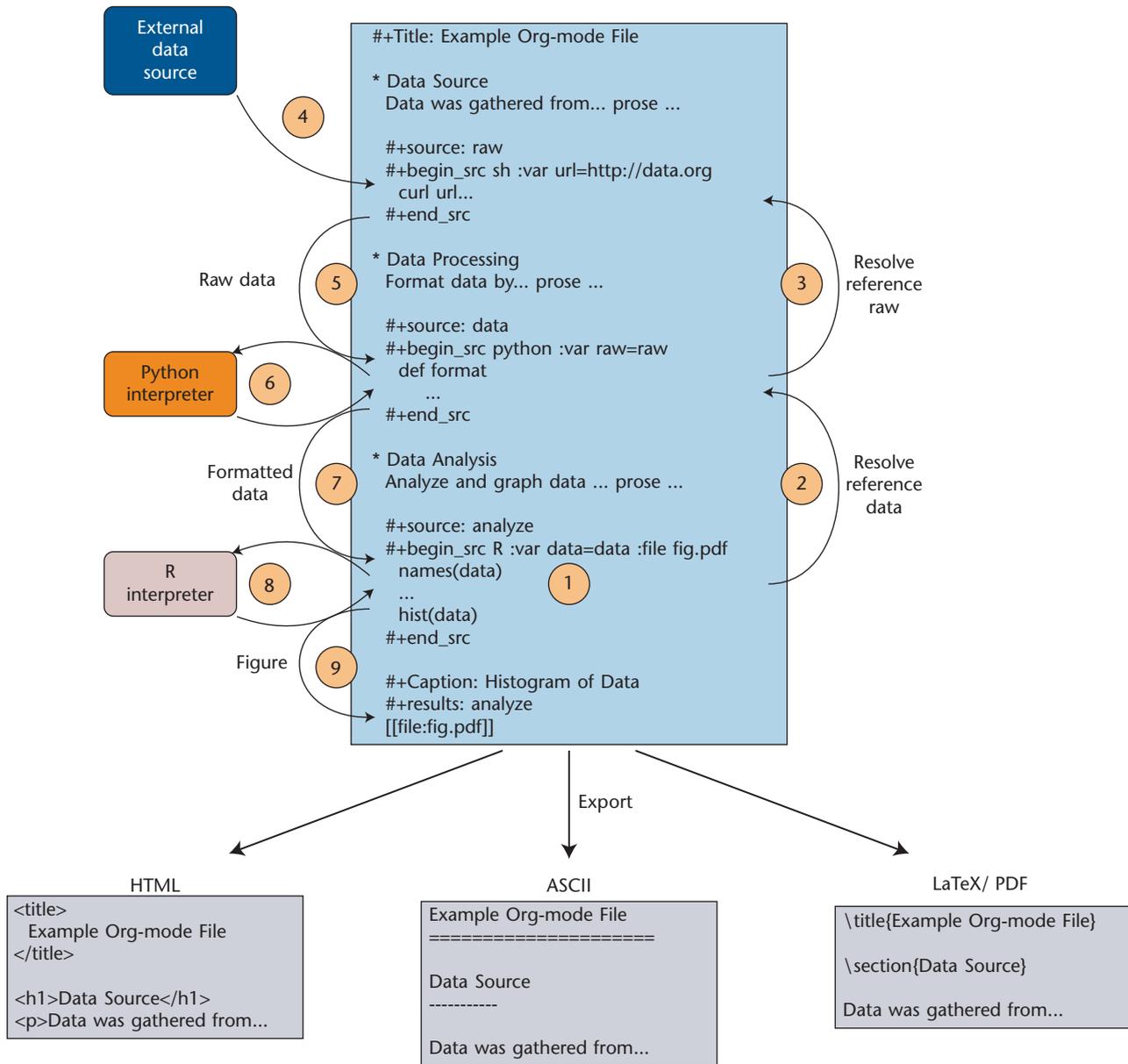


Figure 3. Active Org-mode document. Variables of the `analyze` code block reference the results of previous code blocks (shown on the right), in resolving these references the referenced code blocks are evaluated, and their results are passed back to the `analyze` code block (on the left side).

outputs of the `offensive-stats` and `attendance` code blocks, whose values are saved into the `stats` and `attendance` variables respectively.

The most correlated column, `intentional walk`, can be mentioned in the text using an inline code block. The code below shows the Org-mode syntax for an inline block. The results indicate that the fans' belief in the effects of a large crowd is shared by the visiting team,

which chooses to walk a dangerous home-team hitter rather than take the chance that the crowd will spur him to a potentially damaging performance.

Display

Using `gnuplot`, we can plot the number of forced walks and the attendance for the five games with the most forced walks (see Figures 10 and 11).

```

#+source: season
#+begin_src emacs-lisp
:exports none
2010
#+end_src

```

Figure 4. The example application will correlate home team offensive statistics with attendance for the 2010 Major League Baseball season.

SCIENTIFIC PROGRAMMING

```
#+source: url
#+begin_src sh :var season=season :exports none
    echo "http://www.retrosheet.org/gamelogs/gl$season.zip"
#+end_src
```

Figure 5. The URL code block. This block translates the numerical 2010 season into the URL for the website that collects Major League Baseball statistics.

```
#+source: raw-data
#+headers: :exports none
#+begin_src sh :cache yes :var url=url :file 2010.csv
    wget $url && \
        unzip -p gl2010.zip > 2010.csv && \
        rm gl2010.zip
#+end_src
```

Figure 6. 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.

```
#+source: stat-headers
#+headers: :exports none
#+begin_src python :results list :cache yes :return fields
    import urllib2
    url = 'http://www.retrosheet.org/gamelogs/glfields.txt'
    fp = urllib2.urlopen(url)
    fields = []
    for line in fp:
        if line.find('Visiting team offensive statistics') != -1:
            line = fp.readline()
            while line.find('Visiting team pitching statistics') == -1:
                if line[13] != ' ':
                    fields.append(line.strip().split('.')[0].split('(')[0])
                line = fp.readline()
    #+end_src

#+results[97fdb2368b66e48faa6afb8b6eff34e00f05633b]: stat-headers
- at-bats
- hits
- doubles
- triples
- homeruns
- RBI
- sacrifice hits
- sacrifice flies
- hit-by-pitch
- walks
- intentional walks
- strikeouts
- stolen bases
- caught stealing
- grounded into double plays
- awarded first on catcher's interference
- left on base
```

Figure 7. The stat-headers Python code block. This block returns a list of the names of the offensive statistics to test for correlation with attendance.

```

#+source: offensive-stats
#+headers: :exports none
#+begin_src sh :var file=raw-data
  awk '{for (x=50; x<=66; x++) { printf "%s ", $x } printf "\n" }' FS="," \
    < $file
#+end_src

#+source: attendance
#+headers: :exports none
#+begin_src sh :var file=raw-data
  awk '{ print $18 }' FS="," < $file
#+end_src

```

Figure 8. The `offensive-stats` and `attendance` shell code blocks. These blocks collect the offensive statistics and attendance from the raw data file produced by the `raw-data` code block (see Figure 6).

```

#+source: analysis
#+headers: :var headers=stat-headers :var stats=offensive-stats
#+begin_src R :var attendance=attendance :exports none
  # apply the headers to the list
  colnames(stats) <- headers

  ## The following lines are required because parsing bugs are causing
  ## corrupt data in these two rows.
  badrows <- c(141, 674)
  stats <- stats[-badrows,]
  attendance <- attendance[-badrows,]
  attendance <- as.integer(attendance)

  # perform a simple correlation of each column with the attendance
  corrln <- cor(stats, attendance)

  # return the name of the most correlated column
  rownames(corrln)[which.max(corrln)]
#+end_src

```

Figure 9. The `analysis` code block. This block uses the R statistical programming language to calculate correlations between the outputs of the `offensive-stats` and `attendance` code blocks (see Figure 7) whose values are saved into the `stats` and `attendance` variables respectively.

As this example demonstrates, commingling code and prose lets authors collect all relevant information into a single place. This practice benefits readers, who can reproduce the calculations performed in the work and also extend the analysis, possibly within Org-mode itself. For example, readers of this article can rerun the analysis for another season by simply changing the value of the `season` code block above and re-exporting the file.

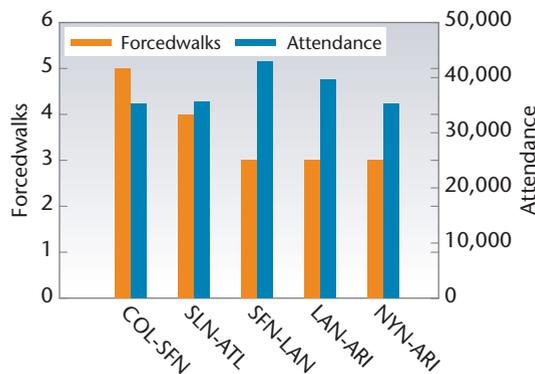


Figure 10. Forced walks and attendance for the top five games by forced walks. Results indicate that the visiting team shares the fans' belief in the effects of a large crowd.

Org-mode has many features 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.⁵ This ensures that Org-mode is now and always will be free and open source software. This directly relates to two RR goals. First, Org-mode

```

#+source: top-8
#+begin_src sh :var data=raw-data :exports none
  cat $data|awk '{print $60,$18,$7"-"$4}'
    FS=","|sed 's//g'|sort -rn |head -5
#+end_src

#+source: figure
#+begin_src gnuplot :var data=top-8 :file plot.png
:exports results
# set term tikz
# set output 'plot.tex'
set yrange [0:6]
set y2range [0:50000]
set key above
set y2tics border
set ylabel 'forced walks'
set y2label 'attendance'
set style fill pattern
set style data histogram
set style histogram clustered
set auto x
set xtic rotate by -45 scale 0
plot data using 1:xtic(3) title 'forced walks', \
  data using 2 axes xly2 title 'attendance'
#+end_src

#+label: fig:top-5
#+attr_latex: width=0.8\textwidth
#+Caption: Top 5 games by forced walks, with forced walks
  and attendance shown.
#+results: figure
[[file:plot.png]]

```

Figure 11. The code for the number of forced walks and the attendance for the five games with the most forced walks.

is available free of charge to install by any user on any system, which ensures access to the software environment required for reproduction. Second, the source code specifying Org-mode's inner workings 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 Emacs relationship. Emacs is one of the world's most widely ported pieces of software, with versions that run on all major operating systems. This ensures that Org-mode documents can be

incorporated into almost any computer work 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 can offer its users a comfortable and familiar editing environment for all content types. Finally, given Org-mode's implementation in the Emacs extension language, *Emacs Lisp*,⁶ users can customize Org-mode's behavior to their particular needs and support arbitrary new programming languages; Org-mode currently supports more than 30 programming languages.

Org-mode addresses many common problems in RR practice. Given

that a single Org-mode document can be used for every stage of a research project—from brainstorming, software development, and experimentation to publication—Org-mode largely relieves authors of the burden of tracking resources required for reproducing their work. Although this information volume can result in extremely large files, Org-mode documents' hierarchical folding lets users comfortably read and edit such files. The files themselves are encoded in plain text, which enhances their portability and makes them easy to integrate with version control systems, allowing for revision tracking and collaboration.⁷

Org-mode documents run the gambit from simple collections of plain-text notes, to complex laboratories housing data and analysis mechanisms, to publishing desks with facilities for displaying and exporting scientific results. There's a friendly community of Org-mode users and developers who communicate on the Org-mode mailing list (<http://lists.gnu.org/mailman/listinfo/emacs-orgmode>). By answering questions and helping each other master Org-mode's many features, this community helps to solve one of the largest hurdles posed by any RR tool—learning how to use it. 

References

1. R.M. Stallman, "Emacs the Extensible, Customizable Self-Documenting Display Editor," *ACM Sigplan Notices*, vol. 16, no. 6, 1981, pp. 147–156.
2. S. Fomel and J.F. Claerbout, "Reproducible Research," *Computing in Science & Eng.*, vol. 11, no. 1, 2009, pp. 5–7.
3. J.B. Buckheit and D.L. Donoho, "Wave-Lab and Reproducible Research," *Wavelets and Statistics*, Springer-Verlag, 1995.

4. C. Dominik et al., *The Org Mode 7 Reference Manual*, Free Software Foundation, 2010.
5. R. Stallman, "Free Software Foundation," *Encyclopedia of Computer Science*, John Wiley & Sons, 2003, pp. 732–733.
6. B. Lewis, D. LaLiberte, and R. Stallman, *GNU Emacs Lisp Reference Manual*, 3rd ed., Free Software Foundation, 2010.
7. K. Hinsien, K. Läufer, and G.K. Thiruvathukal, "Essential Tools: Version Control Systems," *Computing in Science & Eng.*, vol. 11, no. 6, 2009, pp. 84–91.

Eric Schulte is a doctoral student at the University of New Mexico, where he is a research assistant in the Adaptive Systems Lab. His research interests include the naturalization of computer software systems, both in exploring novel distributed architectures that avoid privileged points in space and time and automated program repair using evolutionary techniques. Schulte has a BA in mathematics from Kenyon College. Contact him at eschulte@cs.unm.edu.

Dan Davison is a senior scientist at Counsyl. His research interests include computational

biology, population genomics, reproducible research, and machine learning. Davison has a PhD in population genetics from the University of Chicago. Contact him at davison@ocunsyl.com.

 Selected articles and columns from IEEE Computer Society publications are also available for free at <http://ComputingNow.computer.org>.