Authoring tools to automate data sharing in scientific publishing

John Kitchin

March 14, 2016

Contents

1 Title slide 2
2 Introduction 2
3 Why is data sharing hard today? 2
4 Sharing is tedious, error prone 3
5 Our approach to this issue is integration of tools/data 3
6 Emacs + org-mode for reproducible, functional scientific documents 3
   6.1 sub-heading ................................................................. 4
      6.1.1 sub-sub-heading ...................................................... 4
7 We have used it extensively in scientific publishing 5
8 Reusing the data 6
9 Automating data embedding sharing 7
   9.1 HTML export ................................................................. 7
   9.2 PDF export ................................................................. 8
   9.3 Vanilla export ............................................................. 8
10 What makes this integration possible? 8
11 Concluding thoughts 8
1 Title slide

2 Introduction

• Data sharing is not easy
• But it is increasingly required
• And increasingly desired
  The big question is: "How do we make that happen?"
• Nobody wants more work
• We would all like to get more out of what we already do
• We are working on tools that make that happen that I will talk about today

3 Why is data sharing hard today?

• For many scientific authors data and analysis are not integrated into writing tools
  – Think about a manuscript in MS Word/LaTeX
  – Data in Excel, data files,...
  – Plots in Origin/SigmaPlot/etc.
  – Separate script files for analysis
  – all with limited interconnectivity
    * e.g. where is the data in Fig 2 of the manuscript?
    * Where is the file/script that did the data analysis?
    * How do I reuse the data in Table 2 for a new purpose? Copy and Paste?
These tools are not especially well-integrated and not easily adapted to new use cases

4 Sharing is tedious, error prone

- Sharing then becomes extra work to generate supporting information that reconstructs the effort, copies data you think you used, etc.
- Reconstruction of the work that went into the manuscript is error prone
- and tedious...
- Reuse is not much better
- Yet increasingly required, and desired

5 Our approach to this issue is integration of tools/data

- We developed an integrated set of tools that makes data/code/analysis part of the manuscript preparation/submission process could help with data sharing
- It would leverage the work we already do in scientific writing
- and provide access to reusable data/code
- This does require development of a relatively new tool chain for writing
- We have done this in Emacs + org-mode + code
- It is all open-source (http://github.com/jkitchin/jmax) and cross-platform

6 Emacs + org-mode for reproducible, functional scientific documents

- org-mode is basically a plain text markup language deeply integrated into Emacs (an editor)
- Outline mode at the core, and much much more
6.1 sub-heading

- Narrative text, equations $\int_0^1 e^x dx$, images

6.1.1 sub-sub-heading

Functional tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

3b2a6830d248f31580202ccdbadd5c49.csv:

- citations

- Integrated functional code

1 date

a9564ebc3289b7a14551baf8ad5ec60a:

RESULTS: Mon Mar 14 13:11:05 PDT 2016

- Functional links that can
  - Open document locations
  - Open mail, news, urls
  - run user-defined code in almost any language

1 import time
2 print(time.asctime())

309ec9e61913846c23962de976f11728.py:

We have used it extensively in scientific publishing

- Over dozen papers in print by my group illustrating what can be done
- \(^2\) - all experimental
• 3 - mixed experiment/computation
• 4 - computational DFT on oxides
• 5 - computational coverage/site dependence
• 6 - full sql database described in SI
• 7 - exp/computation H2/D2 exchange on CuPd
• 8 - exp/computation XPS on CuPd alloys
• 9 - 1.8 GB dataset on Zenodo
• 10 - computation DFT+U
• 11 - examples of reusing data
• 1 - examples of reusing data
• 12 - Molecular simulation

Let’s see a working example
• 13 The data is available in the SI
• The json database described in SI
• The source can be extracted from the PDF (goto line 336)

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable

8 Reusing the data
That data is human readable - and machine addressable
TiO$_2$ rutile LDA -2801.64 30.58 259.47
TiO$_2$ rutile AM05 -2733.53 31.31 233.2
TiO$_2$ rutile PBEsol -2759.29 31.22 239.76
TiO$_2$ rutile PBE -2773.21 32.11 215.78

089d3777c582b5e548c243b5c9fa5229.csv:
If you prefer Python, no problem. Here we get the anatase data:

```python
return [x for x in data if x[1] == 'anatase']
```

32c09a6e325db19533e2e272caed35fd.py:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TiO$_2$</td>
<td>anatase</td>
<td>LDA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2802.73</td>
</tr>
<tr>
<td>TiO$_2$</td>
<td>anatase</td>
<td>AM05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2741.12</td>
</tr>
<tr>
<td>TiO$_2$</td>
<td>anatase</td>
<td>PBEsol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2763.61</td>
</tr>
<tr>
<td>TiO$_2$</td>
<td>anatase</td>
<td>PBE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2781.16</td>
</tr>
</tbody>
</table>

6f02bbf48d7abde528dc614a9a7a84f1.csv:

9 Automating data embedding sharing

- org-mode is great - If you use org-mode
- In an Appendix there is code that automatically embeds data and code in org-mode into HTML and PDF.
- One source to many outputs
- We can extract the source code and load it here

```elisp
(org-babel-tangle)
(load-file "data-sharing.el")
```

feeccf7cb9ed36d8b57790a1ed1e6933.elisp:

9.1 HTML export

```elisp
(custom-export-and-open-html)
```
9.2 PDF export

7ebdbc39a7bc97162b6a263e0e68f7f0.elisp:

9.3 Vanilla export

\(\text{(org-open-file(org-latex-export-to-pdf))}\)

10 What makes this integration possible?

- An extensible editor
  - Extensible in a full programming language
  - This allows the tool to become what you want
  - Emacs is ideal for this
- A lightweight markup language
  - to differentiate text, code, data
  - Org-mode is also ideal for this
    * Part structured markup, part api
    * Very good compromise on function and utility with authoring ease
- \(<\text{code}>\) Since we use code to generate and analyse data, this solution works especially well

11 Concluding thoughts

- Emacs + org-mode + \(<\text{code}>\) enables a lot of very exciting capabilities in publishing and data sharing
  - Integrated narrative text, data, code
  - Export to a broad range of other formats
  - Interaction with the world (other computers, instruments) via APIs
    * Materials Project, translation, Internet of Things, ...
- The future is very exciting
- We are not waiting for someone to figure out what we want
  - Anyway, by the time they deliver it we will need something else ;)

12 Extract the references

```
(save-window-excursion
(save-restriction
(widen)
(org-ref-bibliography)
(buffer-string)))
```

2. cite:hallenbeck-2013-effec-o2 Hallenbeck, Alexander P. and Kitchin, John R., "Effects of \ce{O_2} and \ce{SO_2} on the \ce{CuO} Form of a Primary-Amine Based Polymeric \ce{CO_2} Sorbent", Industrial & Engineering Chemistry Research, 52:10788-10794 (2013)
3. cite:miller-2014-simul-temper Spencer D. Miller and Vladimir V. Pushkarev and Andrew J. Gellman and John R. Kitchin, "Simulating Temperature Dependence for Adsorption of \ce{O_2} on \ce{Cu(111)} using \textit{DFT} Derived Coverage Dependent Desorption Barriers", Topics in Catalysis, 57:106-117 (2014)

14. cite:pakin-attachfile Scott Pakin, "attachfile", , : ()

13  Getting started

Source code: http://github.com/jkitchin/jmax

Our starter-kit for Emacs + org-mode configured to do the things I showed you today Should work out of the box on Windows. Directions for using it on Mac/Linux.

Kitchingroup blog: http://kitchingroup.cheme.cmu.edu

@johnkitchin

Check out our YouTube channel: https://www.youtube.com/user/jrkitchin

1333 views (1800+ downloads of org-ref on MELPA!)

20,984 views
14 Appendix

14.1 The custom export code

Here we define a custom table exporter. We use the regular table export mechanism, but save the contents of the table as a csv file. We define exports for two backends: \LaTeX{} and HTML. For \LaTeX{}, we use the attachfile\textsuperscript{14} package to embed the data file in the PDF. For HTML, we insert a link to the data file, and a data uri link to the HTML output. We store the filename of each generated table in a global variable named \texttt{*embedded-files*} so we can create a new Info metadata entry in the exported PDF.

\begin{verbatim}
(defvar *embedded-files* '() "List of files embedded in the output."

(defun my-table-format (table contents info)
  (let* ((tblstart (org-element-property :contents-begin table))
         (tbl-data (save-excursion (goto-char tblstart) (org-babel-read-table)))
         (tblname (or (org-element-property :name table) (md5 (format "%s" tbl-data))))
         (format (elt (plist-get info :back-end) 2))
         (csv-file (concat tblname ".csv"))
         (data-uri-data))
    ;; Here we convert the table data to a csv file
    (with-temp-file csv-file (loop for row in tbl-data do
      (insert (mapconcat (lambda (x) (format "\"%s\"" x)) row)))))

    (data-uri-data))

16
17
18
19
20
21
22
\end{verbatim}
Next, we define an exporter for source blocks. We will write these to a file too, and put links to them in the exported files. We store the filename of each generated source file in a global variable named \*embedded-files* so we can create a new Info metadata entry in the exported PDF.

(table-format.elisp: 

Next, we define an exporter for source blocks. We will write these to a file too, and put links to them in the exported files. We store the filename of each generated source file in a global variable named \*embedded-files* so we can create a new Info metadata entry in the exported PDF.)

(defun my-src-block-format (src-block contents info)
"Custom export for src-blocks. Saves code in block for embedding. Provides backend-specific output."
(let* ((srcname (org-element-property :name src-block))
   (lang (org-element-property :language src-block))
   (value (org-element-property :value src-block))
   (format (elt (plist-get info :back-end) 2))
   (exts '(("python" . ".py")
           ("emacs-lisp" . ".elisp")
           ("C" . ".c")
           ("R" . ".R")))
   (fname (concat (or srcname (md5 value))
                   (cdr (assoc lang exts)))))
   (data-uri-data)
   (with-temp-file fname
     ...)}
Finally, we also modify the results of a code block so they will appear in a gray box and stand out from the text more clearly.

```lisp
(defun my-results (fixed-width contents info)
  "Transform a results block to make it more visible."
  (let ((results (org-element-property :results fixed-width))
          (format (elt (plist-get info :back-end) 2)))
    (value (org-element-property :value fixed-width)))
  (cond
    ((eq 'latex format)
     (format "\begin{tcolorbox}
              \begin{verbatim}
RESULTS: %s
\end{verbatim}
              \end{tcolorbox}")
     value))
    (t
     (format "\preRESULTS: %s\pre" value)))))
```

dafeb6b72e57a11595885a79d0ce2cbe.elisp:

```
RESULTS: my-results
```

An author may also choose to embed a file into their document, using the attachfile package for \LaTeX. Here, we leverage the ability of org-mode
to create functional links that can be exported differently for LaTeX and HTML. We will create an attachfile link, and set it up to export as a LaTeX command or as a data URI for HTML.

\begin{verbatim}
(org-add-link-type
 "attachfile"
 (lambda (path) (org-open-file path)))

;;; formatting
(lambda (path desc format)
 (cond
 ((eq format 'html)
 ;; we want a data URI to the file name
 (let* ((content
 (with-temp-buffer
 (insert-file-contents path)
 (buffer-string)))
 (data-uri
 (base64-encode-string
 (encode-coding-string content 'utf-8))))
 (add-to-list '*embedded-files* path)
 (format (concat "<a href="data:;base64,"%s">%s</a>"
 data-uri
 path)))
 ((eq format 'latex)
 ;; write out the latex command
 (add-to-list '*embedded-files* path)
 (format \"attachfile\(%s\)\" path))))
\end{verbatim}

attachfile-link.elisp:

Here, we define a derived backend for HTML and LaTeX export. These are identical to the standard export backends, except for the modified behavior of the table and src-block elements.

\begin{verbatim}
(org-export-define-derived-backend 'my-html 'html
 :translate-alist '((table . my-table-format)
 (src-block . my-src-block-format)
 (fixed-width . my-results)))

(org-export-define-derived-backend 'my-latex 'latex
 :translate-alist '((table . my-table-format)
 (src-block . my-src-block-format)
 (fixed-width . my-results)))
\end{verbatim}

2efb34a32a9c4653ff697c1d00fd294b.elisp:

\begin{verbatim}
(defun custom-export-and-open-html ()
 "Use my-html custom exporter and open the file."
 (let* ((base (file-name-nondirectory

\end{verbatim}
(defun custom-export-and-open-pdf ()
  "Use my-latex custom exporter and open pdf."
  (save-restriction
    (widen)
    (let* ((org-latex-image-default-width "")
            (*embedded-files* '())
            (base (file-name-nondirectory
                   (file-name-sans-extension (buffer-file-name))))
            (tex (concat base "\.tex"))
            (pdf (concat base "\.pdf"))
            (org-latex-minted-options
             (append
              org-latex-minted-options
              '(("xleftmargin" \parindent"\parindent")))))
          (org-export-to-file 'my-latex tex)
          (ox-manuscript-latex-pdf-process tex)
          (org-open-file pdf)))
)

References


