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**

12 Extract the references **9**

13 Getting started **10**

14 Appendix **11**
14.1 The custom export code 11

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

a	b
2	4
0	1

3b2a6830d248f31580202ccdbadd5c49.csv: 

- citations[1]
- Integrated functional code

```
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

```
import time
print(time.asctime())
```

309ec9e61913846c23962de976f11728.py: 

```
RESULTS: Mon Mar 14 13:11:36 2016
```

```
(current-time-string)
```

7f7bac1b1e471114e487e90437582fbd.elisp: 

```
RESULTS: Mon Mar 14 13:11:45 2016
```

```
Sys.time()
```

3f06a7efae851f9690a285a46ca320dc.R: 

```
RESULTS: [1] "2016-03-14 13:11:51 PDT"
```

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

int main() {
    time_t current_time;
    char* c_time_string;
    current_time = time(NULL);
    /* Convert to local time format. */
    c_time_string = ctime(&current_time);
    printf("%s", c_time_string);
    return 0;
}
```

812a422b1846832bae23932b69c07e4c.c: 

```
RESULTS: Mon Mar 14 13:11:56 2016
```

7 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)

```
pdftk am4059149_si_001.pdf  unpack_files
```

3a4c9126c7b0a1737bd6a65d910ca25a: 

That SI file was generated here (goto line 336): supporting-information.org

8 Reusing the data

That data is human readable - and machine addressable

```
(remove-if-not (lambda (x) (string= "rutile" (nth 1 x))) data)
```

1f85d0194e997e7ea8f170bf787a8598.elisp: 

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:

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

32c09a6e325db19533e2e272caed35fd.py: 

TiO_2	anatase	LDA	-2802.73	33.62	187.4
TiO_2	anatase	AM05	-2741.12	34.33	178.26
TiO_2	anatase	PBEsol	-2763.61	34.25	178.71
TiO_2	anatase	PBE	-2781.16	35.13	171.42

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

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

feeccf7cb9ed36d8b57790a1ed1e6933.elisp: 

9.1 HTML export

```
(custom-export-and-open-html)
```

94b98fc6d9a07d6041b0d18ad27acb59.elisp: 

9.2 PDF export

```
1  (custom-export-and-open-pdf)
```

7ebdbc39a7bc97162b6a263e0e68f7f0.elisp: 

9.3 Vanilla export

```
(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
- </code> Since we use code to generate and analyse data, this solution works especially well

11 Concluding thoughts

- Emacs + org-mode + </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)))
```

4e0979a1b8eb6bf328e800e4164ab2c7.elisp: 

1. cite:kitchin-2015-examp Kitchin, John R., "Examples of Effective Data Sharing in
2. cite:hallenbeck-2013-effec-o2 Hallenbeck, Alexander P. and Kitchin, John R., "Ef
3. cite:miller-2014-simul-temper Spencer D. Miller and Vladimir V. Pushkarev and An
4. cite:xu-2014-relat Zhongnan Xu and John R. Kitchin, "Relating the Electronic Str
5. cite:xu-2014-probin-cover Zhongnan Xu and John R. Kitchin, "Probing the Coverage
6. cite:curnan-2014-effec-concen Curnan, Matthew T. and Kitchin, John R., "Effects
7. cite:boes-2015-estim-bulk Jacob R. Boes and Gamze Gumuslu and James B. Miller an
8. cite:boes-2015-core-cu Jacob R. Boes and Peter Kondratyuk and Chunrong Yin and J
9. cite:xu-2015-linear-respon Xu, Zhongnan and Rossmeisl, Jan and Kitchin, John R.,
10. cite:xu-2015-accur-u Xu, Zhongnan and Joshi, Yogesh V. and Raman, Sumathy and K
11. cite:kitchin-2015-data-surfac-scien John R. Kitchin, "Data Sharing in Surface Sc
12. cite:boes-2016-neural-networ Jacob R. Boes and Mitchell C. Groenenboom and John

13. cite:mehta-2015-ident-poten Prateek Mehta and Paul A. Salvador and John R. Kitch

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

13 Getting started

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

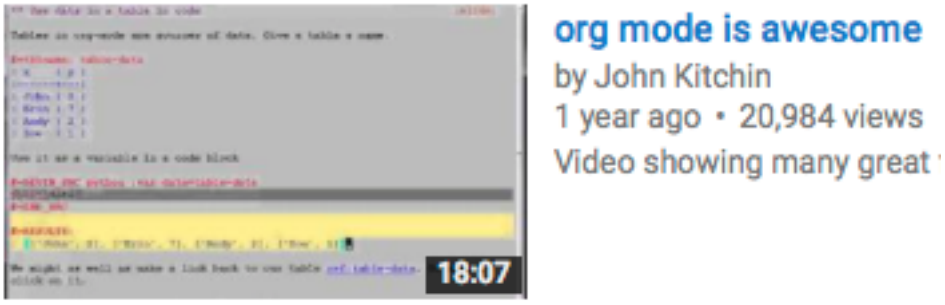
Our starter-kit for Emacs + org-mode configured to do the things I showed you today Should work out of the box on Windows. Directions for using it on Mac/Linux.

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

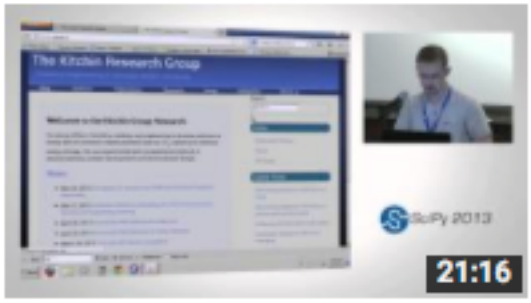
@johnkitchin

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

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



20,984 views



Emacs + org-mode + python in reproducib
by Enthought
2 years ago • 23,094 views
Authors: Kitchin, John Carnegie Mellon University
...

23,094 views!

This talk: https://github.com/jkitchin/ACS-2016-data-sharing

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[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 `*embedded-files*` so we can create a new Info metadata entry in the exported PDF.

```
(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-del-hlines
                     (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
               ", " ))
             (insert "\n"))
       (setq data-uri-data
             (base64-encode-string
              (buffer-string))))

     (add-to-list '*embedded-files* csv-file)

     (cond
      ;; HTML export
      ((eq format 'html)
       (concat
        (org-html-table table contents info)
        (format "<a href=\"%s\">%s</a>"
                csv-file csv-file)
        " "
        (format (concat  "<a href=\"data:text/csv;"
                         "charset=US-ASCII;"
                         "base64,%s\">data uri</a>")
                data-uri-data)))
      ;; LaTeX/PDF export
      ((eq format 'latex)
       (concat
        (org-latex-table table contents info)
        "\n"
        (format "%s: \\attachfile{%s}"
                csv-file csv-file))))))
```

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
```

```
        (insert value)
        (setq data-uri-data (base64-encode-string
                             (buffer-string))))

      (add-to-list '*embedded-files* fname)

      (cond
       ;; HTML export
       ((eq format 'html)
        (concat
         (org-html-src-block src-block contents info)
         (format "<a href=\"%s\">%s</a>" fname fname)
         " "
         (format (concat "<a href=\"data:text/%s;"
                         "charset=US-ASCII;base64,"
                         "%s\">code uri</a>")
                 lang data-uri-data)))
       ;; LaTeX/PDF export
       ((eq format 'latex)
        (concat
         (org-latex-src-block src-block contents info)
         "\n"
         (format "%s: \\attachfile{%s}" fname fname))))))
```

src-block-format.elisp:

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.

```
(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 "<pre>RESULTS: %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.

```
(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)))))
```

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.

```
(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)))
```

2efb34a32a9c4653ff697c1d00fd294b.elisp: 

```
(defun custom-export-and-open-html ()
  "Use my-html custom exporter and open the file."
  (let* ((base (file-name-nondirectory
```

```
                (file-name-sans-extension (buffer-file-name))))
          (html (concat base ".html")))
     (save-restriction (widen)
                       (browse-url (org-export-to-file 'my-html html)))))

(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")))))
      (org-export-to-file 'my-latex tex)
      (ox-manuscript-latex-pdf-process tex)
      (org-open-file pdf))))
```

209ed2ce9b5918b18e58d1c9eb91a1c2.elisp: 

References

[1] John R. Kitchin. Examples of effective data sharing in scientific publishing. *ACS Catalysis*, 5(6):3894–3899, 2015. doi: 10.1021/acscatal.5b00538. URL http://dx.doi.org/10.1021/acscatal.5b00538.

[2] Alexander P. Hallenbeck and John R. Kitchin. Effects of O_2 and SO_2 on the capture capacity of a primary-amine based polymeric CO_2 sorbent. *Industrial & Engineering Chemistry Research*, 52(31):10788–10794, 2013. doi: 10.1021/ie400582a. URL http://pubs.acs.org/doi/abs/10.1021/ie400582a.

[3] Spencer D. Miller, Vladimir V. Pushkarev, Andrew J. Gellman, and John R. Kitchin. Simulating temperature programmed desorption of oxygen on Pt(111) using DFT derived coverage dependent desorption barriers. *Topics in Catalysis*, 57(1-4):106–117, 2014. doi: 10.1007/s11244-013-0166-3. URL http://dx.doi.org/10.1007/s11244-013-0166-3.

[4] Zhongnan Xu and John R. Kitchin. Relating the electronic structure and reactivity of the 3d transition metal monoxide surfaces. *Catalysis Communications*, 52:60–64, 2014. ISSN 1566-7367. doi:

10.1016/j.catcom.2013.10.028. URL http://dx.doi.org/10.1016/j.catcom.2013.10.028.

[5] Zhongnan Xu and John R. Kitchin. Probing the coverage dependence of site and adsorbate configurational correlations on (111) surfaces of late transition metals. *J. Phys. Chem. C*, 118(44):25597–25602, 2014. doi: 10.1021/jp508805h. URL http://dx.doi.org/10.1021/jp508805h.

[6] Matthew T. Curnan and John R. Kitchin. Effects of concentration, crystal structure, magnetism, and electronic structure method on first-principles oxygen vacancy formation energy trends in perovskites. *The Journal of Physical Chemistry C*, 118(49):28776–28790, 2014. doi: 10.1021/jp507957n. URL http://dx.doi.org/10.1021/jp507957n.

[7] Jacob R. Boes, Gamze Gumuslu, James B. Miller, Andrew J. Gellman, and John R. Kitchin. Estimating bulk-composition-dependent H_2 adsorption energies on Cu_xPd_{1-x} alloy (111) surfaces. *ACS Catalysis*, 5: 1020–1026, 2015. doi: 10.1021/cs501585k. URL http://dx.doi.org/10.1021/cs501585k.

[8] Jacob R. Boes, Peter Kondratyuk, Chunrong Yin, James B. Miller, Andrew J. Gellman, and John R. Kitchin. Core level shifts in Cu-Pd alloys as a function of bulk composition and structure. *Surface Science*, 640:127–132, 2015. ISSN 0039-6028. doi: 10.1016/j.susc.2015.02.011. URL http://dx.doi.org/10.1016/j.susc.2015.02.011.

[9] Zhongnan Xu, Jan Rossmeisl, and John R. Kitchin. A linear response DFT+U study of trends in the oxygen evolution activity of transition metal rutile dioxides. *The Journal of Physical Chemistry C*, 119(9): 4827–4833, 2015. doi: 10.1021/jp511426q. URL http://dx.doi.org/10.1021/jp511426q.

[10] Zhongnan Xu, Yogesh V. Joshi, Sumathy Raman, and John R. Kitchin. Accurate electronic and chemical properties of 3d transition metal oxides using a calculated linear response U and a DFT + U(V) method. *The Journal of Chemical Physics*, 142(14):144701, 2015. doi: 10.1063/1.4916823. URL http://scitation.aip.org/content/aip/journal/jcp/142/14/10.1063/1.4916823.

[11] John R. Kitchin. Data sharing in surface science. *Surface Science*, N/A:in press, 2015. ISSN 0039-6028. doi: 10.1016/j.susc.2015.05.007. URL http://www.sciencedirect.com/science/article/pii/S0039602815001326.

[12] Jacob R. Boes, Mitchell C. Groenenboom, John A. Keith, and John R. Kitchin. Neural network and reaxff comparison for Au properties. *Accepted 1/2016, Int. J. Quantum Chemistry*, 2016.

[13] Prateek Mehta, Paul A. Salvador, and John R. Kitchin. Identifying potential BO_2 oxide polymorphs for epitaxial growth candidates. *ACS Appl. Mater. Interfaces*, 6(5):3630–3639, 2015. doi: 10.1021/am4059149. URL http://dx.doi.org/10.1021/am4059149.

[14] Scott Pakin. attachfile. http://www.ctan.org/tex-archive/macros/latex/contrib/attachfile. v1.5b.