Simplifying LaTeX with ORG-mode in Emacs

Emmanuele F. Somma

Abstract

In writing academic or technical articles, you can reduce complexity by using markup or configurations, dropping the LaTeX *markup language*, and adopting ORG-mode in Emacs while retaining LaTeX's high typographical quality. This article shows how `ORG` markup exported to LaTeX works. Also, it explains the attributes of the most common elements of academic papers, such as links, tables, notes, figures, and bibliographic references. Finally, in the appendix, as a practical project, the author's choices for delivering the original article following the typographical rules of ArsTeXnica are documented.

1 Introduction

`ORG` and *Markdown* are *Lightweight Markup Languages* (or LMLs) for creating documents with a well-designed layout, using only a text editor. ORG-mode, or, as the full name says, "Carsten's[1] outline-mode for keeping track of everything", is a plain text environment for recording whatever is useful in a researcher's ordinary work process (agendas, to-do lists, scattered notes, mind maps, etc.), but its most interesting feature is that it can *export* the content of text files to many typesetting systems through translation *backends*. LaTeX is one of them.

The *Markdown* language was created by John Gruber and Aaron Swartz in 2004 [7], as an evolution of the conventions already used in writing simple texts, such as e-mail messages and newsletters, online forums, wikis, and short pieces of plain text documentation. Commonly considered the progenitor, or at least the archetype, of LMLs, it is certainly the one that has been able to get the highest number of online citations; however, it was in fact developed *after* some previous formulations of light markups and *together* with many others, almost similar. It had the merit of streamlining and extending the disparate features usually present in an LML and making documents more accessible, but also the defect of being implemented in many slightly incompatible dialects [12, 14, 22, 23].

Other LMLs, *AsciiDoc*, *Textile*, *reStructuredText*, *txt2tags*, and *WikiText*, the language of Wikipedia pages, i.e., the most common LML dialects, all have an almost contemporary genesis since the early 2000s. ORG-mode is also part of this group: it was first released in 2003. All were preceded by languages with fewer capabilities, such as *BBcode*, *setext*, and *POD*, which date back to the mid-1990s.

In the wake of the success of *Markdown*, interest in LMLs has grown, and there are now many tools based on this approach. The main characteristic of these languages is simplicity of expression, to the detriment of the variety and completeness of functionality. Their use has not gone beyond the limits of editing short and poorly structured texts for online pages. Wikipedia pages are the best-known examples; they are written with the specialized LML *WikiText* mentioned above, similar to Markdown.

ORG-mode is an exception: it allows broader and more structured elaborations and therefore deserves a presentation even to LaTeX users, who are used to producing high-quality texts.

The experience of using an LML, in the workflow of an editorial staff and then in the complete typesetting of a magazine based on LaTeX, has already been told in 2009 in the pages of ArsTeXnica [19]. In that case, a 2002 LML called `BHL` (*Brute to HTML and LaTeX*) by Bastien Guerry was adopted, a direct ancestor of the modern ORG-mode. At first, `BHL` was adopted for editing articles exported in OpenOffice format, in communication between the editorial and the typographical staffs of LinuxMagazine. Later, it was extended and renamed to `TCHL`, to include functions useful for drafting technical articles in a new editorial initiative, where it was dropped into a new workflow completely based on LaTeX.

However, the needs of a technology business magazine are less than those of an academic article, which is broader and more structured, with footnotes, bibliographic references, inline mathematics, and numbered equations, as well as management of multiple languages and possibly different alphabets, and greater variability and complexity of floating elements, such as figures, boxes, and tables.

The `BHL` and Markdown languages in general are not suitable for these uses. It is essential to turn to ad-hoc typesetting programs and, in the case of markup languages, to more sophisticated languages, such as precisely LaTeX or different versions of XML and SGML, at the cost of greater verbosity of the language and difficulty in managing the whole process, or with the need to adopt user interfaces that are not always easy to use and possibly expensive.

First published in *ArsTeXnica* #34, pp. 21–48 (`guitex.org/home/en/numero-34-aprile-2023`); translation by Carla Maggi. Published by permission.

[1] Carsten Dominik is Professor of Astronomy and director of the Anton Pannekoek Institute for Astronomy of the Faculty of Natural Sciences at the University of Amsterdam. He is also the author of the popular LaTeX reference management package called `RefTeX`.

Leveraging the benefits of both approaches, the ORG-mode language can be integrated with LaTeX to produce high-quality documents.

2 Brief comparison among markups

To give just three meaningful examples of the relative simplicity of ORG-mode, let us consider how to implement: a) an italicized word, b) an environment for a direct quote, and c) a table.

In the case of italics, in LaTeX we use a markup such as

```
\emph{in italics}
```

using a keyword (`'emph'`) and 6 different characters. With ORG-mode we follow an ordinary text syntax and it is sufficient to use the same `'/'` character twice:

```
/in italics/
```

For quotations in display mode, in LaTeX it is possible to use the `quote` or `quotation` environment:

```
\begin{quote}
« Lorem ipsum dolor sit amet, consectetur [...]
\end{quote}
```

In ORG-mode, the *block* `quote` is used.

```
#+BEGIN_QUOTE
« Lorem ipsum dolor sit amet, consectetur [...]
#+END_QUOTE
```

To write a quote in a language with a non-Latin alphabet (e.g. Greek), it is sufficient to use a couple of *attributes* to specify the language, namely `':environment foreigndisplayquote'` and `':options {greek}'`, and the right environment in the preamble:

```
#+LATEX_HEADER:\usepackage[autostyle=true]
#+LATEX_HEADER:           {csquotes}
```

to get something like:

ἀγεωμέτρητος μηδεὶς εἰσίτω.

Lastly, for the typesetting of tables in LaTeX, between keywords and specification characters, seven different elements are used:

```
\begin{center}
\begin{tabular}{lll}
Header 1 & Header 2 & Header 3\\
\hline
Cell 11 & Cell 12 & Cell 13\\
Cell 21 & Cell 22 & Cell 23\\
Cell 31 & Cell 32 & Cell 33\\
\end{tabular}
\end{center}
```

whereas the same table in ORG-mode notation is much more parsimonious:[2]

```
| Header 1 | Header 2 | Header 3 |
|----------+----------+----------|
| Cell 11  | Cell 12  | Cell 13  |
| Cell 21  | Cell 22  | Cell 23  |
| Cell 31  | Cell 32  | Cell 33  |
```

in both cases, the result is the same:

Header 1	Header 2	Header 3
Cell 11	Cell 12	Cell 13
Cell 21	Cell 22	Cell 23
Cell 31	Cell 32	Cell 33

`ORG` markup is thus much more concise than LaTeX markup, and many aspects can be modified in the export of individual elements, making it possible to cover many, if not all, of the most common use cases in the writing of an academic article.

3 Integration with Emacs

The comparison between ORG-mode and LaTeX can proceed along two lines: a) comparing the markup language, as just done and as we'll see more in detail from the next section onwards, and b) comparing the integration with the Emacs editor [20].

LaTeX does not enforce the use of any specific editor but this is not the case for ORG-mode, which is closely related to Emacs. There are, it is true, alternative implementations,[3] but from the point of view of the production of most academic papers ORG-mode and Emacs are still one and the same.

On the other hand, it is also true that the *Emacs modes* for editing LaTeX files (and groups of files) and `ORG` are both equally powerful. Any comparison on this point would only involve design choices and implementation details of one or the other. As Emacs users well know, this editor is particularly effective at drafting texts for the various markup languages, as well as for programming ones. We cannot boast of any substantial difference in the use of ORG-mode compared to Emacs's LaTeX-mode [1, 3].

[2] It should also be noted that ORG-mode has a table editing environment that allows you to interact with these text elements as if they were in a spreadsheet, including the calculation of fields using formulas.

[3] See the extensive list of systems given in [16]. To give a single example, visualization of `ORG` markup as web pages has been integrated into Github, which has thus become a convenient tool for online publications using `ORG` markup without any mediation by Emacs.

Nevertheless, they are related: to use the ORG-mode pagination subsystem for the production of PDF documents, LaTeX still needs to be correctly installed and configured. Emacs, and consequently ORG-mode, can adapt, practically without intervention, to a well-made LaTeX installation, such as those now obtained from the common distributions.[4] This usually results in having the `ORG` system for LaTeX immediately available after the successful installation of LaTeX and Emacs.

Even without an installed LaTeX system, it is always possible to produce a typeset document by exporting the text of the `ORG` file in other formats (such as ODT, HTML, or plain text), and from these, discounting the lower quality in typesetting, a PDF.

Text files with the `ORG` markup are identified by the extension '`.org`'. As usual, when Emacs loads a file with a known extension, it starts interacting with the appropriate *major mode* (that's where the name ORG-mode comes from: it's the *major mode* by which Emacs interacts with `.org` files).

A major mode in Emacs activates one or more ad hoc menus and a series of specific *key combinations*, which can be used only when editing a file with that extension. For example, when editing a `.org` file there will be an '`Org`' menu, and when the cursor is inside a table, a further menu for editing tables will also appear, for adding columns, rows, and other operations, as mentioned earlier.[2]

In ORG-mode some interactive functions and related key combinations are fundamental in the work process. For example, `C-c C-e` (the two keystrokes CTRL-C and CTRL-E; the abbreviation to `C-` is customary in Emacs), which corresponds to the command `M-x org-export-dispatch`, opens the panel where the text can be exported in various output formats. From here you choose the export backend and launch the translation operation; for example, the combination '`l p`' creates a PDF file.

Furthermore, `C-u C-c C-e` repeats the last used typesetting command. After the first use of the previous panel, where the desired export is selected, `C-u C-c C-e` becomes the most used key combination in the typesetting phase.

The complete combination `C-c C-e l p` (`M-x org-latex-export-to-pdf`) transforms `.org` source into LaTeX and then typesets it to PDF using pdfLaTeX (or the user's chosen program). If, instead, you use `C-c C-e l o`, at the end of the compilation the system viewer will also open to show the typeset result in PDF.

[4] See also [5].

Thus, if properly configured, ORG-mode performs all the necessary, and possibly multiple, LaTeX compilations to correctly obtain indexes, bibliographies, and references, without the user having to do anything else.

You may find it useful to customize the PDF compilation process with the `latexmk` program (or similar) by specifically configuring the Emacs variable `org-latex-pdf-process` in `~/.emacs`, as in:

```
(setq org-latex-pdf-process (list
  "latexmk -shell-escape -bibtex -f -pdf %f"))
```

Another useful key combination is `C-c C-`; this opens a panel where you can choose a block type to insert (for example, you can use `e` for an *example*, `s` for a program listing in *source* code, `q` for a *quote*, `v` for poetic *verse*, etc.). Many other key combinations are defined to perform specific tasks, such as inserting links (`C-c C-l`), footnotes (`C-c C-x f`), quotes (`C-c C-x [`), and so on.

4 ORG for LaTeX

To obtain a LaTeX file starting from a `.org`, there is no need for any particular operations other than exporting to the `.tex` source file with `C-c C-e l l`.

With ORG-mode, the exported LaTeX document will be an article (class `article`) by default. You can configure this export in a more refined way.

There are two possibilities for configuring ORG-mode operations: a) through the Emacs customization (or configuration) mechanism,[5] which applies to all `ORG` documents, or b) with the definition of *directives*[6] in the `.org` file being processed, which obviously only apply to the file being processed.

[5] In Emacs, we make a distinction between *configuration* and *customization*, although the goal and the final result may be indistinguishable. By *configuration* we mean the introduction in the Emacs initialization file (usually `~/.emacs`) of commands in the Lisp programming language, usually very simple, for activating the packages and defining the *customization variables*. By *customization*, instead, we mean the use of the hierarchical structure of the customization panels of the `Options->Customize` menu to define the same *customization variables* and, in turn, generate a structure within the `.emacs` file based on the `custom-set-variable` Lisp function. The two options have equivalent results; adopting one or the other form, or a mix between the two, is a matter of user preference.

[6] In this article, to make the difference more understandable, we adopt the (uncommon) convention of naming as *directives* the configuration lines present inside the `.org` file (which the documentation usually calls *variables* or `ORG` *commands*), reserving the wording *customization variables* (or simply *Emacs variables*) for the variables at the editor level. Directives will always be represented between the characters `#+` and a `:`, which are necessary for their definition and will always be capitalized, although this is not strictly necessary. To make clear the distinction between *customization variables*, which can be configured by the user, and Emacs *interactive functions*, i.e. commands that can be executed by the user,

The first strategy is suitable for changes to the whole work environment, and is based on the customization mechanism of the `Options->Customize->Group->Specific Group` menu, indicating `org` as a group, or with the command `M-x customize-group RET org`. Alternatively, special Lisp command lines can be introduced in the `~/.emacs` file.

In contrast, in the second case, the configurations can be indicated directly in the `.org` file using `ORG` markup *directives*, which have the form:

```
#+<KEY>: <VALUE>
```

The key is case-insensitive.

Thus, the same effect as changing the variable `org-latex-default-class`, which defines the LaTeX class to be used in the `\documentclass` command at the beginning of the `.tex` file, is obtained using the `#+LATEX_CLASS:` directive in the `.org` file:

```
#+LATEX_CLASS: report
```

If you want to add options to the class, the directive `#+LATEX_CLASS_OPTIONS:` is used:

```
#+LATEX_CLASS_OPTIONS: [12pt,oneside]
```

The `#+LATEX_COMPILER:` directive specifies the compiler to employ.

```
#+LATEX_COMPILER: xetex
```

Lastly, one or more `#+LATEX_HEADER:` directives can be used to include extra lines in the preamble of your LaTeX document. They provide a convenient way to add more packages without having to configure `org-latex-default-packages-alist`, and also to possibly define new commands or particular configurations for LaTeX.

The conversion from `ORG` to LaTeX depends on the `org-latex-classes` variable, which is defined as follows:

```
(("article" "\\documentclass[11pt]{article}"
  ("\\section{%s}"       . "\\section*{%s}")
  ("\\subsection{%s}"    . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
  ("\\paragraph{%s}"     . "\\paragraph*{%s}")
  ("\\subparagraph{%s}"  . "\\subparagraph*{%s}"))
 ("report" "\\documentclass[11pt]{report}"
  ("\\part{%s}"          . "\\part*{%s}")
  ("\\chapter{%s}"       . "\\chapter*{%s}")
  ("\\section{%s}"       . "\\section*{%s}")
  ("\\subsection{%s}"    . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
 ("book" "\\documentclass[11pt]{book}"
  ("\\part{%s}"          . "\\part*{%s}")
  ("\\chapter{%s}"       . "\\chapter*{%s}")
  ("\\section{%s}"       . "\\section*{%s}")
  ("\\subsection{%s}"    . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}")))
```

the name of the latter will always be prefixed with the key combination `M-x` necessary to activate them.

Each element of this list, defined as:

```
(class-name preamble-start
            segmentation-elements)
```

represents one of the usual classes of LaTeX documents (`article`, `book`, and `report`).

The *segmentation elements* of the definition are pairs of titling commands, the first to be used in case of numbered titles, and the second for unnumbered ones. As you can see, for `article` the segmentation starts from the `section`, while for `book` and `report` it starts from `part`, but this can clearly be changed.

When exporting to LaTeX, the backend will apply, before anything else, the class defined globally in `org-latex-default-class` or in the document with the directive `#+LATEX_CLASS:`, whose options are in `org-latex-default-class-options` or with the directive `#+LATEX_CLASS_OPTIONS:`. The default packages, which are listed in the variable `org-latex-default-packages-alist` are inserted later; they can be omitted using the special string `[NO-DEFAULT-PACKAGES]` in `preamble-start` of the class defined in `org-latex-classes`.

Other packages, given in `org-latex-packages-alist`, are also appended later, and can be omitted with `[NO-PACKAGES]`. Finally, the `EXTRA` text will be added, i.e. the lines defined in the `.org` file with the `#+LATEX_HEADER:` directives. This set can also be excluded using `[NO-EXTRA]`. The preamble of an article for ArsTeXnica is shown in the appendix.

The `org-latex-default-packages-alist` variable includes `inputenc`, `fontenc`, `hyperref`, and other packages that are needed for various ORG-mode functionalities. It should not be overridden.

To work with a different LaTeX class, for example, `arstexnica`, the class used for this article (in the Italian original), you will need to properly configure `org-latex-classes`, as shown in the appendix.

Whenever possible, it is preferable to act at the level of the `.org` file, to keep the compilation needs documented in the file itself and not depend on a general configuration that could change over time or between different users.

Only those directives dedicated to LaTeX are indicated, but many other generic directives affect the export. For example, document descriptive directives such as:

```
#+TITLE: Emacs ORG-Mode, LaTeX (and ArsTeXnica)
#+AUTHOR: Emmanuele F. Somma
#+CREATOR: Emacs 28.1 (Org mode 9.5.2)
#+DESCRIPTION: An article for ArsTeXnica
#+KEYWORDS: Emacs ORG LaTeX typesetting
```

are appropriately used by the export backend:

```
\author{Emmanuele F. Somma}
\date{\today}
\title{Emacs ORG-Mode, \LaTeX{} (and \ArsTeXnica{})}
\hypersetup{
 pdfauthor={Emmanuele F. Somma},
 pdftitle={Emacs ORG-Mode, LaTeX (and ArsTeXnica)},
 pdfkeywords={Emacs ORG LaTeX typesetting },
 pdfsubject={An article for ArsTeXnica },
 pdfcreator={Emacs 28.2 (Org mode 9.5.5)},
 pdflang={Italian}}
```

Note also the definition of the language in which the document is written:

```
#+LANGUAGE: it
```

The supported languages are in the variable `org-latex-babel-language-alist`. However, the language definition does not automatically include the related LaTeX package, so that the user can choose whether to use `babel` or `polyglossia`. This can be defined in the general list of default packages with the Lisp command:

```
(add-to-list 'org-latex-packages-alist
  '("AUTO" "babel"
    t ("pdflatex" "xelatex" "lualatex")))
```

or

```
(add-to-list 'org-latex-packages-alist
  '("AUTO" "polyglossia"
    t ("xelatex" "lualatex")))
```

where `AUTO` indicates the language chosen in the `#+LANGUAGE:` directive. As you can see, they can be differentiated by typesetting engine.

Alternatively, the language package can be inserted explicitly in the `.org` file with a line like:

```
#+LATEX_HEADER: \usepackage[italian,greek]{babel}
```

5 Document options

Some `ORG` document options affect LaTeX export. Options are defined in the format `key:value` and inserted in the directive named `#+OPTIONS:`.[7] Here is a lengthy example (we won't describe them all here; see the documentation):

```
#+OPTIONS: ':nil *:t -:t ::t <:t H:3 \n:nil ^:t
#+OPTIONS: author:t broken-links:nil creator:nil
#+OPTIONS: d:(not "LOGBOOK") date:t e:t
#+OPTIONS: email:nil f:t inline:t num:t
#+OPTIONS: prop:nil stat:t tags:t tasks:t tex:t
#+OPTIONS: timestamp:t title:t toc:t todo:t |:t
```

[7] With the key combination `C-c C-e #` or the command `M-x org-export-insert-default-template` you can insert all options into your document. By choosing `default`, you can insert the options common to all backends; with `latex`, `html`, etc. inserting those related to LaTeX, HTML, etc.

Boolean variables are *true* if set to `t`, and *false* if set to `nil`. Here are some useful options:

`num:` turns on section numbering

`toc:` typesets the table of contents[8]

`|:` turns on table typesetting

`^:` uses TeX syntax for superscripts and subscripts[9]

`-:` allows conversion of special strings

`f:` activates conversion of footnotes (indicated as `[fn:...]`)

`*:` activates text emphasis (`*` for bold, `/` for italics, `_` for underlined)

`TeX:` allows TeX macros in the text

`author:` includes author's name in the layout

`email:` includes author's email in the layout

`creator:` includes creation information in the layout

6 Element attributes

The directives and options seen so far apply to the entire export process and determine how the layout is formed. Usually, ORG-mode performs a standard layout of the elements of the LaTeX document, but this is not always sufficient for the author's purposes. It is possible to provide particular specifications for the various elements to be typeset, by defining attributes for the individual elements to be typeset (such as images, tables, etc.). Attributes are defined in `ORG` with a `#+ATTR_<backend>:` directive. For LaTeX it is `#+ATTR_LATEX:`.

For the same element, several attribute directives can be specified, each relating to different backends, to be used in the corresponding exports. And in each `#+ATTR_...` directive, many attributes can be included, in the format `:<attribute> <value>`. Element attributes should not be confused with document options, which are defined in the `#+OPTIONS:` directive, in the format `<option>:<value>`.

To give an example, if you want to introduce an image into the text flow, you can write:

```
[[./img/arstexnica.png]]
```

the following image will be obtained in the typeset text, extended to the page size:

ArsTeXnica
Rivista italiana di TeX e LaTeX

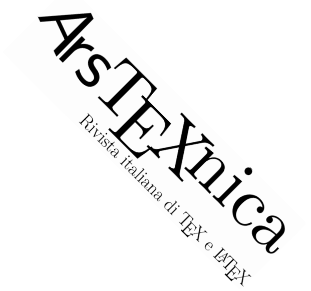
[8] The table of contents is located after the title or at the point where a line with only the string `[TABLE-OF-CONTENTS]` is inserted.

[9] Warning: it is not a boolean flag but must be set as `^:{}`; in this case, `a_{b}` is interpreted, whereas `a_b` is not.

We can define attributes for the image, such as the size or a possible rotation angle, as follows:

```
#+ATTR_LATEX: :width 4cm :options angle=45
[[./img/arstexnica.png]]
```

and you will get:



You can put the image in a floating block and give it a caption with the general command `#+CAPTION:`, which is valid for all export formats:

```
#+CAPTION: The ArsTeXnica logo
#+ATTR_LATEX: :width 4cm
[[./img/arstexnica.png]]
```

Figure 1: The ArsTEXnica logo

Many elements of the ORG markup have specific attributes, like the ones we have just seen for images. One of the most important is `:environment` which selects the environment to apply to the next element. For example:

```
#+ATTR_LATEX: :environment myverbatim
#+BEGIN_EXAMPLE
«  Lorem ipsum...
#+END_EXAMPLE
```

It will apply the `myverbatim` environment to the EXAMPLE block:

```
\begin{myverbatim}
« Lorem ipsum...
\end{myverbatim}
```

although, in this case, it would be even more appropriate to take advantage of the special syntax available to specify the block with the name of the environment:

```
#+BEGIN_myverbatim
« Lorem ipsum...
#+END_myverbatim
```

Using the attributes it is possible to specify the details of the typesetting without directly resorting to the LaTeX commands.

7 External and internal links and element identification

A URI, written verbatim in the document, or enclosed in angle brackets (`<LINK>`) or in double square brackets (`[[LINK]]`), is typeset as an active link in LaTeX. The general format of a hyperlink is:

```
[[LINK][DESCRIPTION]]
```

In this case, the description will be the typeset text, and the link will allow you to access the resource. There are precise rules to correctly represent within links the elements that need to be escaped using the character (\), such as square brackets, for example, and the escape character itself. The best way to insert and modify a URI in the text is to use the key combination provided by Emacs (`C-c C-l`) which automatically takes care of this aspect.

In addition to the external links, using all the usual schemes (`http`, `https`, `ftp`, `email`, etc.), ORG-mode also handles links *internal to the document*, such as references to figures and tables, to sections, and even to single items of a list.

An article element, such as a table or a figure, can be identified with the `#+NAME:` directive placed before the element. This name can then be used for internal references; for example:

```
#+NAME: Tab1
| Col1 | Col2 |
|------+------|
| A    | B    |

As indicated in *Table [[Tab1]]*.
```

which will be typeset like this:

Col1	Col2
A	B

As indicated in **Table 7**.

It is also possible to refer to the single items of a numbered list:

```
1. first item
2. <<p2>>second item

As indicated in item [[p2]].
```

which produces:

1. first item
2. second item

As indicated in item 2.

It is also possible to link to a section title with the notation `*Section [[*Mathematics]]*` (obtaining **Section 10**). The numbering option (`num:t`) must be active.

8 Some important block types

Various specialized blocks are delimited by the directives `#+BEGIN_<type>` and `#+END_<type>`. The `#+BEGIN_` line can also specify options for the block. Common examples:

`ABSTRACT` the `ABSTRACT` block plays a special role because it is typeset as a summary of the document (*abstract*), according to the rules of the class used in the LaTeX document. For example, the `ABSTRACT` block of this article is:

```
#+BEGIN_ABSTRACT
In writing academic ...
#+END_ABSTRACT
```

`CENTER` center the block content on the page.

`EXPORT` a particularly important block, the specification key indicates the backend system (`html`, `latex`, etc.). The content will only be included in exported files for that backend. So to copy a piece of LaTeX code from the `.org` file to the `.tex`, we write:

```
#+BEGIN_EXPORT latex
<LaTeX code> ...
#+END_EXPORT
```

`EXAMPLE` block for showing examples in `verbatim`:

```
#+BEGIN_EXAMPLE
| Header 1 | Header 2 | Header 3 |
|----------+----------+----------|
| Cell 11  | Cell 12  | Cell 13  |
| Cell 21  | Cell 22  | Cell 23  |
| Cell 31  | Cell 32  | Cell 33  |
#+END_EXAMPLE
```

`QUOTE` block for typesetting display quotations.

`SRC` block for including source code in most common programming languages, as indicated in the first option of the `begin` directive.

A relevant aspect of `ORG` documents is that these code blocks are used not only to represent an example (for which there is also the block type `EXAMPLE`) but can be made *active*. Then ORG-mode, with the support of external or internal language interpreters, can in fact execute the code of the block and insert the result of the processing in the output, using the directive `#+RESULT:` placed directly after the `SRC` block.

Alternatively, the code can be exported to external source files, and subsequently interpreted or compiled. This allows a `.org` document to become an active *literate programming notebook* in the same way as the classic `tangle` and `weave` tools, with which TeX was written by Donald Knuth [9, 10], or the more recent *notebook interface* projects like the Jupyter project [4].

For the LaTeX representation of the `SRC` block, the `verbatim` environment is used, but it is also possible to choose `listings` or `minted` by configuring the Emacs variable `org-latex-listings`.

Here's an example:

```
#+BEGIN_SRC python
def fibonacci(n):
  if n <= 1: return n
  else: return fibonacci(n-1)+fibonacci(n-2)
#+END_SRC
```

`VERSE` block for typesetting poems and verses:

```
#+BEGIN_VERSE
There once was a student of science,
Whose essays cast off in decadence
She then used Org to write,
And LaTeX to paginate,
So her colleagues admired in silence.
#+END_VERSE
```

9 LaTeX inside ORG markup

`ORG` is a format that tends to represent text content unaltered, except for markup elements placed in the expected positions or combinations. Anything not recognized as markup is transferred, as is, to the exported file.

It is possible to include text intended for the backend. For example, you could include any LaTeX command, say `\raggedright`, to get the desired LaTeX behavior (the `TeX:t` option must be defined).

Working with a single backend, and therefore not needing to export the result in any other format, there is no need to use the `EXPORT` block. If, on the other hand, you want to take advantage of ORG-mode's ability to export content in multiple formats, you need to limit the export to just the desired backend, using `EXPORT` blocks when necessary.

If you need to insert shorter fragments of LaTeX you can use an inline expression enclosed within a double pair of symbols `@` followed by `latex:`, as in `@@latex: arbitrary LaTeX code@@`. This code will only be inserted in the export for the selected backend. The same can be done for other backends:

```
...this document is typeset with
@@latex: \LaTeX{}@@@@html: HTML@@
and not with @@latex: HTML@@@@html: \LaTeX{}@@
```

Given this sentence in this article, when readers view the typeset PDF, they will see that this document is typeset with LaTeX (and not in HTML); if they see it on a web page, exported directly from ORG-mode in HTML, they would read the opposite.

If you don't need a whole block but just a single line, you can use the `#+LATEX:` directive, as in:

```
#+LATEX: \newpage
```

10 Mathematics

One of the strengths that make LaTeX supporters rightly proud is the advanced ability to typeset documents with mathematical expressions, both inline and displayed. From this point of view, `ORG` uses the most straightforward approach: the LaTeX notation for mathematics. Therefore, inside the `.org` source an expression like the following:

```
\[ \hat{y} = \hat{\beta}_{0}+\hat{\beta}_{1}
    x_{1}+\hat{\beta}_{2}x_{2}+\dots
    +\hat{\beta}_{p}x_{p} \]
```

produces:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_p x_p$$

For a numbered equation, it's enough to use an `equation` block or any other such environment usually used in LaTeX, e.g., `displaymath`, `align`, etc.

For inline math, you can also simply use the LaTeX notation, such as `\( x^{2+y} \)` to obtain x^{2+y}. From this point of view, therefore, ORG-mode does not add or subtract anything to/from LaTeX.[10]

Where the backend provides for it, the mathematics will be typeset correctly even when exported in formats other than LaTeX, for example, HTML, as you can see using the key combination `C-c C-e h O`.

11 Tables

Tables were already introduced above; we need only mention a few attributes:

`:environment` environment name for the LaTeX backend (`tabularx`, `longtable`, `array`, etc.);

`:float` typesets the table as a float, with options `sideways` or `multicolumn` to obtain it respectively rotated or on several columns;

`:mode` modifies the way the table is exported; takes values `table`, `math`, `inline-math`, `verbatim`, or `tabbing`;

`:placement` specifies the positioning of the table; it can take the values `h t b p ! H`, as in LaTeX.

Other options concern:

- export in a math environment: `:math-prefix`, `:math-suffix`, `:math-arguments`;
- the width: `:spread`, `:width`;
- switches to improve handling of the layout: `:booktabs`; `:center` for centered positioning; and `:rmlines` to remove all but the first line for tables typeset in the format expected by `table.el`.

The table format called `table.el` is an alternative Emacs format. It is a little less parsimonious than the standard `ORG` format, because you need to indicate all the intersections of the cells, but it allows you to typeset more complex tables than in the simplified ORG-mode syntax. For example, a table with merged cells, like this one:

A		B
10	11	
20	21	22

is written in the text file as:

```
+-----+-----+-----+
| A         | B   |
+-----+-----+     |
| 10  | 11  |     |
+-----+-----+-----+
| 20  | 21  | 22  |
+-----+-----+-----+
```

12 Footnotes

ORG-mode supports two ways to write footnotes.

1. In the first case, an inline label is used in the format `[fn:NAME]` and the footnote itself will be expressed as an autonomous paragraph, usually at the end of the file or of the section considered, in the format '`[fn:NAME] Footnote text.`'.[11] The definition must begin with the first square bracket in the first column of the line, with no preceding spaces, and ends at the start of a new note or section or with two blank lines.
2. The second case allows insertion of an inline footnote with the format `[fn::Inline footnote text...]` — pay attention to the double `:`. You can also use the notation `[fn:NAME: Inline text]` if you need to refer to the footnote elsewhere as well, as was the case in this article for

[10] Remember that the use of the TeX notation, `$...$` or `$$...$$`, for equations, while possible, is not recommended in LaTeX and, even more so, in ORG-mode.

[11] Footnote text.

the footnote at the end of this sentence, used already twice elsewhere.[2]

Footnotes are automatically numbered.

If you do not want to place your footnotes at the end of the page but at the end of the document, you can introduce the following LaTeX directives where you want the footnotes to appear, at the end of the document:

```
#+LATEX_HEADER: \usepackage{endnotes}
#+LATEX_HEADER: \renewcommand\footnote{\endnote}
#+LATEX: \theendnotes
```

13 Bibliographies and references

Citing, linking, and listing references is one of the fundamental tasks in writing scientific articles and, for a long time, ORG-mode did not have its own standard system for dealing with them. In recent years, this situation has changed, although in a way that may not entirely satisfy an author accustomed to LaTeX.

The most *modern* way in `ORG` of indicating bibliographic citations is entrusted to the `org-cite` subsystem (present in the `oc.el` library), already included in up-to-date ORG-mode distributions.

`org-cite` is based on the definition of *citation processors*, whose job is to offer several functionalities including:

- colored links and tooltips on hover;
- actions on user click in the editor;
- insertion and modification of citations;
- export to different formats.

The ORG-mode quoting mechanism, which was designed to be *back-end agnostic*, introduces a triad consisting of

```
(processor bibliographic-style citation-style)
```

The *citation style* consists of a string based on a uniform pattern, set up as follows:

```
[cite/s/v:@key1;@key2;...]
```

where the two specifiers `/s` and `/v` are optional and represent the *style* and *variant* indicators of the citation. They can be used to select a citation without author (`na` for `noauthor`), i.e. with date only, or in a variant with initial capitalization (`cf` for `caps-full`). They are equivalent to many `biblatex` citation commands, such as `\cite`, `\citetitle`, `\citeyear`, etc.

Following that, `@key1`, `@key2` specify the *citation keys*; there can be only one (in which case the final semicolon is not needed) or many, separated by semicolons. Each citation key is composed of the character `@` followed by the label of the reference used in the `.bib` file, preceded by a possible prefix, which will be typeset before the reference and followed first by a *locator*, i.e. a page, chapter or volume marker or other identifiable elements in the reference and, after a space, a suffix. These elements are optional. The general form of a citation key is therefore as follows:

```
prefix @key locator suffix
```

At the beginning of the entire citation string, there may be a prefix without a citation key, which acts as a common prefix for all citations. Likewise, there may also be a suffix common to all citations. Ultimately, a citation expressed as:

```
[citep/text/full:See;
 @hegel1807phenomenology pp. 184--6]
or
[citep:@hegel1807phenomenology]
```

could be typeset as:

> (See Hegel, The Phenomenology of Spirit (1807) pp. 184–6)
> or
> (Hegel, 1807)

Several processors have been defined, but the main variants are:

`csl` for exporting citations to all backends;

`bibtex` only for LaTeX; there is also `natbib` and `biblatex`, which differ from `bibtex` primarily in the LaTeX package loaded.

The `bibtex` processor uses LaTeX, so `[cite:@hegel1807phenomenology pp. 184--6]` becomes `\cite[184--6]{hegel1807phenomenology}`. In this case, LaTeX will also use `bibtex` (or `biber`, by user choice) to generate the citations. The downside of using this processor is that it *cannot* be used to typeset in formats other than LaTeX.

To have a universal `.org` file you need to use the `csl` processor (Citation Style Language). `CSL` is an XML-based citation and bibliography specification language. It is used by some well-known bibliographic reference management programs such as Zotero, Mendeley, and Papers, and was initially developed within the OpenOffice Bibliographic Project.[12]

The choice of one or the other approach is based on whether or not to rely on `bibtex` for the handling of LaTeX citations. If `csl` is used, both citations and bibliographic references will be composed in a plain text way, according to the chosen bibliographic standard: there will be no `\cite` citation commands

[12] `openoffice.org/bibliographic/`

in the file, nor a \bibliography to read the file created by bibtex.

Those who have mastered bibtex sufficiently, especially in selecting the citation variants or if they have to manage lesser-known fields of the .bib file (such as the crossref and related fields, used to indicate related volumes or language translations), know how powerful it is to rely on the bib(la)tex citation system. The agnostic approach to the csl backend may therefore not be adequate for sophisticated LaTeX enthusiasts.

It is also worth mentioning an alternative package, created by John Kitchin, which was originally used as a template for org-cite, called org-ref. Although it is a package that needs to be installed via M-x package-list-package, it is, still nowadays, the most used system for typesetting citations and bibliographies in LaTeX.

org-ref is comparable to the bibtex (and related) processors of org-cite but contains many features which have not yet found a place in org-cite, and maybe will never be integrated. It is generally more refined in relation to LaTeX. Unfortunately, Kitchin's attempt to integrate org-ref into org-cite, with a project called org-ref-cite, appears to have stalled. Instead, a new, even better functioning version 3.0 of org-ref has been released.

The org-ref syntax for citing is the same as using org-cite, except it uses the & character in the citation key instead of @.

Thus, if you are prepared to lose compatibility with any backends other than LaTeX, org-ref is the most effective choice for typesetting ORG-mode references and bibliographies. On the other hand, org-cite will improve over time, despite having structural limitations that Kitchin considers insurmountable.[13] Looking forward, adopting org-cite could create more maintainable documents over time.

However, org-cite refers to a specific directive #+BIBLIOGRAPHY: which indicates the bibliographic file to consider, whereas org-ref solves the problem by simply adding the suitable lines at the beginning of the LaTeX file, as in

```
#+LATEX_HEADER: \usepackage[
#+LATEX_HEADER:    citestyle=authoryear-icomp,
#+LATEX_HEADER:    bibstyle=authoryear,
#+LATEX_HEADER:    hyperref=true,backref=true,
#+LATEX_HEADER:    maxcitenames=3,url=true,
#+LATEX_HEADER:    backend=biber,natbib=true]
#+LATEX_HEADER:   {biblatex}
#+LATEX_HEADER: \addbibresource{test.bib}
```

and a \printbibliography at the end.

[13] lists.gnu.org/r/emacs-orgmode/2022-03/msg00250

14 Conclusions: why to use or not use ORG-mode

At the end of this introductory presentation to ORG-mode, we can balance the reasons for using or not using ORG markup instead of LaTeX. Using ORG-mode is beneficial for the following reasons:

1. The simpler and cleaner text format of ORG helps the readability of the source, integration in a less-demanding publishing workflow, and greater ease of archiving [17];
2. The ability to create documents that can be exported in different formats allows simultaneous publication on different media as well as simpler collaboration with co-authors who use different formats;
3. The presence of active source blocks, which can programmatically produce results, enables a research approach oriented towards *Open Science*, and the production of reproducible scientific results [11, 18, 21]. The ORG-mode solution is considerably simpler than those working directly with LaTeX [2].

On the other hand, there are also some clear shortcomings:

1. No journal or publisher accepts native .org format, so delivery of an article will necessarily have to be done in a different format (LaTeX, OpenOffice, etc.).
2. An author already competent with LaTeX, possibly already with his own chain of development tools tested, could consider the effort to obtain some modest advantage superfluous, or even counterproductive, from his point of view. (However, an author who is new to LaTeX may find it advantageous to directly learn ORG markup and how to use Emacs at the same time, thus obtaining in one fell swoop a research and publishing environment, moreover of excellent quality.)
3. ORG-mode requires that you use Emacs as your editor, which you may not necessarily like.

One goal of this article was to demonstrate in practice that *if you like* to use Emacs and ORG-mode to deliver an article in LaTeX, it is *always* possible to follow the guidelines of a magazine or a publisher. This is true for LaTeX, which is the subject of this article, but also true if it is to be delivered in a word-processor format (docx, odt, etc.).

ORG-mode represents an additional, supplementary approach, that you may add to your researcher's toolbox, and not a replacement for LaTeX.

Of one thing we can be sure: we can bet on the longevity of ORG-mode, based as it is on LaTeX and Emacs.

A Using ORG-mode with ArsTEXnica

To submit this article written in ORG-mode to ArsTEXnica,[14] it was necessary to comply with the editorial requirements; in particular, to export using the `arstexnica` class and respect the guidelines given in the kit [8].

It's one of those things that turned out to be easier to do than to document.

A.1 `arstexnica` class

The first thing to do was define `arstexnica` as the class used in the export, by indicating the directive `#+LATEX_CLASS:` in the `.org` file:

```
#+LATEX_CLASS: arstexnica
```

However, this is not enough. We need to configure `org-latex-classes` to add the `arstexnica` class; recall the triad:

```
(class-name start-of-preamble segmentation-elements)
```

The Lisp command for `~/.emacs` is therefore:

```
(with-eval-after-load 'ox-latex
 (add-to-list 'org-latex-classes
  '("arstexnica" "\\documentclass{arstexnica}"
    ("\\section{%s}" . "\\section{%s}")
    ("\\subsection{%s}" . "\\subsection{%s}")
    ("\\subsubsection{%s}" . "\\subsubsection{%s}")
    ("\\paragraph{%s}" . "\\paragraph*{%s}")
    ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
```

Already at this point, an export from `.org` allows a good view of the article. However, the produced file does not exactly reflect the example `name.tex` present in the kit. The preamble of the article generated by ORG-mode is different from that of ArsTEXnica. To make it the same, we need to configure some Emacs variables.

The main model of an ArsTEXnica article, contained in the `name.tex` file of the kit, has a fundamental difference from the ORG-mode model: it supports compilation with different typesetting engines for LaTeX, namely pdfLaTeX, XeLaTeX, or LuaLaTeX.

The code that specifies this behavior is this:

```
1   \ifbool{PDFTeX}{%
2       \usepackage[T1]{fontenc}
3       \usepackage[utf8]{inputenc}
4       \usepackage[english, italian]{babel}
5     }{\ifbool{XeTeX}{%
6       \usepackage{polyglossia}
7       \setmainlanguage[babelshorthands]{italian}
8       \PolyglossiaSetup{italian}{indentfirst=false}
9       \setotherlanguage{english}
10    }{\ifbool{LuaTeX}{%
11      \usepackage{polyglossia}
12      \setmainlanguage[babelshorthands]{italian}
13      \PolyglossiaSetup{italian}{indentfirst=false}
14      \setotherlanguage{english}
15      }{%
16      \ArsTeXnicaError\endinput
17      }
18    }
19  }
20
21  \usepackage{cochineal}
22  \ifbool{PDFTeX}{%
23      \usepackage[varqu,varl,var0]{inconsolata}
24    }{%
25      \fontspec[StylisticSet={1,2,3},
26               Scale=MatchLowercase]{inconsolata}
27  }
28 \usepackage[scale=.9,type1]{cabin}
29 \usepackage[cochineal,vvarbb]{newtxmath}
30 \usepackage[cal=boondoxo]{mathalfa}
31
32 \usepackage{microtype}
33 \usepackage{natbib}
34 \usepackage{graphicx}
35 \usepackage{hyperref}
```

Listing 1: The preamble of the template file of an article for ArsTEXnica, present in the kit for authors

In the two blocks on lines 1–19 and 21–27 three cases are given:

1. If the engine is pdfLaTeX, the packages used are `fontenc`, `inputenc`, and `babel` for language management, as well as, specified in the second part (line 25), the fixed-pitch font `inconsolata`.
2. If the engine is XeTeX or LuaTeX, `polyglossia` is used for language management. (The `fontenc` and `inputenc` packages are not needed.) Furthermore, the font `inconsolata` is loaded and its characteristics specified with the `\fontspec` command (lines 25–26).
3. If the engine is none of these three, an error is given with `\ArsTeXnicaError`, and the typesetting stops.

In the next part of the preamble (lines 28–35), other packages are loaded, of which only `graphicx` and `hyperref` are already present in the preamble generated by ORG-mode.

The code produced by ORG-mode does not handle this switch between typesetting engines. Two choices can be made:

1. Assuming that the journal only uses pdfLaTeX for typesetting, you may leave the ORG-mode configurations as they are and insert the missing packages (`cochineal`, `inconsolata`, `cabin`, `newtxmath`, `mathalfa`, `microtype`, and `natbib`)

[14] Editor's note: The original Italian article was prepared in ORG, while this translated version for *TUGboat* was not. We felt this real-world adaptation to a journal's requirement was still valuable to present.

with `#+LATEX_HEADER:` directives at the beginning of the `.org` file. It's the simpler choice.

2. We may try to completely replicate the file `name.tex`.

We will choose the second option. To achieve the desired result, we will add the entire preamble of the main ArsTEXnica file to the global configuration of the translation of the `arstexnica` class; disabling the default and additional packages in ORG-mode, but not the `[EXTRA]` ones.

In addition to this, both the matter of the double abstract, Italian/English, and the definition of the bibliographic style relating to `natbib` must be managed directly within the `.org` file.

Finally, some details will need to be resolved, such as the correct typesetting of the ArsTEXnica name and the employment of some useful macros.

So, the redefinition of the `org-latex-classes` variable becomes a bit more complex, adding a long string constant with the ArsTEXnica preamble (reformatted slightly for *TUGboat*):

```
1        (with-eval-after-load 'ox-latex
2           (add-to-list 'org-latex-classes
3                       '("arstexnica"
4                         "\\documentclass{arstexnica}
5  [NO-DEFAULT-PACKAGES]
6  [NO-PACKAGES]
7  % The following code allows the use of any type-
8  % setting engine among pdfLaTeX, XeLaTeX, or LuaLaTeX.
9  % Please, don't change the preamble unless it is
10 % strictly necessary. You can add other languages or
11 % fonts if it is required by the subject of the paper.
12 % Other customizations should be added to the files
13 % '\\jobname-package.tex' and '\\jobname-command.tex'.
14 %
15 \\ifbool{PDFTeX}{%
16     \\usepackage[T1]{fontenc}
17     \\usepackage[utf8]{inputenc}
18     \\usepackage[english, italian]{babel}
19   }{\\ifbool{XeTeX}{%
20     \\usepackage{polyglossia}
21     \\setmainlanguage[babelshorthands]{italian}
22     \\PolyglossiaSetup{italian}{indentfirst=false}
23     \\setotherlanguage{english}
24     }{\\ifbool{LuaTeX}{%
25       \\usepackage{polyglossia}
26       \\setmainlanguage[babelshorthands]{italian}
27       \\PolyglossiaSetup{italian}{indentfirst=false}
28       \\setotherlanguage{english}
29       }{%
30       \\ArsTeXnicaError\\endinput
31       }
32     }
33 }
34
35 \\usepackage{cochineal}
36 \\ifbool{PDFTeX}{%
37     \\usepackage[varqu,varl,var0]{inconsolata}
38   }{%
39     \\fontspec[StylisticSet={1,2,3},
40               Scale=MatchLowercase]{inconsolata}
41 }
42 \\usepackage[scale=.9,type1]{cabin}
43 \\usepackage[cochineal,vvarbb]{newtxmath}
44 \\usepackage[cal=boondoxo]{mathalfa}
45 \\usepackage{microtype}
46 \\usepackage{natbib}
47 \\usepackage{graphicx}
48 \\usepackage{hyperref}
49 [EXTRA]
50   "
51     ("\\section{%s}" . "\\section{%s}")
52     ("\\subsection{%s}" . "\\subsection{%s}")
53     ("\\subsubsection{%s}" . "\\subsubsection{%s}")
54     ("\\paragraph{%s}" . "\\paragraph*{%s}")
55     ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
```

In practice, the first part of the `name.tex` file has been moved here, while disabling the addition of the default packages via lines 5 and 6. Now, the generation of the `.tex` file by ORG-mode, for the `arstexnica` class, will exactly replicate the preamble of the `name.tex` file, except for what will be explained later.

A.2 Double abstract

ArsTEXnica has a double abstract, presenting the abstract in two languages, Italian and English. Once the first abstract has been defined, two strategies can be used directly inside the `.org` file. The first involves inserting everything necessary for the second abstract in English in a specific `EXPORT` block for LaTeX.

```
#+BEGIN_EXPORT latex
  \begin{otherlanguage}{} % Second language
  \begin{abstract}
  In writing academic ...
  \end{abstract}
  \end{otherlanguage}
#+END_EXPORT
```

The main drawback of this strategy is that the text inserted in the abstract will not be processed by ORG-mode but copied as is. Therefore, any markup will not be transformed, for example, emphasis or representation of the LaTeX logo, and must be expressed directly in LaTeX markup.

The alternative is not to use an `EXPORT` block, but to include the second block of abstracts between two LaTeX command lines that open and close the `otherlanguage` environment, like this:

```
#+LATEX: \begin{otherlanguage}{english}
#+BEGIN_ABSTRACT
In writing academic ...
#+END_ABSTRACT
#+LATEX: \end{otherlanguage}
```

In this case, the content of the second `ABSTRACT` block will also benefit from the `ORG` markup transformation.

A.3 Macros and ArsTEXnica, the logo

The correct typesetting of the ArsTEXnica logo is a detail that could not be overlooked. It was also the hardest thing to do.

An easier solution would have involved using the LATEX command directly in the `.org` source, or using the `#+MACRO:` directive, but this would have meant having to write in the text `'\ArsTeXnica{}` in the first case or `{{{ArsTeXnica()}}}` in the second. Horrible!

For those macros that "are often used when talking about the TEX system" [8, p. 4] like `\pkgname`, `\clsname`, `\optname`, and so on, it can be useful to define ORG-mode macros using `#+MACRO:`, as in:

```
#+MACRO: envname @@latex:\envname{$1}@@
```

and then use them with `{{{envname(verbatim)}}}`.

This approach would make sense if the aim was to obtain a document that could also be typeset in other backends besides LATEX, in which case the `ORG` macro would have to be changed by inserting alternatives for each backend. For example, to be able to handle HTML as well, we could do something like this (except all on one line):

```
#+MACRO: envname @@latex:\envname{$1}
#@@@@html:<b class="envname">$1</b>@@
```

and then define suitable CSS for the `envname` class, possibly with a common representation mode for all environment names. In this case, the increased complexity in the use of the macro would be justified.

However, this article is only intended for delivery in LATEX format to ArsTEXnica, so the best choice is to directly use LATEX macros in the `.org` source. To get an environment name like `verbatim`, we'd like to write `\envname{verbatim}`, just like in LATEX.

But for the ArsTEXnica logo, a different approach will be taken, even only to demonstrate why the open source approach and Lisp programming make Emacs a superior editor, adaptable to the user's needs down to the smallest detail [6, 13, 15].

It should be noted that the text strings `LaTeX` and `TeX` present in the `.org` source are automatically transformed by the LATEX backend into their equivalent LATEX command, i.e. `\LaTeX{}` and `\TeX{}`. We want to get the same result with `ArsTeXnica` as well.

Unfortunately, this operation is not immediate or configurable in any way, because it is performed by a non-customizable Lisp function. We have to modify the programming of the `ox-latex.el` library.

Here the flexibility of the Lisp language comes to our aid. It is an interpreted environment where a function can be replaced simply by redefining it.

With a little research in the source code of the ORG-mode libraries, it turns out that the responsible function is `org-latex-plain-text` as defined in `ox-latex.el`. Its definition is this:

```
(defun org-latex-plain-text (text info)
 "Transcode a TEXT string from Org to LaTeX.
TEXT is the string to transcode.  INFO is a plist
holding contextual information."
 (let* ((specialp (plist-get info
                            :with-special-strings))
        (output
         ;; Turn LaTeX into \LaTeX{}
         ;; and TeX into \TeX{}.
         (let ((case-fold-search nil))
           (replace-regexp-in-string
            "\\<\\(?:La\\)?TeX\\>" "\\\\\\&{}"
            ;; Protect ^, ~, %, #, &, $, _, { and }.
            ;; Also protect \. However, if
            ;; special strings are used, be careful
            ;; not to protect "\" in "\-" constructs.
            (replace-regexp-in-string
             (concat "[%$#&{}_~^]\\|\\\\"
                     (and specialp "\\([^-]\\|$\\)"))
             (lambda (m)
               (cl-case (string-to-char m)
                 (?\\ "$\\\\backslash$\\1")
                 (?~ "\\\\textasciitilde{}")
                 (?^ "\\\\^{}")
                 (t "\\\\\\&")))
             text)))))
   ;; Activate smart quotes.  Be sure to provide
   ;; original TEXT string since OUTPUT may have
   ;; been modified.
   (when (plist-get info :with-smart-quotes)
     (setq output (org-export-activate-smart-quotes
                    output :latex info text)))
   ;; Convert special strings.
   (when specialp
     (setq output (replace-regexp-in-string
                    "\\.\\.\\." "\\\\ldots{}"
                    output)))
   ;; Handle break preservation if required.
   (when (plist-get info :preserve-breaks)
     (setq output (replace-regexp-in-string
                   "\\(?:[ \t]*\\\\\\\\\\)?[ \t]*\n"
                   (concat org-latex-line-break-safe
                           "\n")
                   output nil t)))
   ;; Return value.
   output))
```

This is a translation function that transforms the basic text from `ORG` markup to LATEX, limited to some elements that cannot be expressed as they are; among other things, it translates `TeX` and `LaTeX`. So this is the right place to transform `ArsTeXnica` as well.

We need to change the regular expression on line 10 of the previous listing where the matching regular expression (`"\\<\\(?:La\\)?TeX\\>"`) captures the string `TeX`, possibly preceded by `La`, and replaces it with the replacement pattern (`"\\\&{}"`), i.e. it prefixes the recognized string with a \, followed by the string captured in the parser, denoted as `\&`, finally followed by the opening and closing braces (i.e., changing `LaTeX` to `\LaTeX{}`). The backslashes of the replacement pattern must, in turn, be protected with a backslash so the definitive pattern, present at the end of line 10 in the listing, will have as many as six backslashes in sequence: `"\\\\\\&{}"`. Besides the many parentheses, these *trains* of backslashes are one of the least appreciated features of the Emacs Lisp language.

To also match `ArsTeXnica`, we need to modify the matching regular expression in this way:

```
"\\<\\(?:La\\|Ars\\)?TeX\\(nica\\)?\\>"
```

At this point, we may as well also add the other acronyms defined in the `arsacro.sty` file included in the ArsTEXnica kit by completing the table:

String ORG-mode	Formatted with string in ORG-mode (e.g. `TeX`)	Formatted with LATEX macro (e.g. `\TeX`)
`TeX`	TEX	TEX
`LaTeX`	LATEX	LATEX
`ArsTeXnica`	ArsTEXnica	ArsTEXnica
`PCTeX`	PC TEX	PC TEX
`pcTeX`	pcTEX	pcTEX
`pdfTeX`	pdfTEX	pdfTEX
`pdfLaTeX`	pdfLATEX	pdfLATEX
`PiC`	PIC	PIC
`PiCTeX`	PICTEX	PICTEX
`plain`	`plain`	`plain`
`SliTeX`	SLITEX	SLITEX

The final regular expression will then be (except written on one line):

```
"\\<\\(?:La\\|Ars\\|pc\\|PC\\|pdf\\|pdfLa\\|PiC
\\|Sli\\)?TeX\\(nica\\)?\\|plain\\|PiC\\>"
```

The problem is that there is no way to insert this regular expression change into the `org-latex-plain-text` function; we must *completely replace* this function, by redefining it with the new regular expression, and including it for example in the initial Emacs configuration file (`~/.emacs`). Even better will be to define an Emacs *customization variable*, with a meaningful name (let's say `org-latex-plain-subs-regexp`) and insert this variable in the new function, so that it can be subsequently configurable by the user.

```
; the concat is due to TUGboat's column width.
(defcustom org-latex-plain-subs-regexp (concat
 "\\<\\(\\(?:La\\|Ars\\|pc\\|PC\\|pdf\\|pdfLa\\|PiC"
     "\\|Sli\\)?TeX\\(nica\\)?\\|plain\\|PiC\\)\\>")
 "Regular expression that recognizes strings to be
  transformed into LaTeX commands."
 :group 'org-export-latex
 :type '(string :tag "Regular Expression")
 :safe #'stringp)
```

The new function to insert in the `~/.emacs` file is therefore just the same as presented on the previous page, except line 12 now uses our new variable:

```
...
           (replace-regexp-in-string
            org-latex-plain-subs-regexp
            "\\\\\\&{}"
...
```

Now, in the `.org` source, it will be possible to use `ArsTeXnica` to obtain ArsTEXnica.

The last problem is that there is no LATEX macro called `\ArsTeXnica` since the kit only defines the command `\Ars`. At this point, as recommended by the guidelines [8, p. 5], you can add a definition (either in the preamble or directly in the `.org` file with a `#+LATEX_HEADER:` directive):

```
#+LATEX_HEADER: \let\ArsTeXnica\Ars
```

A.4 Last details: hyperref and the article environment

One of the small problems with ORG-mode's standard export algorithm is the automatic creation of the PDF file metadata definition in the `\hypersetup` command, which must not be defined for delivery to ArsTEXnica:

```
\hypersetup{
 pdfauthor={Emmanuele F. Somma},
 pdftitle={Emacs ORG-Mode, LaTeX (and ArsTeXnica)},
 pdfkeywords={Emacs ORG LaTeX typesetting },
 pdfsubject={An article for ArsTeXnica },
 pdfcreator={Emacs 28.2 (Org mode 9.5.5)},
 pdflang={Italian}}
```

The `org-latex-hyperref-template` variable must therefore be set to `nil`. It can be done also directly in the `.org` file using the `#+BIND:` directive:

```
#+BIND: org-latex-hyperref-template nil
```

Finally, all the article text must be enclosed by an `article` environment, which is specific to ArsTEXnica. To handle this, we can use two `#+LATEX:` directives, one at the beginning and one at the end of the text.

```
#+LATEX: \begin{article}
...
#+LATEX: \end{article}
```

A.5 And finally... the delivery

After all the revisions and checks, when the moment of delivery arrives, one wonders: "Now, what to deliver?"

Currently, ArsTEXnica does not accept `.org` sources: we therefore need to generate the LaTeX source with `C-c C-e l l` and include it together with all necessary additional elements, such as the images, in a zip package and finally deliver it to the editorial and peer-review process.

Done! Happy peer review... and happy reading to all.[15]

References

[1] A. Babenhauserheide. Tutorial: Writing papers for ACPD using Emacs Org-mode. `www.draketo.de/files/howto-write-for-acpd-with-emacs.pdf`, 2014.

[2] H. Bar, H. Wang. Reproducible Science with LaTeX. *Journal of Data Science*, 19(1):111–125, 2021. `arxiv.org/abs/2010.01482`, `doi.org/10.6339/21-JDS998`

[3] M. Borkowski. TEXing in Emacs. *TUGboat* 39(1), 2018. `tug.org/TUGboat/tb39-1/tb121borkowski-emacs.pdf`

[4] M. Fruchart, B. Guinhouya, et al. Jupyter Notebooks for introducing data science to novice users. *Studies in Health Technology and Informatics*, 294:823–824, 2022. `doi.org/10.3233/shti220598`

[5] M. Giordano, O. Iovino, M. Leccardi. *Guida pratica all'uso di GNU Emacs e AUCTEX*. GuIT, 2013. `github.com/GuITeX/guidaemacsauctex`

[6] B. Glickstein. *Writing GNU Emacs Extensions: Editor Customizations and Creations with Lisp*. O'Reilly Media, Inc., 1997.

[7] J. Gruber. Markdown: Syntax, 2004. `daringfireball.net/projects/markdown/syntax`

[8] GuIT. Istruzioni per gli autori, 2022. `guitex.org/home/en/for-authors`

[9] D.E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. `doi.org/10.1093/comjnl/27.2.97`

[10] D.E. Knuth, S. Levy. *The CWEB System of Structured Documentation*. Addison-Wesley Professional, 1993. `www-cs-faculty.stanford.edu/~knuth/cweb.html`

[11] A. Leha, T. Beißbarth. The Emacs Org-mode: Reproducible research and beyond. In *The R User Conference, useR! 2011 August 16-18 2011 University of Warwick, Coventry, UK*, p. 28, 2011. `www.r-project.org/conferences/useR-2011/abstracts/010411-lehaandreas.pdf`

[12] S. Leonard. Guidance on Markdown: Design philosophies, stability strategies, and select registrations. RFC 7764, IETF, Mar 2016. `www.rfc-editor.org/rfc/rfc7764.txt`

[13] B. Lewis, D. LaLiberte, et al. *GNU Emacs Lisp Reference Manual*. Free Software Foundation, 1997. `gnu.org/s/emacs/manual/html_node/elisp`

[14] T. Mailund. *Introducing Markdown and Pandoc: Using Markup Language and Document Converter*. Apress, 2014. `doi.org/10.1007/978-1-4842-5149-2`

[15] S. Monnier, M. Sperber. Evolution of Emacs Lisp. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–55, 2020. `doi.org/10.1145/3386324`

[16] Org-mode contributors. Org mode tools, 2022. `orgmode.org/worg/org-tools/`

[17] C. Schöch. The right tool for the job: Five collaborative writing tools for academics, 4 April 2014. Impact of Social Sciences Blog. `blogs.lse.ac.uk/impactofsocialsciences/2014/04/04/five-collaborative-writing-tools-for-academics/`

[18] E. Schulte, D. Davison. Active documents with Org-mode. *Computing in Science & Engineering*, 13(3):66–73, 2011. `doi.org/10.1109/MCSE.2011.41`

[19] E. Somma. Il respawn di Infomedia (LaTeX-based). *ArsTEXnica*, (8):92–101, Ottobre 2009. `guitex.org/home/numero-8`

[20] R.M. Stallman. Emacs: The extensible, customizable self-documenting display editor. In *SIGPLAN SIGOA Symposium on Text Manipulation*, 1981. `doi.org/10.1145/872730.806466`

[21] L. Stanisic, A. Legrand, V. Danjean. An effective Git and Org-mode based workflow for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):61–70, 2015.

[22] J. Voegler, J. Bornschein, G. Weber. Markdown – A simple syntax for transcription of accessible study materials. In *Computers Helping People with Special Needs: 14th International Conference, ICCHP 2014, Paris, France, July 9-11, 2014, Proceedings, Part I*, pp. 545–548. Springer, 2014. `doi.org/10.1007/978-3-319-08596-8_85`

[23] J.J. White. Using markup languages for accessible scientific, technical, and scholarly document creation. *Journal of Science Education for Students with Disabilities*, 25(1):22 pp., 2022. Article 5. `scholarworks.rit.edu/jsesd/vol25/iss1/5/`

◇ Emmanuele F. Somma
Bank of Italy
Economics, Statistics and Research
Structural Economic Analysis
`ef dot somma (at) exedre dot org`

[15] The `.org` code of this article is available on the repository `gitlab.com/exedre/arstexnica-orgmode`.