

Building a modern editing environment on Windows around GNU Emacs and AUCTeX

Arash Esbati

Abstract

In this article, we describe how to set up GNU Emacs with AUCTeX as an editing environment for (L^A)TeX on Microsoft Windows using the MSYS2 distribution.

1 Introduction

What we know today as GNU Emacs is a text editor originally developed by Richard Stallman, who is also the founder of the Free Software Foundation (FSF) and the initiator of the GNU Project. The earliest recorded release of GNU Emacs is version 13 from March 1985, though a long history preceded that. Emacs' original inception was as a set of macros and keybindings for the TECO text editor (hence the meaning of "Emacs" as "editor macros"). For a thorough overview of Emacs timeline and technical development, please refer to [4].

TeX had major releases TeX78, TeX82 and TeX3.0 in 1990. Considering that these programs were developed more or less in the same time period and both are free software (free in terms of "free software" and not "open source"[6]), it is not a surprise that Emacs has a long history and very good support for editing TeX files.

Software around TeX and Emacs have their heritage on Unix-like operating systems where the source is provided and the software is built by distros or by the user. On Microsoft Windows, the process of building software by a user is rather uncommon. Finally, porting and building *nix software on Windows is not a task for casual users.

And this is where MSYS2 comes into play. It introduces itself as "a collection of tools and libraries providing an easy-to-use environment for building, installing and running native Windows software. ... Our package repository contains more than 2.600 pre-built packages ready to install." [5]

We will use MSYS2 in order to compile Emacs from the source and install auxiliary packages which will be used by Emacs during editing.

2 Installing the MSYS2 distribution

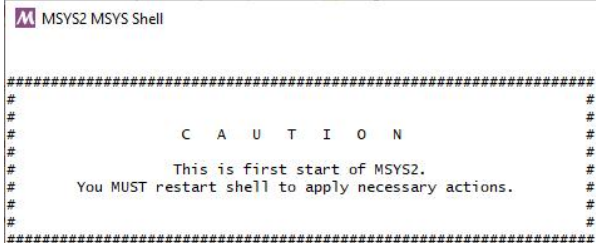
Before we start: As of March 2020, MSYS2 cannot be installed on a 32-bit system. And since January 2023, MSYS2 no longer supports Windows 7 and 8.0. So we need a 64-bit version of Windows 8.1 or higher in order to install the distribution. There are two other points to consider:

1. choosing a installation directory, and
2. choosing a HOME directory.

Regarding item 1, MSYS2 requires extracting its distribution into a folder where the name consists of only ASCII characters and no spaces. It also makes good sense to use a path that is not too long (due to PATH_MAX being 260); `c:\msys64` is ideal.

Regarding item 2, I suggest following the advice "ASCII only, no spaces", and possibly choosing a directory other than `c:\Users\`. I recommend setting the value of the HOME environment variable to the directory chosen above globally on Windows — this is the only variable set outside MSYS2. The HOME directory is the place where Emacs looks for its init file upon start.¹

Now we can fetch MSYS2 from `repo.msys2.org/distrib`. We want to install the portable version, so we download the `msys2-x86_64-latest.tar.xz` archive and unpack it under `c:\`. In the file Explorer, we go to `c:\msys64` and double click on `msys2.exe` which opens a MSYS shell, does the initial setup and ideally shows:



```

MSYS2 MSYS Shell
#####
#                                     #
#                                     #
#          C A U T I O N              #
#                                     #
#          This is first start of MSYS2. #
#          You MUST restart shell to    #
#          apply necessary actions.     #
#                                     #
#                                     #
#####

```

We follow the advice, close the window and double click `msys2.exe` again. To update all packages we run the command `pacman -Syu`. We follow the instructions and close the terminal if requested, then we start a new terminal and update again with `pacman -Su`. That's it!

Working with MSYS2 is easy: If we want to update the distribution or install new packages, we open a MSYS shell (`msys2.exe`) and when we want to use the installed packages, we open a MinGW64 shell (`mingw64.exe`).²

3 Installing Emacs

There are some options available for installing Emacs on Windows. The current stable release is Emacs 29.1.

3.1 Emacs release

The Emacs project provides pre-compiled binaries for Windows on a best-effort basis from `ftpmirror.gnu`.

¹ gnu.org/software/emacs/manual/html_node/efaq-w32/Location-of-init-file.html

² This is the way the author uses MSYS2; changing the shells isn't strictly necessary any more; it is more a habit.

org/emacs in the windows/ subdirectory where each major version of Emacs is kept in its own subdirectory. The compressed files also contain the libraries needed to support various features in Emacs, such as image support.

3.2 MSYS2 release

The MSYS2 project provides also pre-compiled Emacs binaries, usually the latest stable version. It can be installed via pacman with:

```
MSYS shell
$ pacman -S mingw-w64-x86_64-emacs
```

3.3 Building from the source

First, we have to install some tools we need for building Emacs. We want to have a full-fledged Emacs, hence we install a large number of packages. Please refer to [2] for more details. We run `msys2.exe` and enter the following command in the shell (you can paste them into the shell with `Shift+Ins`):

```
MSYS shell
$ pacman -S --needed base-devel \
mingw-w64-x86_64-toolchain \
mingw-w64-x86_64-xpm-nox \
mingw-w64-x86_64-gmp \
mingw-w64-x86_64-giflib \
mingw-w64-x86_64-gnutls \
mingw-w64-x86_64-harfbuzz \
mingw-w64-x86_64-jansson \
mingw-w64-x86_64-lcms2 \
mingw-w64-x86_64-libjpeg-turbo \
mingw-w64-x86_64-libpng \
mingw-w64-x86_64-librsvg \
mingw-w64-x86_64-libtiff \
mingw-w64-x86_64-libwebp \
mingw-w64-x86_64-libxml2 \
mingw-w64-x86_64-sqlite3 \
mingw-w64-x86_64-tree-sitter \
mingw-w64-x86_64-zlib
```

We will build the current development version of Emacs from the Git repository. The code is in sync with what will be Emacs 30. First, install Git:

```
MSYS shell
$ pacman -S git
```

The `autocrlf` feature of Git may interfere with the `configure` file, so we disable it by running:

```
MSYS shell
$ git config --global core.autocrlf false
```

Next we close the current MSYS shell and run `mingw64.exe`. We clone the Emacs repository under a temporary directory:

```
MinGW64 shell
$ mkdir emacs-git
$ cd emacs-git
$ git clone \
```

```
https://git.savannah.gnu.org/git/emacs.git
$ cd emacs
```

The next series of commands builds Emacs.³ With this setup, there is no need to install Emacs; we can invoke it out of the Git tree from the `src` directory. But to do an installation, we create a directory and run `make install` passing that directory to `prefix`:

```
MinGW64 shell
$ mkdir -p /c/msys64/opt/emacs
$ ./autogen.sh
$ ./configure --with-native-compilation \
--without-dbus --without-imagemagick \
--without-mailutils --without-pop
$ make
$ make install prefix=/c/msys64/opt/emacs
```

Note that we can run `make` with the `-j` option:

```
$ make -jN
```

where `N` is the number of CPU-cores in our system; the parallel execution will run significantly faster, speeding up the build process.

3.4 Adjusting the \$PATH

The final step is to add the directory which contains `emacs.exe`, e.g., `c:\msys64\opt\emacs\bin`, to our `$PATH`. We do this in our `~/.bash_profile`:

```
MinGW64 shell
$ cd ~
$ touch .bash_profile
$ echo 'export \
PATH=$PATH:/c/msys64/opt/emacs/bin' \
>>.bash_profile
```

And while we're at it, we do the same for `TeX Live`:

```
MinGW64 shell
$ echo 'export \
PATH=$PATH:/c/texlive/2023/bin/windows' \
>>.bash_profile
```

4 Starting Emacs

The above installs Emacs as a portable application. We will configure other applications, that we'll install later, in our `~/.bash_profile`. So we have to invoke Emacs and other programs from the command-line interface (CLI), `mingw64.exe` in our case, which starts `bash`. We type:

```
MinGW64 shell
$ emacs &
```

This starts Emacs in graphical mode, as shown in figure 1.

³ There is a known issue with GCC 13.1; if the build process breaks, have a look at the file `etc/PROBLEMS` in the Emacs source tree and search for "Building the MS-Windows port with native compilation fails".

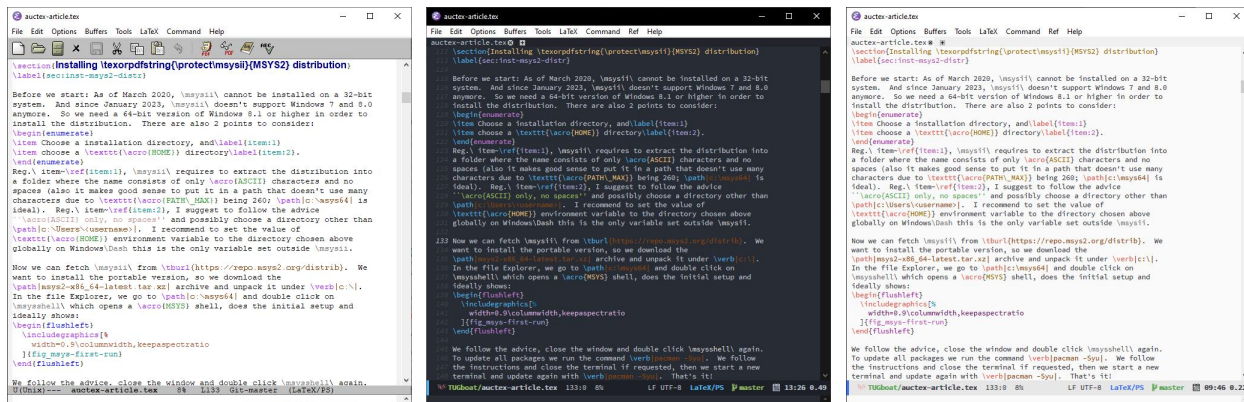


Figure 1: Emacs appearance: Vanilla Emacs (left); with doom-one theme, Windows dark mode and line numbers (middle); and doom-one-light theme (right), the latter two with disabled tool bar

5 Customizing Emacs

Emacs has the reputation for being highly customizable, and some even say Emacs users “customize to live”. For basic usage and customization of Emacs, please refer to the Emacs manual [7], especially chapter 49. A good beginner’s guide is also available.⁴

Next, we briefly mention some initialization code which is useful for installing and using AUCT_EX. We will use the GNU Emacs Lisp Package Archive (ELPA) to install AUCT_EX. The command `list-packages` gives for me a GPG error, but this can be circumvented by adding this to the Emacs init file:

```
Emacs init file
(setq package-check-signature nil)
```

We have to start server communicatoins for backward search in PDF files:

```
Emacs init file
(server-start)
```

We also like to select some text and then start typing where typed text replaces the selection, therefore:

```
Emacs init file
(delete-selection-mode 1)
```

Just in case we want to use a mouse to get a context menu, we add:

```
Emacs init file
(context-menu-mode 1)
```

Finally, if we want to change the font used by Emacs, we use the entry **Options** in the menu bar and go to **Set Default Font**. Figure 1 shows the result of some customization effort: On the left, we see Emacs showing this file without any adjustments, to the middle, a dark theme with Windows dark mode, and to the right, the way the author uses Emacs.

⁴ www.masteringemacs.org/article/beginners-guide-to-emacs

The famous last words before entering the Emacs customizing realms:

- Try to use the Easy Customization Interface.
- Don’t copy every snippet you find on the net into your init file.
- If you do that, read the manual and/or the docstring try to understand what the code does.
- If you don’t understand the change, you probably don’t need it.

6 Choosing a T_EX mode for Emacs

After installing Emacs, it’s time to choose the appropriate support for authoring (L^A)T_EX files. Emacs has two major modes for this purpose: A built-in mode and the one provided by the AUCT_EX package.⁵

So, which to choose? A general guideline might be: If you rely only on vanilla L^AT_EX commands and environments, then try the built-in variant. If you will use large number of packages, want completion for the macros or environments and their (key-value) arguments, including syntax highlighting, and might define your own macros and environments and completion support for them is desired, then go for AUCT_EX.

7 Installing AUCT_EX

The modern and strongly recommended way of installing AUCT_EX is by using the package manager integrated in Emacs to fetch it from ELPA. We type `M-x list-packages RET /n auctex RET`, put the cursor on `auctex`, press `i` and we see this:

⁵ Each mode provides dedicated support for plain T_EX, L^AT_EX, DocT_EX (for `.dtx` files) and SliT_EX, but we will focus on L^AT_EX.

Package[name:auctex]	Version	Status ▼	Archiv
I auctex	13.1.9	available	gnu
auctex-cluttex	20220730.1100	available	melpa
auctex-latexmk	20221025.1219	available	melpa
auctex-lua	20151121.1610	available	melpa
auto-complete-auctex	20140223.1758	available	melpa
company-auctex	20200529.1835	available	melpa

Now we hit `x` to execute the installation procedure. That’s all. Using the ELPA version has several advantages. Besides being platform and OS independent, we will receive intermediate bugfix releases between major AUCTEX releases.

A word of caution: The way we installed AUCTEX, we must not have a line like this in our init file:

```
(load "auctex.el" nil t t)
```

or even worse:

```
(require 'tex-site)
```

Having either such line in our init file may be harmful for the correct operations of AUCTEX.

8 Configuring AUCTEX

AUCTEX comes with a huge number of customization options; the figure below shows the various groups of options, some with subgroup(s).

AUCTEX group: A (La)TeX environment.
 State: visible group members are all at standard
 See also [Home Page](#) and [Manual](#).

► **Tex Modes**
 List of modes provided by AUCTEX. [More](#)

Subgroups:

- [LaTeX](#): LaTeX support in AUCTEX.
- [Tex Command](#): Calling external commands from AUCTEX.
- [Tex File](#): Files used by AUCTEX.
- [Tex Indentation](#): Indentation of TeX buffers in AUCTEX.
- [Tex Macro](#): Support for TeX macros in AUCTEX.
- [Tex Misc](#): Various AUCTEX settings.
- [Tex Output](#): Parsing TeX output.
- [Tex Parse](#): Parsing TeX files from AUCTEX.
- [Tex Quote](#): Quoting in AUCTEX.
- [Font Latex](#): Font-latex text highlighting package.

They are well described in the AUCTEX manual [10]. We will discuss some important options below which should be set before starting work.

Documents we edit can be a single file, or spread over many files consisting of a “master” file in which we include other files via L^AT_EX macros like `\input` and `\include`. AUCTEX can deal with both single and multi-file projects and knows which file to compile via the variable `TeX-master`. This variable should be set in the Emacs init file and will also be inserted in each file’s local variables. In general, it is a good idea to do:

```
Emacs init file
(setq-default TeX-master nil)
```

which means that when we create a new T_EX file, AUCTEX will ask for the name of the “master” file associated with the buffer and insert a marker as a

file variable in that file. For a single file project, it will look like this:

```
_____ .tex file _____
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

For a multi-file project, it might look like this:

```
_____ .tex file _____
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "../phd-main"
%%% End:
```

Another important variable is `TeX-parse-self`. AUCTEX depends heavily on being able to extract information from the buffers by parsing them. Since parsing the buffer can be somewhat slow, the parsing is initially disabled. We enable it by adding the following line to our init file:

```
_____ Emacs init file _____
(setq TeX-parse-self t)
```

This change means: Upon loading a `<filename>.tex`, AUCTEX will look in an `auto` subdirectory for parsed information stored in `<filename>.el`. If it finds that file, it is loaded and the information from it is applied to the current editing buffer. If there is no such file, AUCTEX parses the current buffer and applies that information to the buffer. The information applied consists of names of used packages, where AUCTEX loads its corresponding support files, user-defined macros and environments, defined labels for completion, etc. There is a catch here: AUCTEX doesn’t distinguish among extensions of parsed files. So if we have a T_EX file named, say, `geometry.tex`:

```
_____ Example for geometry.tex _____
\documentclass{article}
\usepackage{xcolor}
\begin{document}
text
\end{document}
```

AUCTEX will save the information after parsing in `geometry.el`; upon the next loading of the saved `geometry.el`, it loads `article.el`, `xcolor.el` and the file `geometry.el` provided by AUCTEX itself which adds support for macros provided by `geometry.sty`—but we did not load that package. In general, we should always use distinct names for our T_EX files in order to avoid this sort of clash.

A related option is `TeX-auto-save`. When set to non-`nil`, AUCTEX will parse the file and write the information each time the T_EX file is saved. Again, this option is initially disabled. We can still force the parsing of the T_EX file by pressing `C-c C-n` for `TeX-normal-mode`. This is often the best choice, as

we will be able to decide when it is necessary to reparse the file.

If we use packages which define table environments and we want to put captions above the tables, we adjust the variable `LaTeX-top-caption-list`:

```

----- Emacs init file -----
(setq LaTeX-top-caption-list
  ('("table"          "table*"
     "Sctable"        "Sctable*"
     "sidewaystable" "sidewaystable*")))

```

Finally, we tell `AUCTeX` to convert all tabs in multiple spaces, preserving the indentation, when we save a file:

```

----- Emacs init file -----
(setq TeX-auto-untabify t)

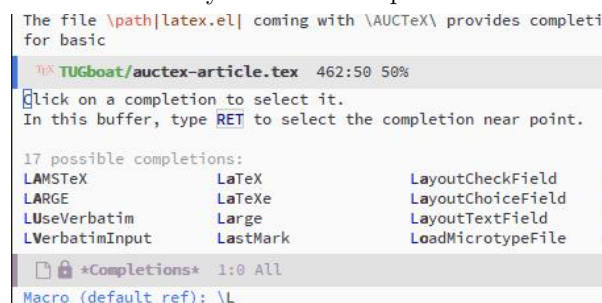
```

9 Using AUCTeX

`AUCTeX` has an extensive manual which describes its usage in great detail [10]. Hence, we will discuss only some general usage aspects, focusing on completion of macros and environments with their arguments.

The file `latex.el` that comes with `AUCTeX` provides completion support for basic \LaTeX macros and environments. As package files extend \LaTeX 's functionality, `AUCTeX`'s style files extend its completion support. These style files are named after the package or class names used in a \TeX file or the \TeX file which was parsed, so (as mentioned above) `geometry.el` contains completion support for the macros provided by `geometry.sty`.

Completion support in `AUCTeX` is built around Emacs' *minibuffer completion*.⁶ The entry points for inserting with completion are the functions `TeX-insert-macro` (bound to `C-c C-m` or `C-c RET`) and `LaTeX-insert-environment` (bound to `C-c C-e`). For example, this is what we see after hitting `C-c C-m L` followed by a `TAB` for completion candidates:



```

The file \path|latex.el| coming with \AUCTeX\ provides completi
for basic

TUGboat/auctex-article.tex 462:50 50%
Click on a completion to select it.
In this buffer, type RET to select the completion near point.

17 possible completions:
LAMSTeX      LaTeX      LayoutCheckField
LARGE       LaTeXe    LayoutChoiceField
LUseVerbatim Large     LayoutTextField
LVerbatimInput LastMark  LoadMicrotypeFile

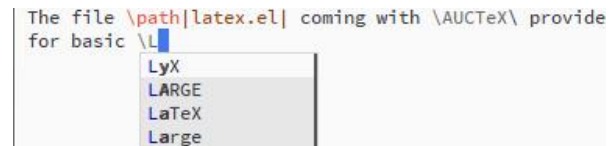
*Completions* 1:0 All
Macro (default ref): \L

```

`AUCTeX` presents the known candidates and we can narrow down the choices by typing further and hitting `RET` once we have the right macro which is inserted into the buffer and further arguments are queried, if applicable.

⁶ gnu.org/software/emacs/manual/html_node/emacs/Completion.html

But sometimes we just want to insert the macro directly into the buffer, or find out we have forgotten a key-value pair in an argument where hitting the keystrokes described above will not help: we want *in-buffer completion*. As in the scenario above where we wanted to insert the `\LaTeX` macro, we can insert `\L` in the buffer followed by `TAB` and we get:



```

The file \path|latex.el| coming with \AUCTeX\ provide
for basic \L
LyX
LARGE
LaTeX
Large

```

where we can choose the macro and hit `RET` to insert.

`AUCTeX` also checks if we are in math mode and offers math symbols for completion. In order to get in-buffer completion, we need to install a package like `corfu`⁷ or `company`⁸ and configure it accordingly. It should be noted that in-buffer completion is not implemented in `AUCTeX` for all macro and environment arguments; this is work in progress.

10 Hacking AUCTeX

One of `AUCTeX`'s chief achievements is that its parser is “hackable”, i.e., `AUCTeX` users and style files can extend the built-in parser with Lisp code. For example, this document uses the `fvextra` package, which loads `fancyvrb` in turn, and defines a custom verbatim environment, named `codesnippet`, like this:

```

----- Custom environment -----
\DefineVerbatimEnvironment{codesnippet}
{Verbatim}{%
  fontsize = \small ,
  frame    = topline ,
  breaklines ,
  framesep = 4pt
}

```

`AUCTeX` has a style file `fvextra.el`, which loads the style `fancyvrb.el` in turn, which contains code telling `AUCTeX` about the macro and its arguments defining a new verbatim environment. With the \TeX code above in a file, `AUCTeX` sets its internal variables properly itself upon next parsing and no user intervention is needed. The new environment `codesnippet` is available when `C-c C-e` is hit, including completion and query for the optional key-value argument. Syntax highlighting support is also set automatically:

⁷ github.com/minad/corfu

⁸ company-mode.github.io

```

\begin{codesnippet}[label={Custom environment}]
\DefineVerbatimEnvironment{codesnippet}
{Verbatim}{%
  fontsize = \small ,
  frame    = topline ,
  breaklines
  framesep = 4pt
}
\end{codesnippet}

```

The general strategy for extending the parser is to write an AUCTEX style file where we:

- initialize the new entry to the parser by calling the `TeX-auto-add-type` lisp macro with its arguments;
- write a variable containing the regular expression which should be added to the parser and plug it into AUCTEX inside the hook;
- write a function which is run before parsing, resetting the results from the last parser run;
- write a function which is run after parsing, processing the results from the actual parser run.

We will discuss this process with two examples.

10.1 A simple example

The `geometry` package provides a facility to save the page dimensions as a *<name>* and load these dimensions later in the document. The macros are `\savegeometry` for saving the page dimensions, and `\loadgeometry` for loading. The AUCTEX style file `geometry.el` has the following code to parse the newly defined *<name>*. First, a new entry for the parser is setup with:

```

_____ geometry.el _____
(TeX-auto-add-type "geometry-savegeometry"
  "LaTeX"
  "geometry-savegeometries")

```

`TeX-auto-add-type` is a Lisp macro which takes two mandatory and one optional arguments: The first argument is a *<name>*, which is prefixed by the second argument *<prefix>*. Usually, *<name>* is composed as *<package-macro>* and *<prefix>* is the name of the engine or format used, in this case LaTeX. The third argument is the plural form of the first argument; by default just an `s` is added. The Lisp macro defines: the variable `LaTeX-auto-geometry-savegeometry` which holds the bare results after a successful parsing run; the function `LaTeX-geometry-savegeometry-list` which sorts and eliminates any dupes from `LaTeX-auto-geometry-savegeometry`; the variable `LaTeX-geometry-savegeometry-list` which holds the information returned by the function of the same name; and the function `LaTeX-add-geometry-savegeometries` which can be used to add new elements to `LaTeX-geometry-savegeometry-list`.

Next, `geometry.el` defines the variable `LaTeX-geometry-savegeometry-regexp`:

```

_____ geometry.el _____
(defvar LaTeX-geometry-savegeometry-regexp
  '("\\\\savegeometry{\\([~]+\\}")
  1 LaTeX-auto-geometry-savegeometry))

```

which is a list of three elements: A string with the regular expression to match against, including a grouping construct for future reference, in this case the argument of `\savegeometry` with `{\\([~]+\\)}`. The second element is an integer or a list of integers containing the number(s) of substring(s) matched, and finally the name of the variable to put the parsed substring(s) in. After this, a function is defined in preparation for parsing and is added to `TeX-auto-prepare-hook`:

```

_____ geometry.el _____
(defun LaTeX-geometry-auto-prepare ()
  (setq LaTeX-auto-geometry-savegeometry nil))

(add-hook 'TeX-auto-prepare-hook
  #'LaTeX-geometry-auto-prepare t)

```

And finally, the defined regular expression is added to the parser with the function `TeX-auto-add-regexp` inside the hook. Also, two entries are defined for the L^AT_EX macros:

```

_____ geometry.el _____
(TeX-add-style-hook
  "geometry"
  (lambda ()
    (TeX-auto-add-regexp
     LaTeX-geometry-savegeometry-regexp)
    (TeX-add-symbols
     `("savegeometry"
       ,(lambda (optional)
          (let ((name (TeX-read-string
                       (TeX-argument-prompt
                        optional nil "Name"))))
              (LaTeX-add-geometry-savegeometries
               name)
              (TeX-argument-insert name
                                   optional))))))
    '("loadgeometry"
      (TeX-arg-completing-read
       (LaTeX-geometry-savegeometry-list)
       "Name")))))

```

The entry for "savegeometry" queries for a name and adds the user input to list of new names. The entry for "loadgeometry" retrieves all defined names and offers them as argument with completion.

10.2 A more complex example

For a more complex example, we look at the AUCTEX style file `enumitem.el` which contains code to parse new environments defined with the `\newlist` macro:

```

enumitem.el
(TeX-auto-add-type "enumitem-newlist" "LaTeX")

(defvar LaTeX-enumitem-newlist-regex
  ("\\newlist{\\([~]+\\)}{\\([~]+\\)}"
   (1 2) LaTeX-auto-enumitem-newlist))

```

`\newlist` takes three arguments, but only the first two, a *name* and *type*, are relevant. So the regular expression matches two arguments and both are added to the variable containing the results. Next, two functions are defined to prepare the parsing and process the results:

```

enumitem.el
(defun LaTeX-enumitem-auto-prepare ()
  (setq LaTeX-auto-enumitem-newlist nil))

(defun LaTeX-enumitem-auto-cleanup ()
  ;; \newlist{<name>}{<type>}{<depth>}
  ;; env=<name>, type=<type>
  (dolist (env-type
           (LaTeX-enumitem-newlist-list))
    (let* ((env (car env-type))
           (type (cadr env-type)))
      (LaTeX-add-environments
       ` (,env
         LaTeX-env-item-args
         [TeX-arg-key-val
          (LaTeX-enumitem-key-val-options)]))
      (when (member type '("description"
                          "description*"))
        (add-to-list
         'LaTeX-item-list
         ` (,env . LaTeX-item-argument)))
      (TeX-ispell-skip-setcdr
       ` ((,env ispell-tex-arg-end 0))))))

```

The second function is the interesting one: Every user-defined environment is added to the list of known environments, including support for key-value query for the optional argument. For description-like environments, the optional argument of `\item` will be queried as well. And finally, the optional argument of the environment is ignored during spell-checking (see §14). These functions and the regular expression are added to AUCTeX with:

```

enumitem.el
(add-hook 'TeX-auto-prepare-hook
  #'LaTeX-enumitem-auto-prepare t)
(add-hook 'TeX-auto-cleanup-hook
  #'LaTeX-enumitem-auto-cleanup t)

(TeX-add-style-hook
 "enumitem"
 (lambda ()
  (TeX-auto-add-regex
   LaTeX-enumitem-newlist-regex)))

```

The techniques described above can also be used for any user-defined macros which define new macros and/or environments. The best approach is to put the L^AT_EX macros inside a package and the corresponding Lisp code inside an AUCTeX style file saved in a directory which is part of `TeX-style-private`. This way, the Lisp code is loaded each time the custom package is requested with `\usepackage`.

11 Using preview-latex

`preview-latex` is a package embedding preview fragments into Emacs source buffers under the AUCTeX editing environment for L^AT_EX. It uses `preview.sty` for the extraction of certain environments (most notably displayed formulas). `preview-latex` was originally written by David Kastrup and is now maintained by the AUCTeX team. It has an extensive manual describing the relevant aspects of usage and configuration [3].

12 Using RefTeX

RefTeX is a package for managing labels, references, citations and index entries for L^AT_EX documents within Emacs. RefTeX has been bundled and pre-installed with Emacs since version 20.2. Originally written by Carsten Dominik, it is currently maintained by the AUCTeX team. RefTeX has an excellent manual describing its functionality and options [1].

RefTeX can be used with both the built-in L^AT_EX mode and AUCTeX. In order to plug RefTeX into AUCTeX, these two lines in our init file suffice:

```

Emacs init file
(add-hook 'LaTeX-mode-hook #'turn-on-reftex)
(setq reftex-plugin-into-AUCTeX t)

```

The first line activates RefTeX automatically when AUCTeX is loaded and the second line turns on all RefTeX features within AUCTeX. The integration of the packages is seamless: AUCTeX checks in its style files if RefTeX is activated and updates RefTeX's variables with parsed elements where appropriate, and RefTeX's advanced mechanism for inserting labels and referencing them is used when AUCTeX's functions are invoked.

For example, within this document, a new environment `codesnippet` is defined (see §10). The `fancyvrb` package provides a key `reflabel` to define a new label to be used by `\pageref`. Now when we hit C-c C-e code<TAB> RET ref<TAB> RET without = and a value, AUCTeX completes the key and also adds `={lst:1}` to the key where the value is generated by RefTeX. We can now reference this label by hitting C-c C-m RET `pageref` RET and now AUCTeX delegates the request for labels to RefTeX and

we choose the label type in the minibuffer with 1 and see the following:

```
12 Using \texorpdfstring{\protect\RefTeX}{RefTeX}
> lst:1
(add-hook 'LaTeX-mode-hook #'turn-on
```

Similar things happen with citation macros.

Since the L^AT_EX release of October 2019, it is possible to use non-ASCII characters in labels such as `\label{eq:größer}`. With the standard setup, RefT_EX will not allow us to enter such a label and complain about invalid characters. This behavior can be changed with the following addition to our Emacs init file:

```
Emacs init file
(setq refTeX-label-illegal-re
 "[^-[[:alnum:]]_+=:;,.]")
```

13 Using a PDF viewer

On Windows, there are two T_EX friendly PDF viewers: SumatraPDF⁹ and Sioyek.¹⁰ Both keep the PDF file unlocked, and both support SyncT_EX. SumatraPDF has been around since 2006, Sioyek since 2021. We will use SumatraPDF. Installing SumatraPDF is easy: We fetch the portable version and unpack the single binary into `c:\msys64\usr\local\bin`. We run `mingw64.exe` and rename the file:

```
MinGW64 shell
$ cd /usr/local/bin
$ mv SumatraPDF-3.4.6-64.exe SumatraPDF.exe
```

Now we have to tell both parties, Emacs and SumatraPDF, about their counterparts. AUCT_EX has built-in support for SumatraPDF, so there is not much to do but put this in our init file:

```
Emacs init file
(setq TeX-view-program-selection
 '((output-pdf "SumatraPDF")))
```

Emacs will find `SumatraPDF.exe` since it's installed in the MSYS2 file tree.

Next, under SumatraPDF options for inverse search command-line, we enter the following (except all on one line):

```
SumatraPDF options
c:\msys64\opt\emacs\bin\emacsclientw.exe -n
--alternate-editor=
c:\msys64\opt\emacs\bin\runemacs.exe
+%l "%f"
```

which means: Use the program `emacsclientw.exe` to connect to Emacs server, and if there is no Emacs server running, invoke `runemacs.exe` to open Emacs and connect to it. Note that this only works when SumatraPDF is invoked from a MinGW64 shell with:

⁹ sumatrapdfreader.org
¹⁰ sioyek.info

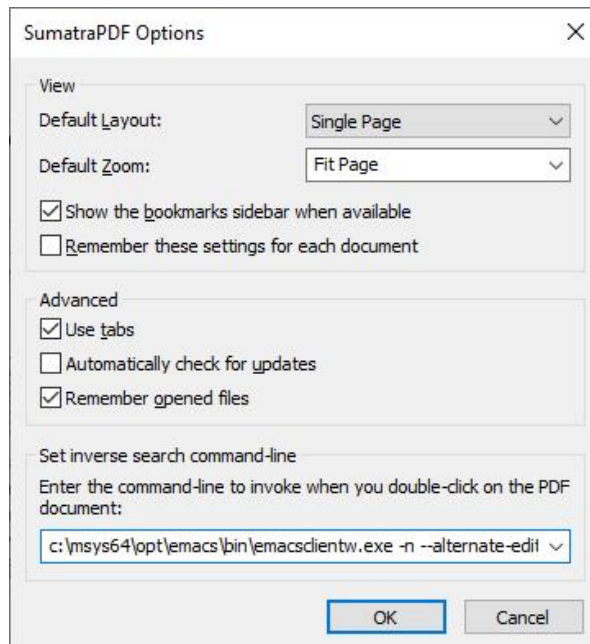


Figure 2: AUCT_EX options for SumatraPDF, including inverse search.

```
MinGW64 shell
$ SumatraPDF.exe &
```

Or when invoked with `C-c C-v` from Emacs, everything works just fine.

Finally, we tell AUCT_EX during editing to enable SyncT_EX (“inverse search”) when running the compiler; see figure 2. If we want to enable SyncT_EX ad-hoc for a file, we can hit `C-c C-t C-s` which activates `TeX-source-correlate-mode` for the current file. If we want to have this mode activated for a specific file, we can add the following to the file:

```
.tex file
%%% Local Variables:
%%% mode: latex
%%% TeX-source-correlate-mode: t
%%% End:
```

And if we want to have the mode always enabled, we can customize the variable `TeX-source-correlate-mode` to `t`.

14 Using a spelling checker program

Emacs supports the external spell checkers Hunspell, Aspell, Ispell and Enchant. These programs are not part of Emacs and must be installed separately. We’ll use Hunspell because it has the feature that we can use multiple language dictionaries at once. The complete setup consists of three parts:

- install the program itself;
- install the language dictionaries;
- set up Emacs to use the above.

Installing the program is easy: We run `msys2.exe` and enter:

```
MSYS shell
$ pacman -S mingw-w64-x86_64-hunspell
```

Next we need to create the directory where we will install the dictionaries, say under `/usr/local/share/hunspell`. We enter this in the shell and exit:

```
MSYS shell
mkdir -p /c/msys64/usr/local/share/hunspell
```

In our `~/ .bashrc`, we add the following lines:

```
~/ .bashrc
DICPATH=/c/msys64/usr/local/share/hunspell
WORDLIST=$HOME/.emacs.d/hunspell_default
export DICPATH WORDLIST
```

Next we download dictionaries for US English¹¹ and other languages.¹² We rename the `.oxt` extension to `.zip` so we can open the archive easily and we move the files with `.aff` and `.dic` extension into the `DICPATH` directory chosen above. Now we run `mingw64.exe` and enter:

```
MinGW64 shell
$ hunspell -D
```

Hunspell should report the available dictionaries in the `msys64` file tree.

Now we tell Emacs about Hunspell and add the following line to our init file:

```
Emacs init file
(setopt ispell-program-name "hunspell")
```

The next line tells Emacs about the default dictionary to use. E.g., for people preferring to write in German, it would be:

```
Emacs init file
(setq ispell-dictionary "deutsch8")
```

When we're writing L^AT_EX, we have to pass the `-t` option to Hunspell:

```
Emacs init file
(add-hook 'LaTeX-mode-hook
  (lambda ()
    (setq-local ispell-extra-args
      ("-t"))))
```

We also set the name of our personal dictionary:

```
Emacs init file
(setq ispell-personal-dictionary
  (expand-file-name
    "~/ .emacs.d/hunspell_default"))
```

This file must exist for Hunspell, but it can be an empty file. Finally, we define some key bindings to switch dictionaries:

¹¹ downloads.sourceforge.net/wordlist/hunspell-en-US-2020.12.07.zip

¹² extensions.libreoffice.org

```
Emacs init file
(keymap-global-set
 "C-c i e"
 (lambda ()
  (interactive)
  (ispell-change-dictionary "english")))
```

```
(keymap-global-set
 "C-c i d"
 (lambda ()
  (interactive)
  (ispell-change-dictionary "deutsch8")))
```

```
(keymap-global-set
 "C-c i a"
 (lambda ()
  (interactive)
  (require 'ispell)
  (ispell-set-spellchecker-params)
  (ispell-hunspell-add-multi-dic
   "de_DE,en_US")
  (ispell-change-dictionary
   "de_DE,en_US")))
```

Now we can invoke Hunspell inside Emacs with `M-x ispell` or inside AUCT_EX with `C-c C-c Spell`. More information can be obtained from the Emacs manual.¹³ AUCT_EX provides a library `tex-ispell.el` which contains extensions for skipping certain macros, arguments and environments when spell checking. The supported packages are listed in the header of the library. These extensions are activated by default; they can be disabled by setting the value of `TeX-ispell-extend-skip-list` to `nil`.

15 Using Pygments

If we want to use the `minted` package, we have to install the additional software `Pygments`. We run `msys2.exe` and enter:

```
MSYS shell
$ pacman -S mingw-w64-x86_64-python-pygments
```

We can check the installation by running `mingw64.exe` and:

```
MinGW64 shell
$ which pygmentize.exe
```

which returns `/mingw64/bin/pygmentize.exe`.

`minted` requires that we pass the `-shell-escape` option to the L^AT_EX processor. This can be done by setting the AUCT_EX variable `TeX-command-extra-options` as a file local variable:

```
.tex file
%% Local Variables:
%% mode: latex
%% TeX-command-extra-options: "-shell-escape"
%% End:
```

¹³ gnu.org/software/emacs/manual/html_node/emacs/Spelling.html

AUCTeX has extensive support for the minted package, so using the package should work flawlessly.

16 Using a linter

There are two linters available for L^AT_EX documents: lacheck¹⁴ and ChkTeX.¹⁵ Both of them are available with T_EX Live as part of `collection-binextra`.

Both programs are supported by AUCTeX, so the question is how to invoke them. This is mostly a matter of preference: Some people like running the linter now and then and see the results, and some want to have it running all the time during typing. For the former case, one can hit `C-c C-c Check RET` for lacheck or `C-c C-c ChkTeX RET` for ChkTeX. Then a buffer is created with the result:

```
Both programs are supported by \AUCTeX, so the question is how to invoke
them. This is mostly a matter of preference: Some people like running the
linter now and then and see the results, and some want to have it running
all the time during typing. For the former case, one can hit
\keystroke|C-c C-c Check RET| for lacheck or \keystroke|C-c C-c ChkTeX RET|
for ChkTeX. Then a buffer is created with the result:

TUGboat/auctex-article.tex 757:55 91%

*compilation* * *
"auctex-article.tex", line 756.65:(#20) User-specified pattern found: ChkTeX.
"auctex-article.tex", line 775.62:(#25) You might wish to put this between
"auctex-article.tex", line 775.69:(#25) You might wish to put this between
"auctex-article.tex", line 782.62:(#25) You might wish to put this between
"auctex-article.tex", line 782.69:(#25) You might wish to put this between
"auctex-article.tex", line 821.1:(#16) Mathmode still on at end of LaTeX fi
ChkTeX v1.7.8 - Copyright 1995-96 Jens T. Berger Thielemann.
```

For the latter case of on-the-fly syntax checking, Emacs provides a minor mode called Flymake which is supported by AUCTeX. It can be activated with `M-x flymake-mode`. The same result now looks like this:

```
Both programs are supported by \AUCTeX, so the question is how to invoke
them. This is mostly a matter of preference: Some people like running the
linter now and then and see the results, and some want to have it running
all the time during typing. For the former case, one can hit
\keystroke|C-c C-c Check RET| for lacheck or \keystroke|C-c C-c ChkTeX RET|
for ChkTeX. Then a buffer is created with the result:

TUGboat/auctex-article.tex 757:55 91% LF

*Flymake diagnostics for 'auctex-article.tex'* * *
Line Col Type Backend Message
756 64 warning L-f User-specified pattern found: ChkTeX.
774 61 warning L-f You might wish to put this between a pair of `{
774 68 warning L-f You might wish to put this between a pair of `{
781 61 warning L-f You might wish to put this between a pair of `{
781 68 warning L-f You might wish to put this between a pair of `{
820 0 warning L-f Mathmode still on at end of LaTeX file.
```

Note also the visual effects we get with Flymake. Flymake has also an extensive manual [8].

17 Using a LSP server

Emacs 29 ships with a new library called `eglot.el` (for *Emacs Polyglot*) which is a built-in client for the *Language Server Protocol* (LSP). LSP is a standardized communications protocol between source code editors and language servers—programs external to Emacs which analyze the source code on behalf of Emacs. We can now open a source file and type `M-x eglot`, presuming that an appropriate language server is installed. Eglot comes with a manual describing the details [9].

¹⁴ ctan.org/pkg/lacheck

¹⁵ ctan.org/pkg/chktxe

Currently, two LSP servers are available for L^AT_EX: TexLab¹⁶ and Digestif.¹⁷ Installing TexLab is easy: We download the correct version from project's page and unpack `texlab.exe` into `c:\msys64\usr\local\bin`. Digestif is part of T_EX Live and distributed as `digestif.exe`.

`eglot` knows about both TexLab and Digestif, so we can activate a LSP server by hitting `M-x eglot` and choosing the one we want in case both servers are installed. That's it. The next figure shows an example for this document with TexLab which adds the section number to the `\label` macro and provides annotated completion for the `\ref` macro.

```
\section{Using a \texorpdfstring{\acro}{}{LSP} server}
!\label{sec:using-lsp-server} Section 17

!!\ref{sec:c}
[ ] sec:choosing-tex-mode Section 6 (Choosing a \texorpdf
[ ] sec:conclusion
[ ] sec:configuring-auctex Section 8 (Configuring \texorpdf
[ ] sec:custom-emacs
[ ] sec:adjusting-path Subsection 3.4
[ ] sec:bibtex-database Section 18 (Editing Bib\textor
[ ] sec:building-from-source Subsection
```

18 Editing BIB_TE_X databases

Emacs has a built-in major mode for editing BIB_TE_X files which is used when we open a `.bib` file. This major mode supports both BIB_TE_X and BIB_LA_TE_X; BIB_TE_X is the default. This can be changed by customizing the variable `bibtex-dialect`:

```
Emacs init file
(setopt bibtex-dialect 'biblatex)
```

Once the mode is active, it is easy to use the menus or the context menu to add new entries and operate on the fields.

19 Miscellaneous settings

This section describes various other settings which should make the daily work easier.

AUCTeX provides in-buffer completion which can be activated with the `TAB` key. The `TAB` key is somewhat overloaded since it is also used for indentation. The operation of `TAB` can be controlled with the variable `tab-always-indent`. We can set this in our init file:

```
Emacs init file
(setq tab-always-indent 'complete)
```

which means: `TAB` first tries to indent the current line, and if the line was already indented, then try to complete the thing at point.

T_EX Live provides a batch script `tlmgr.bat` for managing the distribution. Being a batch file, it is not possible to run the script inside a MinGW64 shell. We can change this by putting this small

¹⁶ github.com/latex-lsp/texlab

¹⁷ github.com/astoff/digestif

snippet under `c:\msys64\usr\local\bin` and name it `tlmgr`:

```

_____          tlmgr script          _____
#!/bin/sh
# This is a small wrapper around tlmgr.bat
# Note the double // for escaping /
cmd.exe //c tlmgr.bat "$@"; exit $?

```

When we're inside the MinGW64 shell, hitting TAB provides completion for executables and/or file names. Under Windows, also files with `.dll` suffix are offered for executable completion. We change this with this line in our `~/bashrc`:

```

_____          ~/bashrc          _____
export EXECIGNORE=*.dll

```

EXECIGNORE is a colon-separated list of glob patterns to ignore when completing on executables. This is an MSYS2¹⁸ feature.

Another handy idea is to alias `emacsclient` to run `emacsclient.exe` with some options:

```

_____          ~/bashrc          _____
alias emacsclient='emacsclient -n \
--alternate-editor=runemacs'

```

20 Conclusion

TeX has been around for some time now, and so has Emacs. Both carry the original ideas of their developers, but they have also managed to evolve over the decades. Emacs can be set up to look modern,¹⁹ but more importantly, it also supports modern techniques to support users to write L^AT_EX documents.

With the advent of MSYS2, it is easily possible to build Emacs from the source on Windows, so an initial barrier to getting the program is gone. With AUCT_EX, a configurable major mode for L^AT_EX is available which can be installed easily as a package from ELPA. Ref_TE_X is a great tool for managing labels and citations and is bundled with Emacs. Other tools around the editor such as spell-checker, PDF viewer, Pygments, linter, etc., can be integrated into the editing environment without trouble.

One new feature in Emacs 29 is the built-in client for LSP servers which works out of the box for available language servers. The support for this feature is expected to grow. Another new feature in Emacs 29 is the built-in support for the incremental parsing library Tree-sitter. The usage of Tree-sitter with Emacs for TeX editing is an area which needs more exploration in the future.

¹⁸ Cygwin, to be more precise.

¹⁹ Depending on the definition, which currently seems to be Microsoft Visual Studio Code.

Overall, Emacs provides a very good environment for editing (L^A)T_EX documents using up-to-date tools and techniques which can be easily set up on Windows.

Acknowledgments

I'm grateful to the AUCT_EX development team and Óscar Fuentes (MSYS2 contributor) for their comments on this article.

References

- [1] C. Dominik. *Ref_TE_X — Support for L^AT_EX labels, references, citations and index entries with GNU Emacs*. gnu.org/software/auctex/manual/reftex.index.html
- [2] Emacs. Building and Installing Emacs on 64-bit MS-Windows using MSYS and MinGW-w64. git.savannah.gnu.org/cgit/emacs.git/tree/nt/INSTALL.W64
- [3] D. Kastrup, J.Å. Larsson, et al. *preview-latex — A L^AT_EX preview mode for AUCT_EX in Emacs*. gnu.org/software/auctex/manual/preview-latex.index.html
- [4] S. Monnier, M. Sperber. Evolution of Emacs Lisp. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020. doi.org/10.1145/3386324
- [5] MSYS. MSYS Software Distribution and Building Platform for Windows. msys2.org
- [6] R. Stallman. Why Open Source Misses the Point of Free Software. gnu.org/philosophy/open-source-misses-the-point.en.html, 2007–2021.
- [7] R. Stallman, et al. *GNU Emacs Manual (updated for Emacs version 29.1)*, 1985–2023. gnu.org/software/emacs/manual/emacs
- [8] J. Távora, P. Kobiakov. *GNU Flymake*. gnu.org/software/emacs/manual/flymake.html
- [9] J. Távora, E. Zaretskii. *Eglot: The Emacs Client for the Language Server Protocol*. joatavora.github.io/eglot/
- [10] K.K. Thorup, P. Abrahamsen, et al. *AUCT_EX: A sophisticated TeX environment for Emacs*. gnu.org/software/auctex/manual/auctex.index.html

◇ Arash Esbati
Germany
[arash \(at\) gnu dot org](mailto:arash@gnu.org)