editors

Using Emacs and AucTEX for preparing LATEX documents

Piet van Oostrum

abstract

Users of TEX and LATEX can be helped very much by an editor that knows about the specifics of these packages. For instance it can do syntax coloring so that the user sees immediately the difference between normal text and TEX commands; it can insert skeletons for often used commands and environments in order to prevent missing elements (e.g. missing `\end` parts), etc.

This article describes the use of the Emacs editing environment with the AucTEX package for the preparation of LATEX documents. The main characteristics of Emacs are discussed, followed by a more detailed description of the facilities that AucTEX offers to assist the author of LATEX documents. Finally we describe how AucTEX can be customized to support your own or external commands and LATEX packages.

keywords

Emacs, AucTEX, TEX, LATEX, packages, editing

What is Emacs?

Emacs (originally the name stood for "*Editing macros*") is basically an editor. But calling Emacs just an editor isn't fair: Emacs is much more. It can be used for reading and writing email and Usenet news, and as a Java development environment, to mention just a few applications. It would be better to call it as least an editing environment, or maybe a text work environment.

There exist several versions of Emacs, the most widely used one of which is GNU Emacs which was created by Richard Stallman. Even this one exists in two separate development streams, one maintained by Richard Stallman and the FSF (Free Software Foundation) and the other one (XEmacs) which started as an offspring of GNU Emacs at the now defunct Lucid Corporation. The main difference between these versions is the graphical user interface and the graphical possibilities which in XEmacs is more advanced. Most Emacs packages run on both versions, however. In the rest of the article we will use the generic term "Emacs" for both versions.

There are also several Emacs clones, which were primarily targeted at personal computers, because GNU Emacs and XEmacs are rather large. These clones are, however, not compatible with the main versions of Emacs, although some of them (e.g. Jed) have quite advanced possibilities, also for TEX and LATEX editing. We will not deal with these in this article, however. Both GNU Emacs and XEmacs have now complete versions running on The Win32 platform, and are therefore also available for the majority of PC users.

Emacs is a very flexible editing environment, which can be tailored to a lot of tasks. The main flexibility stems from the fact that Emacs is a *programmable* editor. It has a built-in Lisp interpreter with special primitives for text handling, buffer management, screen display, network functions (TCP/IP), operating system interface, and similar things. The basic operations and those where speed is important have been written in C, but all other functions are in Lisp. This makes it very easy to change the behaviour of Emacs, as the Lisp functions can be redefined at runtime, new functions can be added, and most of this happens automatically. Therefore Emacs is dynamically customizable and extendible.

Emacs also has an online documentation system, both for itself and for external programs. As an example, the documentation of LATEX commands is available and can be displayed while editing LATEX documents, even with defaulting automatically to the command or environment where the cursor is located.

Of course, there exist other editors which give similar support to the TEX or LATEX user, but most of these have been specifically written to give just this support and in most cases cannot be customized in the same flexible way as Emacs.

Emacs elements

Emacs uses the following terms:

Buffer A chunk of text, usually tied to a file. Editing happens in buffers, but not all buffers have to be files. Sometimes Emacs uses buffers just as a working space, and some buffers have their information from somewhere else, e.g. a directory listing.

Frame What most operating systems now call a *window*. When Emacs grew up it was used on plain old terminals ("glass TTYs") and the notion of windows as in GUI's was not known. And even Java calls them "Frame".

Window A part of a frame that displays a buffer. Windows in the Emacs sense cannot overlap, they tile the Frame (or the screen on a terminal). A buffer can be displayed in more than one window, even in different frames. The cursor position can be different in the different windows. In this way one can easily rearrange portions of text within the same file, or compare different parts of a file.

Major mode The way Emacs deals with a buffer. The major mode (or just mode) influences the way the text may be displayed (syntax coloring), what commands are available, which keystrokes are used for commands, which menus are available etc. There are many modes available, e.g. C-mode, Java-mode, Perl-mode, SGML-mode, plain-tex-mode, and latex-mode. The mode is usually inferred from the filename (extension), but can also be determined in other ways, for instance from the contents of the file. For the extension `.tex` the choice between plain-tex-mode and latex-mode will be made on the existence of `documentclass` or `documentstyle` in the file. It is also possible to state the mode explicitly in the file, or to change it manually after loading the file. Emacs comes with a standard mode for TEX and LATEX, but AucTEX is a more advanced one.

Minor mode A local change to the mode, to change some behaviour, e.g. `auto-fill-mode` for automatically wrapping words, `outline-mode` for showing outlines, or `abbrev-mode` for expansion of abbreviations. Minor modes can usually be toggled.

How Emacs interprets editing actions

Emacs uses lots of control characters, in conjunction with the Escape, ALT or Meta[1] keys for editing actions. The major editing actions are also available as menu entries. None of these have a fixed meaning, however. In line with the general customizability of Emacs every key and menu entry can be given any meaning. This is even true for character keys. In programming language modes (C, Java, Perl) the } key for instance not only inserts this character, but also positions it on a logical place on the line to support proper indentation. In AucTEX the quote character " generates the proper quotes to be used (" or "), in `auto-fill-mode` the space key triggers word wrapping etc.

Emacs uses *keymaps* to get this behaviour. A keymap is a table that binds keystrokes or combinations of them to Emacs functions. The functions can be written in Lisp or be built-in. Therefore one can let a key do almost anything. The "normal" character keys are bound to the function `self-insert-command` which just inserts the character.

AucTEX

AucTEX is a collection of Emacs lisp functions supporting a number of modes for editing TEX and LATEX documents. There are modes for plain TEX, LATEX, and texinfo (the Emacs documentation system), but others like a Context mode could easily be added. AucTEX has the following properties:

□ Syntax coloring for macros, parameters of macros, comment, and environments. Some macros even influence the coloring, e.g., `\textbf{}` shows its parameter in bold.

□ Menus and control keys for inserting chapter, section headers, etc., and environments. AucTEX knows the standard environments and can give the user a choice from these. When using a menu item to insert an environment (Insert Environment), one gets a menu with the known environment. When using the keystrokes to insert the environment one has to type in the environment name. At any point one can type the space key to get a list of possible completions of the name typed thus far. When only one completion is possible, AucTEX will give it, otherwise, it will complete as much as possible and present a list of the possibilities from which one can be chosen with the mouse or by pressing ENTER on the requires choice. But it is also possible to ignore these and type a completely different one, for instance an environment that you have defined yourself. The next time this will be amongst the choices presented (even in the menu). AucTEX knows more about the environments than just the name, but also what parameters are required, so it will prompt for them or reserve space for later insertion. Environments like `itemize` or `enumerate` will have an `\item` automatically inserted; as will happen for *description*, but in the latter case the `\item` will prompt for a label.

□ Administration of `\label` and `\ref`.

□ AucTEX knows about certain packages.

□ Support of running of programs like LATEX, bibtex, makeindex, either on the whole document or parts of it. AucTEX makes intelligent guesses about which command is the next to use. For a new file or a file with recent changes this will be TEX or LATEX, after which bibtex may be necessary. Of course this is the default choice and can always be overridden.

1. One of the expansions of the EMACS acronym is "Escape Meta Alt Control Shift.

◻ Multi-file support. A file that is included in a main document can state which is its master file, such that running LaTeX will automatically give the main document as argument to the commands rather than the included document.

◻ Error processing. After running TeX or LaTeX, AucTeX parses the log file to find error message. With a keystroke or menu entry one can have AucTeX jump to the proper location in the source file.

Reference mechanisms

There are three mechanisms for easy navigation in a LaTeX file: Two come standard with Emacs or AucTeX; the third one is supported by an additional Emacs package.

The standard mechanism is supported by Emacs's imenu package, which gives a menu of the main divisions in a file. For programming languages, these are the functions and/or classes and methods, for LaTeX, this is the section structure. Each section-like command generates one entry in the menu, and selecting this menu entry causes the cursor to jump to that location in the file.

The second mechanism is the so-called *speedbar*: a narrow navigation frame similar to the left-hand panel in MS Window's Explorer. The speedbar frame can be started with a menu entry or an interactive command. When it starts up it displays the files in the current directory. Subdirectory entries can be opened to show their files or be closed, similar to Explorer in MS Windows. However, the same can be done for LaTeX files, or indeed any file type with imenu support. The opened document will then show the outline structure in the same way as the menu that is created by imenu. This gives an overview and makes it easy to navigate quickly through the document.

The speedbar entry for this article looks like the following (for some reason the entries are in reverse order):

```
0:[-] auctex.tex #
 1: > References}
 1: > Customizing Auc\TeX{}
 1: > Reference mechanisms}
 1: > Auc\TeX{}
 1: > How Emacs interprets editing actions}
 1: > Emacs elements}
 1: > What is Emacs?}
```

The third way is to use the additional Emacs package *RefTeX*. This package not only works with AucTeX, but also with the simpler standard TeX-mode that comes with Emacs.

RefTeX is a specialized package for support of labels, references, citations, and the index in LaTeX. RefTeX wraps itself round 4 LaTeX macros: `\label`, `\ref`, `\cite`, and `\index`. Using these macros usually requires looking up different parts of the document and searching through BibTeX database files. RefTeX automates these time-consuming tasks almost entirely. It also provides functions to display the structure of a document (the table of contents) and to move around in this structure quickly. The table of contents will be displayed in a separate window in the same frame as the LaTeX document, and you can jump from here to the location in the document with a single keystroke. It also works on multi-file documents, whereas imenu only supports a single file at a time.

This is the TOC of the current document:

```
1 What is Emacs?
2 Emacs elements
3 How Emacs interprets editing actions
4 Auc\TeX{}
5 Reference mechanisms
6 Customizing Auc\TeX{}
7 References
```

Customizing AucTeX

When starting to edit on a document AucTeX get its information from a number of Emacs Lisp files (`.el` files). Some of its intelligence is built-in, but this applies only to the general structure of TeX and LaTeX documents. For instance the fact that `\begin` and `\end` delimit environments; or the fact that a LaTeX document is characterized by a `\documentclass` or `\documentstyle` command.

For plain TeX documents, this is about all the knowledge that is available, but for LaTeX documents, AucTeX also reads the code in the Emacs lisp file `latex.el` and the ⟨classname⟩`.el` file, e.g. `book.el`. In the same way, `.el` files are read for documentstyle options, packages loaded with `\usepackage`, and included files. Some information can be automatically extracted by AucTeX, such as which commands and environments are defined in the files; other information must be provided by the package writer, such as special treatment of commands or environments. Therefore, AucTeX has two directories for these files, `auto` for automatically generated files and `style` for additional files.

Here is an excerpt of the file `latex.el`:

```
(TeX-add-symbols
  '("arabic" TeX-arg-counter)
  '("label" TeX-arg-define-label)
  '("ref" TeX-arg-ref)
  '("newcommand" TeX-arg-define-macro
         [ "Number of arguments" ] t)
```

It adds support for all the know LaTeX command and knows that `\arabic` should have a counter as argument,

\label defines a label, and \newcommand should prompt for “Number of arguments”. The label support function remembers the label given, so that the ref function can supply a list of known labels.

Here is the main part of book.el. It sets the highest section-like structure to “chapter”.

```
(TeX-add-style-hook "book"
 (function (lambda ()
  (setq LaTeX-largest-level
    (LaTeX-section-level "chapter")))))
```

We can also use these mechanisms to support our own packages. We give three examples: one for a simple command and two for an environment.

Suppose we have a package qa.sty which defines a command \qa with two parameters: a question and an answer. It will have a definition like

```
%% va.sty
\newcommand{\qa}[2]{....}
```

We now create a va.el file that defines the \qa command to prompt for the two parameters. These parameters are added literally (in braces).

```
(TeX-add-style-hook "qa"
  #'(lambda ()
    (TeX-add-symbols
     '("qa"
         TeX-arg-literal "Question"
         TeX-arg-literal "Answer"))))
```

There are other functions to add the arguments without braces, to ask for counters or other macros, etc. The file qa.el will be loaded automatically if \usepackage{qa} is present when it is in the proper directory (style).

The second example is similar, but for an environment rather than a command. This example is from the AucTEX manual. We want to have a package loop that defines an environment loop with three parameters: “From”, “To”, and “Step” are the prompts for the parameters.

```
  \newenvironment{loop}[3]{...}{...}
```

We create loop.el as follows:

```
  (TeX-add-style-hook "loop"
   (function
    (lambda ()
      (LaTeX-add-environments
       '("loop" "From" "To" "Step")))))
```

The last example is the modification of an existing environment. The package enumerate redefines the enumerate environment to accept an optional argument which describes the way the counter is to be formatted. We want the enumerate environment, when added by AucTEX to prompt for this parameter and insert it if a non-empty value is given. In this case we want to do some more non-trivial formatting, e.g. adding \item on the next line with proper indentation. We define our own function LaTeX-enumerate-hook to do the work. It reads the argument with the supplied prompt, test if an empty string is given, and if not, adds it to the document with square brackets added. We then insert the \item command with the function LaTeX-insert-item. As AucTEX starts a new line before the \item, we have to go back to the end of the previous line, delete the newline and any spaces inserted before calling this function.

```
; enumerate.el
(TeX-add-style-hook "enumerate"
 (function
  (lambda ()
    (LaTeX-add-environments
     '("enumerate" LaTeX-enumerate-hook
                "(Optional) Argument: ")))))
  (defun LaTeX-enumerate-hook (name prompt)
   "Insert enumeration with optional argument."
   (let ((arg (read-string prompt)))
     (LaTeX-insert-environment name
             (if (string-equal arg "")
                nil
                (concat "[" arg "]" ))))
   (end-of-line 0)
   (delete-char 1)
   (delete-horizontal-space)
   (LaTeX-insert-item))
```

References

GNU emacs: GNU Emacs: http://www.gnu.org/software/emacs/emacs.html
XEmacs: http://www.xemacs.org/
AucTEX: http://www.sunsite.auc.dk/auctex/
Reftex: http://www.strw.leidenuniv.nl/~dominik/Tools