

AUCT_EX

A sophisticated T_EX environment for Emacs
Version 14.1.0, 2025-07-11_17:22:36

Kresten Krab Thorup
Per Abrahamsen
David Kastrup and others

This manual is for AUCT_EX (version 14.1.0 from 2025-07-11_17:22:36), a sophisticated T_EX environment for Emacs.

Copyright © 1992-1995, 2001, 2002, 2004-2024 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”

Table of Contents

Executive Summary	1
Copying	2
1 Introduction	3
1.1 Overview of AUCT _E X	3
1.2 Installing AUCT _E X	4
1.2.1 Prerequisites	4
1.2.2 Activating the package	5
1.2.3 Providing AUCT _E X as a package	5
1.2.4 Using AUCT _E X from ELPA-devel	6
1.2.5 Using AUCT _E X from local Git repo	6
1.2.6 Customizing	7
1.3 Quick Start	7
1.3.1 Functions for editing TeX files	8
1.3.1.1 Making your T _E X code more readable	8
1.3.1.2 Entering sectioning commands	8
1.3.1.3 Inserting environments	8
1.3.1.4 Inserting macros	8
1.3.1.5 Changing the font	9
1.3.1.6 Other useful features	9
1.3.2 Creating and viewing output, debugging	9
1.3.2.1 One Command for L ^A T _E X, helpers, viewers, and printing	9
1.3.2.2 Choosing an output format	10
1.3.2.3 Debugging L ^A T _E X	10
1.3.2.4 Running L ^A T _E X on parts of your document	11
2 Editing the Document Source	12
2.1 Insertion of Quotes, Dollars, and Braces	12
2.2 Inserting Font Specifiers	15
2.3 Inserting chapters, sections, etc.	16
2.4 Inserting Environment Templates	18
2.4.1 Equations	20
2.4.2 Floats	21
2.4.3 Itemize-like Environments	22
2.4.4 Tabular-like Environments	22
2.4.5 Customizing Environments	22
2.5 Entering Mathematics	23
2.6 Completion	25
2.7 Marking Environments, Sections, or Texinfo Nodes	28
2.7.1 L ^A T _E X Commands for Marking Environments and Sections	28
2.7.2 Texinfo Commands for Marking Environments and Sections	29

2.8	Commenting	29
2.9	Indenting	30
2.10	Filling	33
3	Controlling Screen Display	36
3.1	Font Locking	36
3.1.1	Fontification of macros	37
3.1.2	Fontification of quotes	40
3.1.3	Fontification of mathematical constructs	41
3.1.4	Verbatim macros and environments	42
3.1.5	Faces used by font-latex	42
3.1.6	Known fontification problems	42
3.2	Folding Macros and Environments	43
3.3	Outlining the Document	48
3.4	Narrowing	49
3.5	Prettifying	49
4	Starting Processors, Viewers and Other Programs	50
4.1	Executing Commands	50
4.1.1	Starting a Command on a Document or Region	50
4.1.2	Selecting and Executing a Command	52
4.1.3	Options for T _E X Processors	55
4.2	Viewing the Formatted Output	58
4.2.1	Starting Viewers	58
4.2.2	Forward and Inverse Search	60
4.3	Catching the errors	62
4.3.1	Controlling warnings to be reported	63
4.3.2	List of all errors and warnings	64
4.4	Checking for problems	64
4.5	Controlling the output	65
4.6	Cleaning intermediate and output files	66
4.7	Documentation about macros and packages	66
5	Customization and Extension	68
5.1	Modes and Hooks	68
5.2	Multifile Documents	68
5.3	Automatic Parsing of T _E X Files	70
5.4	Language Support	72
5.4.1	Using AUCT _E X with European Languages	72
5.4.1.1	Typing and Displaying Non-ASCII Characters	73
5.4.1.2	Style Files for Different Languages	73
5.4.2	Using AUCT _E X with Japanese T _E X	75
5.5	Automatic Customization	77
5.5.1	Automatic Customization for the Site	78

5.5.2 Automatic Customization for a User	78
5.5.3 Automatic Customization for a Directory	79
5.6 Writing Your Own Style Support	79
5.6.1 A Simple Style File	79
5.6.2 Adding Support for Macros	81
5.6.3 Adding Support for Environments	85
5.6.4 Adding or Examining Other Information	88
5.6.4.1 Adding bibliographies in style hooks	88
5.6.4.2 Examining Package/Class Options	88
5.6.4.3 Adding Support for Option Completion	89
5.6.5 Automatic Extraction of New Things	90

Appendix A Copying, Changes, Development, FAQ, Texinfo Mode 92

A.1 Copying this Manual	92
A.1.1 GNU Free Documentation License	92
A.2 Changes and New Features	99
A.3 Future Development	99
A.3.1 Mid-term Goals	100
A.3.2 Wishlist	101
A.3.3 Bugs	103
A.4 Frequently Asked Questions	103
A.5 Features specific to AUCT _E X's Texinfo major mode	105
A.5.1 How AUCT _E X and the native mode work together	106
A.5.2 Where the native mode is superseded	106
A.5.3 Where key bindings are mapped to the native mode	107
A.5.4 Which native mode key bindings are missing	108

Indices 109

Key Index	109
Function Index	110
Variable Index	112
Concept Index	115

Executive Summary

AUCT_EX is an integrated environment for editing L_AT_EX, ConT_EXt, docT_EX, Texinfo, and T_EX files.

Although AUCT_EX contains a large number of features, there are no reasons to despair. You can continue to write T_EX and L_AT_EX documents the way you are used to, and only start using the multiple features in small steps. AUCT_EX is not monolithic, each feature described in this manual is useful by itself, but together they provide an environment where you will make very few L_AT_EX errors, and makes it easy to find the errors that may slip through anyway.

It is a good idea to make a printout of AUCT_EX's reference card `tex-ref.tex` or one of its typeset versions.

If you want to make AUCT_EX aware of style files and multifile documents right away, insert the following in your init file (usually `~/.emacs.d/init.el`).

```
(setq TeX-auto-save t)
(setq TeX-parse-self t)
(setq-default TeX-master nil)
```

Another thing you should enable is RefT_EX, a comprehensive solution for managing cross references, bibliographies, indices, document navigation and a few other things. (See Section “Installation” in *The RefT_EX manual*.)

For detailed information about the preview-latex subsystem of AUCT_EX, see Section “Introduction” in *The preview-latex Manual*.

There is a mailing list for general discussion about AUCT_EX: write a mail with “subscribe” in the subject to `auctex-request@gnu.org` to join it. Send contributions to `auctex@gnu.org`.

Bug reports should go to `bug-auctex@gnu.org`, suggestions for new features, and pleas for help should go to either `auctex-devel@gnu.org` (the AUCT_EX developers), or to `auctex@gnu.org` if they might have general interest. Please use the command `M-x TeX-submit-bug-report` RET to report bugs if possible. You can subscribe to a low-volume announcement list by sending “subscribe” in the subject of a mail to `info-auctex-request@gnu.org`.

Copying

AUCT_EX primarily consists of Lisp files for Emacs, but there are also installation scripts and files and T_EX support files. All of those are *free*; this means that everyone is free to use them and free to redistribute them on a free basis. The files of AUCT_EX are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the files that constitute AUCT_EX, that you receive source code or else can get it if you want it, that you can change these files or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of parts of AUCT_EX, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for AUCT_EX. If any parts are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the files currently being distributed as part of AUCT_EX are found in the General Public Licenses that accompany them. This manual specifically is covered by the GNU Free Documentation License (see Section A.1 [Copying this Manual], page 92).

1 Introduction

1.1 Overview of AUCT_EX

AUCT_EX is a comprehensive customizable integrated environment for writing input files for T_EX, L^AT_EX, ConT_EXt, Texinfo, and docT_EX using Emacs.

It supports you in the insertion of macros, environments, and sectioning commands by providing completion alternatives and prompting for parameters. It automatically indents your text as you type it and lets you format a whole file at once. The outlining and folding facilities provide you with a focused and clean view of your text.

AUCT_EX lets you process your source files by running T_EX and related tools (such as output filters, post processors for generating indices and bibliographies, and viewers) from inside Emacs. AUCT_EX lets you browse through the errors T_EX reported, while it moves the cursor directly to the reported error, and displays some documentation for that particular error. This will even work when the document is spread over several files.

One component of AUCT_EX that L^AT_EX users will find attractive is preview-latex, a combination of folding and in-source previewing that provides true “What You See Is What You Get” experience in your sourcebuffer, while letting you retain full control.

More detailed information about the features and usage of AUCT_EX can be found in the remainder of this manual.

AUCT_EX is written entirely in Emacs Lisp, and hence you can easily add new features for your own needs. It is a GNU project and distributed under the ‘GNU General Public License Version 3’.

AUCT_EX is a package distributed at ELPA, Emacs Lisp Package Archive. You can manage it in Emacs package manager. (see Section “Packages” in `emacs`)

WWW users may want to check out the AUCT_EX page at <https://www.gnu.org/software/auctex/> and <https://elpa.gnu.org/packages/auctex.html>.

If you are considering upgrading AUCT_EX, the recent changes are described on the latter of the above WWW sites. You can see the same change logs in `NEWS.org` file available at your ELPA AUCT_EX contents directory, typically `~/.emacs.d/elpa/auctex-x.y.z/` where ‘x.y.z’ is the version number of the installed AUCT_EX.

If you want to discuss AUCT_EX with other users or its developers, there are several mailing lists you can use.

Send a mail with the subject “subscribe” to `auctex-request@gnu.org` in order to join the general discussion list for AUCT_EX. Articles should be sent to `auctex@gnu.org`. In a similar way, you can subscribe to the `info-auctex@gnu.org` list for just getting important announcements about AUCT_EX. The list `bug-auctex@gnu.org` is for bug reports which you should usually file with the `M-x TeX-submit-bug-report RET` command. If you want to address the developers of AUCT_EX themselves with technical issues, they can be found on the discussion list `auctex-devel@gnu.org`.

1.2 Installing AUCT_EX

AUCT_EX is a package distributed at ELPA, Emacs Lisp Package Archive. To install AUCT_EX, simply do *M-x list-packages* RET, mark the auctex package for installation with *i*, and hit *x* to execute the installation procedure. That's all.

Caution. If you have installed former AUCT_EX as regular tarball release, uninstall it and delete the initialization codes

```
(load "auctex.el" nil t t)
(load "preview-latex.el" nil t t)
```

in your init file. Otherwise you'll get into troubles.

`use-package` users can use this simple recipe in their `user-init-file` which essentially does the same as the manual installation explained above.

```
(use-package auctex
  :ensure t)
```

For past ELPA releases, see <https://elpa.gnu.org/packages/auctex.html>. Once the installation is completed, you can skip the rest of this section and proceed to Section 1.3 [Quick Start], page 7.

1.2.1 Prerequisites

- GNU Emacs 28.1 or higher

Using `preview-latex` requires a version of Emacs compiled with image support.

Windows Precompiled versions are available from <https://ftp.gnu.org/gnu/emacs/windows/>.

macOS For an overview of precompiled versions of Emacs for macOS see for example <https://www.emacswiki.org/emacs/EmacsForMacOS>.

GNU/Linux

Most GNU/Linux distributions nowadays provide a recent variant of Emacs via their package repositories.

Self-compiled

Compiling Emacs yourself requires a C compiler and a number of tools and development libraries. Details are beyond the scope of this manual. Instructions for checking out the source code can be found at <https://savannah.gnu.org/git/?group=emacs>.

- A working T_EX installation

Well, AUCT_EX would be pointless without that. `preview-latex` requires Dvips or `dvipng` for its operation in DVI mode. The default configuration of AUCT_EX is tailored for T_EX Live-based distributions, but can be adapted easily.

- A recent Ghostscript

This is needed for operation of `preview-latex` in both DVI and PDF mode. Ghostscript version 7.07 or newer is required.

For some known issues with various software, see Section “Known problems” in *the preview-latex manual*.

1.2.2 Activating the package

You can detect the successful activation of AUCTeX and preview-latex in the menus after loading a L^AT_EX file like `circ.tex`: AUCTeX then gives you a ‘Command’ menu, and preview-latex gives you a ‘Preview’ menu.

For site-wide activation in GNU Emacs, see Section 1.2.3 [Advice for package providers], page 5.

Once activated, the modes provided by AUCTeX are used per default for all supported file types, namely `plain-tex-mode`, `latex-mode`, `doctex-mode` and `texinfo-mode`. This might not match your preference. You can have control over which AUCTeX mode is activated per file types by `TeX-modes` option. For example, you can use Emacs built-in `plain-tex-mode` for plain T_EX files while you can use AUCTeX L^AT_EX-mode for L^AT_EX files.

TeX-modes

[User Option]

List of Emacs built-in T_EX modes redirected to AUCTeX modes. If you prefer a particular built-in mode over AUCTeX mode, remove it from this list. Type

```
M-x customize-option RET TeX-modes RET
```

to manipulate the contents of `TeX-modes`.

Don’t remove `tex-mode` from `TeX-modes` unless you set `TeX-modes` empty to disable AUCTeX completely, otherwise it results in inconsistent behavior.

On Emacs 29 and later, AUCTeX uses either `major-mode-remap-defaults` or `major-mode-remap-alist` for redirection. But we recommend not to customize them directly because the customization code for `TeX-modes` takes care of some other compatibility issues.

When there is a site-wide installation of AUCTeX and you don’t want to use it, you can disable it by

```
(push '(auctex nil) package-load-list)
```

in your early init file (see Section “Early Init File” in `emacs`). (We recommend this treatment over setting `TeX-modes` to `nil`, because it doesn’t leave unused autoloader persisted.)

It is no longer possible to disable the site-wide installation by

```
(unload-feature 'tex-site)
```

, so don’t use it. This was the instruction described in former versions of this document, but now it causes error.

1.2.3 Providing AUCTeX as a package

As a package provider, you should make sure that your users will be served best according to their intentions, and keep in mind that a system might be used by more than one user, with different preferences.

There are people that prefer the built-in Emacs modes for editing T_EX files, in particular plain T_EX users. There are various ways to tell AUCTeX even after auto-activation that it should not get used, and they are described in the Section 1.2.2 [Activating the package], page 5.

So if you have users that don’t want to use the preinstalled AUCTeX, they can easily get rid of it. Installing AUCTeX as site-wide default is therefore a good choice.

You can install ELPA AUCT_EX package under a directory listed in `package-directory-list` to have site-wide default.

1.2.4 Using AUCT_EX from ELPA-devel

It is possible to use the latest development version of AUCT_EX conveniently as a package installed from GNU-devel ELPA. This package tracks the latest change in AUCT_EX Git repository and is intended for brave users who want to test the distribution and report possible issues. The following addition to `user-init-file` instructs Emacs to change the archive AUCT_EX is installed from:

```
(add-to-list 'package-archives
             '("elpa-devel" . "https://elpa.gnu.org/devel/") t)
(setq package-archive-priority
      '(("elpa" . 10)
        ("elpa-devel" . 5)))
(setq package-pinned-packages
      '((auctex . "elpa-devel")))
```

In a nutshell, the code adds the new archive to the list of known archives under the name ‘`elpa-devel`’, gives it a lower priority than the regular archive, and instructs Emacs to fetch only AUCT_EX from the new archive and don’t bother with other packages installed.

1.2.5 Using AUCT_EX from local Git repo

It is also possible to use AUCT_EX directly from a local Git repository. Let’s assume you have your Git repositories under ‘`~/development/`’.

First, you have to fetch a copy of the AUCT_EX Git repository. In a shell, change directory to ‘`~/development/`’ and do:

```
git clone https://git.savannah.gnu.org/git/auctex.git
```

Now change directory to ‘`~/development/auctex`’ and run

```
make
```

Now you have to tell Emacs about the plan. Put the following code in your init file:

```
(load "~/development/auctex/auctex-autoloads.el" nil t t)
(with-eval-after-load 'info
  (add-to-list 'Info-additional-directory-list
               "~/development/auctex/doc"))
```

and you’re finished.

Note for ‘`use-package`’ users: you can wrap the above recipe with `use-package` like this:

```
(use-package auctex
  :init
  (load "~/development/auctex/auctex-autoloads.el" nil t t)
  (with-eval-after-load 'info
    (add-to-list 'Info-additional-directory-list
                  "~/development/auctex/doc"))))
```

1.2.6 Customizing

Most of the site-specific customization should already have happened during installation of AUCTeX. Any further customization can be done with customization buffers directly in Emacs. Just type *M-x customize-group* RET *AUCTeX* RET to open the customization group for AUCTeX or use the menu entries provided in the mode menus. Editing the file `tex-site.el` as suggested in former versions of AUCTeX should not be done anymore because the installation routine will overwrite those changes.

You might check some options with a special significance. They are accessible directly by typing *M-x customize-option* RET *option* RET.

TeX-macro-global

[User Option]

Directories containing the site's T_EX style files.

Normally, AUCTeX will only allow you to complete macros and environments which are built-in, specified in AUCTeX style files or defined by yourself. If you issue the *M-x TeX-auto-generate-global* command after loading AUCTeX, you will be able to complete on all macros available in the standard style files used by your document. To do this, you must set this variable to a list of directories where the standard style files are located. The directories will be searched recursively, so there is no reason to list subdirectories explicitly. Automatic configuration will already have set the variable for you if it could use the program `kpsewhich`. In this case you normally don't have to alter anything.

Note that `TeX-auto-generate-global` is not so smart and it can introduce unexpected side effects as discussed in <https://lists.gnu.org/r/auctex/2021-01/msg00037.html>.

1.3 Quick Start

AUCTeX is a powerful program offering many features and configuration options. If you are new to AUCTeX this might be deterrent. Fortunately you do not have to learn everything at once. This Quick Start Guide will give you the knowledge of the most important commands and enable you to prepare your first L^AT_EX document with AUCTeX after only a few minutes of reading.

In this introduction, we assume that AUCTeX is already installed on your system. If this is not the case, you should read the installation instructions in this manual (see Section 1.2 [Installation], page 4). We also assume that you are familiar with the way keystrokes are written in Emacs manuals. If not, have a look at the Emacs Tutorial in the Help menu.

In order to get support for many of the L^AT_EX packages you will use in your documents, you should enable document parsing as well, which can be achieved by putting

```
(setq TeX-auto-save t)
(setq TeX-parse-self t)
```

into your init file. Finally, if you often use `\include` or `\input`, you should make AUCTeX aware of the multifile document structure. You can do this by inserting

```
(setq-default TeX-master nil)
```

into your init file. Each time you open a new file, AUCTeX will then ask you for a master file.

This Quick Start Guide covers two main topics: First we explain how AUCTeX helps you in editing your input file for TeX, L^AT_EX, and some other formats. Then we describe the functions that AUCTeX provides for processing the input files with L^AT_EX, BibTeX, etc., and for viewing and debugging.

1.3.1 Functions for editing TeX files

1.3.1.1 Making your TeX code more readable

AUCTeX can do syntax highlighting of your source code, that means commands will get special colors or fonts. This is enabled by default. You can disable it locally by typing *M-x font-lock-mode* RET.

AUCTeX will indent new lines to indicate their syntactical relationship to the surrounding text. For example, the text of a `\footnote` or text inside of an environment will be indented relative to the text around it. If the indenting has gotten wrong after adding or deleting some characters, use TAB to reindent the line, *M-q* for the whole paragraph, or *M-x LaTeX-fill-buffer* RET for the whole buffer.

1.3.1.2 Entering sectioning commands

Insertion of sectioning macros, that is `\chapter`, `\section`, `\subsection`, etc. and accompanying `\label` commands may be eased by using *C-c C-s*. You will be asked for the section level. As nearly everywhere in AUCTeX, you can use the TAB or SPC key to get a list of available level names, and to auto-complete what you started typing. Next, you will be asked for the printed title of the section, and last you will be asked for a label to be associated with the section.

1.3.1.3 Inserting environments

Similarly, you can insert environments, that is `\begin{}`–`\end{}` pairs: Type *C-c C-e*, and select an environment type. Again, you can use TAB or SPC to get a list, and to complete what you type. Actually, the list will not only provide standard L^AT_EX environments, but also take your `\documentclass` and `\usepackage` commands into account if you have parsing enabled by setting `TeX-parse-self` to `t`. If you use a couple of environments frequently, you can use the up and down arrow keys (or *M-p* and *M-n*) in the minibuffer to get back to the previously inserted commands.

Some environments need additional arguments. Often, AUCTeX knows about this and asks you to enter a value.

1.3.1.4 Inserting macros

C-c C-m, or simply *C-c* RET will give you a prompt that asks you for a L^AT_EX macro. You can use TAB for completion, or the up/down arrow keys (or *M-p* and *M-n*) to browse the command history. In many cases, AUCTeX knows which arguments a macro needs and will ask you for that. It even can differentiate between mandatory and optional arguments—for details, see Section 2.6 [Completion], page 25.

An additional help for inserting macros is provided by the possibility to complete macros right in the buffer. With point at the end of a partially written macro, you can complete it by typing *M-TAB*.

1.3.1.5 Changing the font

AUCTEX provides convenient keyboard shortcuts for inserting macros which specify the font to be used for typesetting certain parts of the text. They start with *C-c C-f*, and the last *C-* combination tells AUCTEX which font you want:

C-c C-f C-b

Insert **bold face** ‘`\textbf{★}`’ text.

C-c C-f C-i

Insert *italics* ‘`\textit{★}`’ text.

C-c C-f C-e

Insert *emphasized* ‘`\emph{★}`’ text.

C-c C-f C-s

Insert *slanted* ‘`\textsl{★}`’ text.

C-c C-f C-r

Insert roman ‘`\textrm{★}`’ text.

C-c C-f C-f

Insert sans serif ‘`\textsf{★}`’ text.

C-c C-f C-t

Insert typewriter ‘`\texttt{★}`’ text.

C-c C-f C-c

Insert SMALL CAPS ‘`\textsc{★}`’ text.

C-c C-f C-d

Delete the innermost font specification containing point.

If you want to change font attributes of existing text, mark it as an active region, and then invoke the commands. If no region is selected, the command will be inserted with empty braces, and you can start typing the changed text.

Most of those commands will also work in math mode, but then macros like `\mathbf` will be inserted.

1.3.1.6 Other useful features

AUCTEX also tries to help you when inserting the right “quote” signs for your language, dollar signs to typeset math, or pairs of braces. It offers shortcuts for commenting out text (*C-c ;* for the current region or *C-c %* for the paragraph you are in). The same keystrokes will remove the % signs, if the region or paragraph is commented out yet. With `TeX-fold-mode`, you can hide certain parts (like footnotes, references etc.) that you do not edit currently. Support for Emacs’ outline mode is provided as well. And there’s more, but this is beyond the scope of this Quick Start Guide.

1.3.2 Creating and viewing output, debugging

1.3.2.1 One Command for L^AT_EX, helpers, viewers, and printing

If you have typed some text and want to run L^AT_EX (or T_EX, or other programs—see below) on it, type *C-c C-c*. If applicable, you will be asked whether you want to save changes,

and which program you want to invoke. In many cases, the choice that AUCTeX suggests will be just what you want: first `latex`, then a viewer. If a `latex` run produces or changes input files for `makeindex`, the next suggestion will be to run that program, and AUCTeX knows that you need to run `latex` again afterwards—the same holds for BibTeX.

When no processor invocation is necessary anymore, AUCTeX will suggest to run a viewer, or you can chose to create a PostScript file using `dvips`, or to directly print it.

Actually, there is another command which comes in handy to compile documents: type `C-c C-a` (`TeX-command-run-all`) and AUCTeX will compile the document for you until it is ready and then run the viewer. This is the same as issuing repeatedly `C-c C-c` and letting AUCTeX guess the next command to run.

At this place, a warning needs to be given: First, although AUCTeX is really good in detecting the standard situations when an additional `latex` run is necessary, it cannot detect it always. Second, the creation of PostScript files or direct printing currently only works when your output file is a DVI file, not a PDF file.

Ah, you didn't know you can do both? That brings us to the next topic.

1.3.2.2 Choosing an output format

From a L^AT_EX file, you can produce DVI output, or a PDF file directly *via* `pdflatex`. You can switch on source specials for easier navigation in the output file, or tell `latex` to stop after an error (usually `--noninteractive` is used, to allow you to detect all errors in a single run).

These options are controlled by toggles, the keystrokes should be easy to memorize:

`C-c C-t C-p`

This command toggles between DVI and PDF output

`C-c C-t C-i`

toggles interactive mode

`C-c C-t C-s`

toggles SyncTeX (or source specials) support

`C-c C-t C-o`

toggles usage of Omega/lambda.

There is also another possibility: compile the document with `tex` (or `latex`) and then convert the resulting DVI file to PDF using `dvips-ps2pdf` sequence or `dvipdfmx` command. If you want to go by this route, customize `TeX-PDF-from-DVI` option. Then AUCTeX will suggest you to run the appropriate command when you type `C-C C-c`. For details, see Section 4.1.3 [Processor Options], page 55.

1.3.2.3 Debugging L^AT_EX

When AUCTeX runs a program, it creates an output buffer in which it displays the output of the command. If there is a syntactical error in your file, `latex` will not complete successfully. AUCTeX will tell you that, and you can get to the place where the first error occurred by pressing `C-c `` (the last character is a backtick). The view will be split in two windows, the output will be displayed in the lower buffer, and both buffers will be centered around the place where the error occurred. You can then try to fix it in the document buffer, and

use the same keystrokes to get to the next error. This procedure may be repeated until all errors have been dealt with. By pressing `C-c C-w` (`TeX-toggle-debug-bad-boxes`) you can toggle whether AUCTEX should notify you of overfull and underfull boxes in addition to regular errors.

Issue `M-x TeX-error-overview` RET to see a nicely formatted list of all errors and warnings reported by the compiler.

If a command got stuck in a seemingly infinite loop, or you want to stop execution for other reasons, you can use `C-c C-k` (for “kill”). Similar to `C-l`, which centers the buffer you are in around your current position, `C-c C-l` centers the output buffer so that the last lines added at the bottom become visible.

1.3.2.4 Running L^AT_EX on parts of your document

If you want to check how some part of your text looks like, and do not want to wait until the whole document has been typeset, then mark it as a region and use `C-c C-r`. It behaves just like `C-c C-c`, but it only uses the document preamble and the region you marked.

If you are using `\include` or `\input` to structure your document, try `C-c C-b` while you are editing one of the included files. It will run `latex` only on the current buffer, using the preamble from the master file.

2 Editing the Document Source

The most commonly used commands/macros of AUCTeX are those which simply insert templates for often used TeX, LaTeX, or ConTeXt constructs, like font changes, handling of environments, etc. These features are very simple, and easy to learn, and help you avoid mistakes like mismatched braces, or ‘\begin{ }’-‘\end{ }’ pairs.

Apart from that this chapter contains a description of some features for entering more specialized sorts of text, for formatting the source by indenting and filling and for navigating through the document.

2.1 Insertion of Quotes, Dollars, and Braces

Quotation Marks

In TeX, literal double quotes “like this” are seldom used, instead two single quotes are used ‘‘like this’’. To help you insert these efficiently, AUCTeX allows you to continue to press " to insert two single quotes. To get a literal double quote, press " twice.

TeX-insert-quote *count* [Command]
 (") Insert the appropriate quote marks for TeX.

Inserts the value of **TeX-open-quote** (normally ‘‘’) or **TeX-close-quote** (normally ‘’’) depending on the context. With prefix argument, always inserts “” characters.

TeX-open-quote [User Option]
 String inserted by typing " to open a quotation. (See Section 5.4.1 [European], page 72, for language-specific quotation mark insertion.)

TeX-close-quote [User Option]
 String inserted by typing " to close a quotation. (See Section 5.4.1 [European], page 72, for language-specific quotation mark insertion.)

TeX-quote-after-quote [User Option]
 Determines the behavior of ". If it is non-nil, typing " will insert a literal double quote. The respective values of **TeX-open-quote** and **TeX-close-quote** will be inserted after typing " once again.

The ‘babel’ package provides special support for the requirements of typesetting quotation marks in many different languages. If you use this package, either directly or by loading a language-specific style file, you should also use the special commands for quote insertion instead of the standard quotes shown above. AUCTeX is able to recognize several of these languages and will change quote insertion accordingly. See Section 5.4.1 [European], page 72, for details about this feature and how to control it.

In case you are using the ‘csquotes’ package, you should customize **LaTeX-csquotes-open-quote**, **LaTeX-csquotes-close-quote** and **LaTeX-csquotes-quote-after-quote**. The quotation characters will only be used if both variables—**LaTeX-csquotes-open-quote** and **LaTeX-csquotes-close-quote**—are non-empty strings. But then the ‘csquotes’-related values will take precedence over the language-specific ones.

Dollar Signs

In AUCTeX, dollar signs should match like they do in T_EX. This has been partially implemented, we assume dollar signs always match within a paragraph. By default, the first ‘\$’ you insert in a paragraph will do nothing special. The second ‘\$’ will match the first. This will be indicated by moving the cursor temporarily over the first dollar sign.

TeX-insert-dollar *arg* [Command]

(*\$*) Insert dollar sign (or another math delimiter).

Show matching dollar sign if this dollar sign end the T_EX math mode.

With optional *arg*, insert that many dollar signs.

T_EX and L^AT_EX users often look for a way to insert inline equations like ‘\$...\$’ or ‘\(...\)’ simply typing \$. AUCTeX helps them through the customizable variable **TeX-electric-math**.

TeX-electric-math [User Option]

If the variable is non-**nil** and you type \$ outside math mode, AUCTeX will automatically insert the opening and closing symbols for an inline equation and put the point between them. The opening symbol will blink when **blink-matching-paren** is non-**nil**. If **TeX-electric-math** is **nil**, typing \$ simply inserts ‘\$’ at point, this is the default.

Besides **nil**, possible values for this variable are (“\$” . “\$”) for T_EX inline equations ‘\$...\$’, and (“\ (“ . “\)”) for L^AT_EX inline equations ‘\(...\)’.

In addition, when the variable is non-**nil** and there is an active region outside math mode, typing \$ will put around the active region symbols for opening and closing inline equation and keep the region active, leaving point after the closing symbol. By pressing repeatedly \$ while the region is active you can toggle between an inline equation, a display equation, and no equation. To be precise, ‘\$...\$’ is replaced by ‘\$\$...\$\$’ in plain-TeX-mode or ‘\[...\]’ in LaTeX-mode and docTeX-mode, whereas ‘\(...\)’ is replaced by ‘\[\...\]’.

If you want to automatically insert ‘\$...\$’ in plain T_EX files, and ‘\(...\)’ in L^AT_EX files by pressing \$, add the following to your init file

```
(add-hook 'plain-TeX-mode-hook
  (lambda () (setq-local TeX-electric-math
    (cons "$" "$"))))

(add-hook 'LaTeX-mode-hook
  (lambda () (setq-local TeX-electric-math
    (cons "\\ (" "\\)"))))
```

Math mode which didn't start with dollar(s) shouldn't be closed with dollar.

TeX-refuse-unmatched-dollar [User Option]

This option *has no effect* when **TeX-electric-math** is non-**nil**.

This option determines the behavior when the user types \$ at a position where AUCTeX thinks that it is in math mode which didn't start with dollar(s).

When this option is **nil**, AUCTeX behaves in the same way as non-math mode, assuming that the user knows it isn't in math mode actually. This is the default.

When this option is non-nil, AUCTEX refuses to insert ‘\$’ to prevent unmatched dollar.

Note that Texinfo mode does nothing special for \$. It inserts dollar sign(s) just in the same way as the other normal keys do.

AUCTEX provides the command `LaTeX-make-inline` which converts the display math environment at point to inline math.

LaTeX-make-inline [Command]
 Convert L^AT_EX display math environment at point to inline math. This command replaces the enclosing math environment such as ‘\[…\]’ or ‘\begin{equation}...\end{equation}’ with the value of `TeX-electric-math` or ‘\$...\$’ by default. (It may not work correctly in doc_TE_X.)

Braces

To avoid unbalanced braces, it is useful to insert them pairwise. You can do this by typing `C-c {`.

TeX-insert-braces [Command]
 (`C-c {`) Make a pair of braces and position the cursor to type inside of them. If there is an active region, put braces around it and leave point after the closing brace.

When writing complex math formulas in L^AT_EX documents, you sometimes need to adjust the size of braces with pairs of macros like ‘\left’-‘\right’, ‘\bigl’-‘\bigr’ and so on. You can avoid unbalanced pairs with the help of `TeX-insert-macro`, bound to `C-c C-m` or `C-c RET` (see Section 2.6 [Completion], page 25). If you insert left size adjusting macros such as ‘\left’, ‘\bigl’ etc. with `TeX-insert-macro`, it asks for left brace to use and supplies automatically right size adjusting macros such as ‘\right’, ‘\bigr’ etc. and corresponding right brace in addition to the intended left macro and left brace.

The completion by `TeX-insert-macro` also applies when entering macros such as ‘\langle’, ‘\lfloor’ and ‘\lceil’, which produce the left part of the paired braces. For example, inserting ‘\lfloor’ by `C-c C-m` is immediately followed by the insertion of ‘\rfloor’. In addition, if the point was located just after ‘\left’ or its friends, the corresponding ‘\right’ etc. will be inserted in front of ‘\rfloor’. In both cases, active region is honored.

As a side effect, when `LaTeX-math-mode` (see Section 2.5 [Mathematics], page 23) is on, just typing ‘(’ inserts not only ‘\langle’, but also ‘\rangle’.

If you do not like such auto completion at all, it can be disabled by a user option.

TeX-arg-right-insert-p [User Option]
 If this option is turned off, the automatic supply of the right macros and braces is suppressed.

When you edit L^AT_EX documents, you can enable automatic brace pairing when typing ‘(’, ‘{’ and ‘[’.

LaTeX-electric-left-right-brace [User Option]

If this option is on, just typing `(`, `{` or `[` immediately adds the corresponding right brace `)`, `}` or `]`. The point is left after the opening brace. If there is an active region, braces are put around it.

They recognize the preceding backslash or size adjusting macros such as `\left`, `\bigl` etc., so the following completions will occur:

- (when typing single left brace)
 - `'(-> '()`
 - `'{ -> '{}'`
 - `'[-> '[]'`
- (when typing left brace just after a backslash)
 - `'\(' -> '\(\)'`
 - `'\{' -> '\{\}'`
 - `'\[-> '\[\\]'`
- (when typing just after `\left` or `\bigl`)
 - `'\left(' -> '\left(\right)'`
 - `'\bigl[' -> '\bigl[\bigr]'`
- (when typing just after `\Bigl\`)
 - `'\Bigl\{' -> '\Bigl\{\Bigr\}'`

This auto completion feature may be a bit annoying when editing an already existing L^AT_EX document. In that case, use `C-u 1` or `C-q` before typing `(`, `{` or `[`. Then no completion is done and just a single left brace is inserted. In fact, with optional prefix *arg*, just that many open braces are inserted without any completion.

2.2 Inserting Font Specifiers

Perhaps the most used keyboard commands of AUCT_EX are the short-cuts available for easy insertion of font changing macros.

If you give an argument (that is, type `C-u`) to the font command, the innermost font will be replaced, i.e. the font in the T_EX group around point will be changed. The following table shows the available commands, with `*` indicating the position where the text will be inserted.

`C-c C-f C-b`

Insert **bold face** `'\textbf{*}'` text.

`C-c C-f C-m`

Insert medium face `'\textmd{*}'` text.

`C-c C-f C-i`

Insert *italics* `'\textit{*}'` text.

`C-c C-f C-e`

Insert *emphasized* `'\emph{*}'` text.

`C-c C-f C-s`

Insert *slanted* `'\textsl{*}'` text.

C-c C-f C-r

Insert roman ‘`\textrm{★}`’ text.

C-c C-f C-f

Insert sans serif ‘`\textsf{★}`’ text.

C-c C-f C-t

Insert typewriter ‘`\texttt{★}`’ text.

C-c C-f C-c

Insert SMALL CAPS ‘`\textsc{★}`’ text.

C-c C-f C-l

Insert upper lower case ‘`\textulc{★}`’ text.

C-c C-f C-w

Insert SWASH ‘`\textsw{★}`’ text.

C-c C-f C-n

Insert normal ‘`\textnormal{★}`’ text.

C-c C-f C-d

Delete the innermost font specification containing point.

TeX-font *replace what*

[Command]

(*C-c C-f*) Insert template for font change command.

If *replace* is non-nil, replace current font. *what* determines the font to use, as specified by **TeX-font-list**.

TeX-font-list

[User Option]

List of fonts used by **TeX-font**.

Each entry is a list with three elements. The first element is the key to activate the font. The second element is the string to insert before point, and the third element is the string to insert after point. An optional fourth element means always replace if non-nil.

LaTeX-font-list

[User Option]

List of fonts used by **TeX-font** in LaTeX mode. It has the same structure as **TeX-font-list**.

2.3 Inserting chapters, sections, etc.

Insertion of sectioning macros, that is ‘`\chapter`’, ‘`\section`’, ‘`\subsection`’, etc. and accompanying ‘`\label`’s may be eased by using *C-c C-s*. This command is highly customizable, the following describes the default behavior.

When invoking you will be asked for a section macro to insert. An appropriate default is automatically selected by AUCTeX, that is either: at the top of the document; the top level sectioning for that document style, and any other place: The same as the last occurring sectioning command.

Next, you will be asked for the actual name of that section, and last you will be asked for a label to be associated with that section. The label will be prefixed by the value specified in **LaTeX-section-hook**.

LaTeX-section arg [Command]

(*C-c C-s*) Insert a sectioning command.

Determine the type of section to be inserted, by the argument *arg*.

- If *arg* is `nil` or missing, use the current level.
- If *arg* is a list (selected by *C-u*), go downward one level.
- If *arg* is negative, go up that many levels.
- If *arg* is positive or zero, use absolute level:
 - + 0 : part
 - + 1 : chapter
 - + 2 : section
 - + 3 : subsection
 - + 4 : subsubsection
 - + 5 : paragraph
 - + 6 : subparagraph

The following variables can be set to customize the function.

LaTeX-section-hook

Hooks to be run when inserting a section.

LaTeX-section-label

Prefix to all section references.

The precise behavior of **LaTeX-section** is defined by the contents of **LaTeX-section-hook**.

LaTeX-section-hook [User Option]

List of hooks to run when a new section is inserted.

The following variables are set before the hooks are run

LaTeX-level

Numeric section level, default set by prefix *arg* to **LaTeX-section**.

LaTeX-name

Name of the sectioning command, derived from **LaTeX-level**.

LaTeX-title

The title of the section, default to an empty string.

LaTeX-toc

Entry for the table of contents list, default `nil`.

LaTeX-done-mark

Position of point afterwards, default `nil` meaning after the inserted text.

A number of hooks are already defined. Most likely, you will be able to get the desired functionality by choosing from these hooks.

LaTeX-section-heading

Query the user about the name of the sectioning command. Modifies **LaTeX-level** and **LaTeX-name**.

LaTeX-section-title

Query the user about the title of the section. Modifies **LaTeX-title**.

LaTeX-section-toc

Query the user for the toc entry. Modifies **LaTeX-toc**.

LaTeX-section-section

Insert \LaTeX section command according to **LaTeX-name**, **LaTeX-title**, and **LaTeX-toc**. If **LaTeX-toc** is **nil**, no toc entry is inserted. If **LaTeX-toc** or **LaTeX-title** are empty strings, **LaTeX-done-mark** will be placed at the point they should be inserted.

LaTeX-section-label

Insert a label after the section command. Controlled by the variable **LaTeX-section-label**.

To get a full featured **LaTeX-section** command, insert

```
(setq LaTeX-section-hook
  '(LaTeX-section-heading
    LaTeX-section-title
    LaTeX-section-toc
    LaTeX-section-section
    LaTeX-section-label))
```

in your init file such as **init.el** or **.emacs**.

The behavior of **LaTeX-section-label** is determined by the variable **LaTeX-section-label**.

LaTeX-section-label

[User Option]

Default prefix when asking for a label.

If it is a string, it is used unchanged for all kinds of sections. If it is **nil**, no label is inserted. If it is a list, the list is searched for a member whose car is equal to the name of the sectioning command being inserted. The cdr is then used as the prefix. If the name is not found, or if the cdr is **nil**, no label is inserted.

By default, chapters have a prefix of **'cha:'** while sections and subsections have a prefix of **'sec:'**. Labels are not automatically inserted for other types of sections.

2.4 Inserting Environment Templates

A large apparatus is available that supports insertions of environments, that is **'\begin{'** — **'\end{'** pairs.

AUCTEX is aware of most of the actual environments available in a specific document. This is achieved by examining your **'\documentclass'** command, and consulting a precompiled list of environments available in a large number of styles.

Most of these are described further in the following sections, and you may easily specify more. See Section 2.4.5 [Customizing Environments], page 22.

You insert an environment with **C-c C-e**, and select an environment type. Depending on the environment, AUCTEX may ask more questions about the optional parts of the selected environment type. With **C-u C-c C-e** you will change the current environment.

LaTeX-environment *arg* [Command]

(*C-c C-e*) AUCTeX will prompt you for an environment to insert. At this prompt, you may press **TAB** or **SPC** to complete a partially written name, and/or to get a list of available environments. After selection of a specific environment AUCTeX may prompt you for further specifications.

If the optional argument *arg* is non-**nil** (i.e. you have given a prefix argument), the current environment is modified and no new environment is inserted.

AUCTeX helps you adding labels to environments which use them, such as ‘**equation**’, ‘**figure**’, ‘**table**’, etc. . . . When you insert one of the supported environments with *C-c C-e*, you will be automatically prompted for a label. You can select the prefix to be used for such environments with the **LaTeX-label-alist** variable.

LaTeX-label-alist [User Option]

List the prefixes to be used for the label of each supported environment.

This is an alist whose car is the environment name, and the cdr either the prefix or a symbol referring to one.

If the name is not found, or if the cdr is **nil**, no label is automatically inserted for that environment.

If you want to automatically insert a label for a environment but with an empty prefix, use the empty string "" as the cdr of the corresponding entry.

As a default selection, AUCTeX will suggest the environment last inserted or, as the first choice the value of the variable **LaTeX-default-environment**.

LaTeX-default-environment [User Option]

Default environment to insert when invoking **LaTeX-environment** first time. When the current environment is ‘**document**’, it is overridden by **LaTeX-default-document-environment**.

LaTeX-default-document-environment [Variable]

Default environment when invoking ‘**LaTeX-environment**’ and the current environment is ‘**document**’. It is intended to be used in L^AT_EX class style files. For example, in **beamer.el** it is set to **frame**, in **letter.el** to **letter**, and in **slides.el** to **slide**.

If the document is empty, or the cursor is placed at the top of the document, AUCTeX will default to insert a ‘**document**’ environment prompting also for the insertion of ‘**\documentclass**’ and ‘**\usepackage**’ macros. You will be prompted for a new package until you enter nothing. If you do not want to insert any ‘**\usepackage**’ at all, just press **RET** at the first ‘**Packages**’ prompt.

AUCTeX distinguishes normal and expert environments. By default, it will offer completion only for normal environments. This behavior is controlled by the user option **TeX-complete-expert-commands**.

TeX-complete-expert-commands [User Option]

Complete macros and environments marked as expert commands.

Possible values are **nil**, **t**, or a list of style names.

nil Don’t complete expert commands (default).

t Always complete expert commands.
(styles ...)
 Only complete expert commands of *styles*.

You can close the current environment with `C-c]`, but we suggest that you use `C-c C-e` to insert complete environments instead.

LaTeX-close-environment [Command]
(C-c]) Insert an ‘`\end`’ that matches the current environment. When called with prefix argument *(C-u)*, reopen environment afterwards.

AUCTEX offers keyboard shortcuts for moving point to the beginning and to the end of the current environment.

LaTeX-find-matching-begin [Command]
(C-M-a) Move point to the ‘`\begin`’ of the current environment.

If this command is called inside a comment and `LaTeX-syntactic-comments` is enabled, try to find the environment in commented regions with the same comment prefix.

The key bind `C-M-a` actually calls `beginning-of-defun`, which in turn calls `LaTeX-find-matching-begin`.

LaTeX-find-matching-end [Command]
(C-M-e) Move point to the ‘`\end`’ of the current environment.

If this command is called inside a comment and `LaTeX-syntactic-comments` is enabled, try to find the environment in commented regions with the same comment prefix.

The key bind `C-M-e` actually calls `end-of-defun`, which in turn calls `LaTeX-find-matching-end`.

2.4.1 Equations

When inserting equation-like environments, the ‘`\label`’ will have a default prefix, which is controlled by the following variables:

LaTeX-equation-label [User Option]
 Prefix to use for ‘equation’ labels.

LaTeX-eqnarray-label [User Option]
 Prefix to use for ‘eqnarray’ labels.

LaTeX-amsmath-label [User Option]
 Prefix to use for amsmath equation labels. Amsmath equations include ‘`align`’, ‘`alignat`’, ‘`xalignat`’, ‘`multline`’, ‘`flalign`’ and ‘`gather`’.

2.4.2 Floats

Figures and tables (i.e., floats) may also be inserted using AUCTEX. After choosing either ‘figure’ or ‘table’ in the environment list described above, you will be prompted for a number of additional things.

float position

This is the optional argument of float environments that controls how they are placed in the final document. In L^AT_EX this is a sequence of the letters ‘htbp’ as described in the L^AT_EX manual. The value will default to the value of `LaTeX-float`.

caption This is the caption of the float. The default is to insert the caption at the bottom of the float. You can specify floats where the caption should be placed at the top with `LaTeX-top-caption-list`.

short caption

If the specified caption is greater than a specific length, then a short caption is prompted for and it is inserted as an optional argument to the ‘`\caption`’ macro. The length that a caption needs to be before prompting for a short version is controlled by `LaTeX-short-caption-prompt-length`.

label The label of this float. The label will have a default prefix, which is controlled by the variables `LaTeX-figure-label` and `LaTeX-table-label`.

Moreover, you will be asked if you want the contents of the float environment to be horizontally centered. Upon a positive answer a ‘`\centering`’ macro will be inserted at the beginning of the float environment.

LaTeX-float [User Option]
Default placement for floats.

LaTeX-figure-label [User Option]
Prefix to use for figure labels.

LaTeX-table-label [User Option]
Prefix to use for table labels.

LaTeX-top-caption-list [User Option]
List of float environments with top caption.

LaTeX-short-caption-prompt-length [User Option]
Number of chars a caption should be before prompting for a short caption.

It is also possible to have a caption and label in non-floating listing environments like in ‘`lstlisting`’ environment provided by the ‘`listings`’ package, or user defined listing floats defined with the ‘`newfloat`’ package. Such labels are prefixed with the value of `LaTeX-listing-label`.

LaTeX-listing-label [User Option]
Prefix to use for listing labels.

2.4.3 Itemize-like Environments

In an itemize-like environment, nodes (i.e., ‘\item’s) may be inserted using *C-c* LFD or *M-RET*. The latter is only defined as an alias if the key binding is still available.

LaTeX-insert-item [Command]
(C-c LFD or *M-RET)* Close the current item, move to the next line and insert an appropriate ‘\item’ for the current environment. That is, ‘itemize’ and ‘enumerate’ will have ‘\item ’ inserted, while ‘description’ will have ‘\item[] ’ inserted.

TeX-arg-item-label-p [User Option]
 If non-*nil*, you will always be asked for optional label in items. Otherwise, you will be asked only in description environments.

2.4.4 Tabular-like Environments

When inserting Tabular-like environments, that is, ‘tabular’ ‘array’ etc., you will be prompted for a template for that environment. Related variables:

LaTeX-default-format [User Option]
 Default format string for array and tabular environments.

LaTeX-default-width [User Option]
 Default width for minipage and tabular* environments.

LaTeX-default-position [User Option]
 Default position string for array and tabular environments. If *nil*, act like the empty string is given, but don’t prompt for a position.

AUCTEX calculates the number of columns from the format string and inserts the suitable number of ampersands.

You can use *C-c* LFD or *M-RET* (**LaTeX-insert-item**) to terminate rows in these environments. It supplies line break macro ‘\\’ and inserts the suitable number of ampersands on the next line. AUCTEX also supports the ‘*{num}{cols}’ notation (which may contain another ‘*-expression’) in the format string when calculating the number of ampersands. Please note that ‘num’ and ‘cols’ must be enclosed in braces; expressions like ‘*21’ are not recognized correctly by the algorithm.

LaTeX-insert-item [Command]
(C-c LFD or *M-RET)* Close the current row with ‘\\’, move to the next line and insert an appropriate number of ampersands for the current environment.

Similar supports are provided for various amsmath environments such as ‘align’, ‘gather’, ‘alignat’, ‘matrix’ etc. Try typing *C-c* LFD or *M-RET* in these environments. It recognizes the current environment and does the appropriate job depending on the context.

2.4.5 Customizing Environments

See Section 5.6.3 [Adding Environments], page 85, for how to customize the list of known environments.

2.5 Entering Mathematics

\TeX is written by a mathematician, and has always contained good support for formatting mathematical text. \AUCTeX supports this tradition, by offering a special minor mode for entering text with many mathematical symbols. You can enter this mode by typing `C-c ~`.

LaTeX-math-mode [Command]

(`C-c ~`) Toggle \LaTeX Math mode. This is a minor mode rebinding the key `LaTeX-math-abbrev-prefix` to allow easy typing of mathematical symbols. ``` will read a character from the keyboard, and insert the symbol as specified in `LaTeX-math-default` and `LaTeX-math-list`. If given a prefix argument, the symbol will be surrounded by dollar signs.

You can use another prefix key (instead of ```) by setting the variable `LaTeX-math-abbrev-prefix`.

To enable \LaTeX Math mode by default, add the following in your init file such as `init.el` or `.emacs`:

```
(add-hook 'LaTeX-mode-hook #'LaTeX-math-mode)
```

LaTeX-math-abbrev-prefix [User Option]

A string containing the prefix of `LaTeX-math-mode` commands; This value defaults to ```.

The string has to be a key or key sequence in a format understood by the `kbd` macro. This corresponds to the syntax usually used in the manuals for Emacs Lisp.

The variable `LaTeX-math-list` allows you to add your own mappings.

LaTeX-math-list [User Option]

A list containing user-defined keys and commands to be used in \LaTeX Math mode. Each entry should be a list of two to four elements.

First, the key to be used after `LaTeX-math-abbrev-prefix` for macro insertion. The key can be a character (e.g. `'?o'`) for a single stroke or a string (e.g. `"o a"`) for a multi-stroke binding. If it is `nil`, the symbol has no associated keystroke (it is available in the menu, though).

Second, a string representing the name of the macro (without a leading backslash.)

Third, a string representing the name of a submenu the command should be added to. Use a list of strings in case of nested menus.

Fourth, the position of a Unicode character to be displayed in the menu alongside the macro name. This is an integer value.

LaTeX-math-menu-unicode [User Option]

Whether the \LaTeX Math menu should try using Unicode for effect. Your Emacs built must be able to display include Unicode characters in menus for this feature.

\AUCTeX 's reference card `tex-ref.tex` includes a list of all math mode commands.

\AUCTeX can help you write subscripts and superscripts in math constructs by automatically inserting a pair of braces after typing `_` or `^` respectively and putting point between the braces. In order to enable this feature, set the variable `TeX-electric-sub-and-superscript` to a non-`nil` value.

TeX-electric-sub-and-superscript [User Option]

If non-nil, insert braces after typing `^` and `_` in math mode.

You can automatically turn off input methods, used to input non-ascii characters, when you begin to enter math constructs.

TeX-math-input-method-off-regexp [User Option]

Input method matching this regular expression is turned off when `$` is typed to begin math mode or a math environment is inserted by `C-c C-e` (`LaTeX-environment`).

Modifying Math Delimiters and Environments

AUCTEX offers the command `LaTeX-modify-math` to convert the mathematical construct at point—whether it is inline math such as `$...$` or `\(...\)`, a display construct such as `$$...$$` or `\[...\]`, or an environment such as `\begin{equation} ... \end{equation}`—into a different kind of delimiter or environment.

LaTeX-modify-math [Command]

Interactively, prompt for the target delimiter or environment. The completion table contains the inline delimiters `$` and `\(`, the display delimiters `$$` and `\[`, and every math environment known to `texmathp`, such as `equation`, `align*`, or anything in the user option `texmathp-tex-commands`. The current construct is then rewritten using the chosen form, taking care to

- keep any trailing punctuation outside inline math,
- put display constructs on their own lines, and
- strip any `\label{}` commands when converting to inline math.

When called from Lisp, *new-type* may be a string naming a delimiter or environment, or a cons `((open . close) . inline)` specifying custom delimiters, where *inline* is non-nil for inline math.

This command does *not* understand macro-based math wrappers such as `\ensuremath`. It may also fail in `docTeX` buffers.

A related command, invoked with a prefix argument, is `C-u C-c C-e` (`LaTeX-environment`) (see Section 2.4 [Environments], page 18). This modifies the current `LaTeX` environment, while `LaTeX-modify-math` also handles inline/display constructs.

You can define commands that convert to a particular form, e.g. by adding the following to your init file::

```
(defun my-LaTeX-make-brackets ()
  "Convert math construct at point to \"\\=\\[\\.\\.\\]\"."
  (interactive)
  (LaTeX-modify-math "\\[")
  (defun my-LaTeX-make-equation* ()
    "Convert math construct at point to \"equation*\"."
    (interactive)
    (LaTeX-modify-math "equation*"))
```

You can use `LaTeX-modify-math` to build higherlevel toggles. The following modifies any math construct to an `equation*` environment, then toggles the numbered status:

```
(defun my-LaTeX-toggle-numbered ()
```

```

"Convert math construct at point to \"equation*\".
If the math construct is already \"equation*\", then toggle with the
numbered variant \"equation\"."
(interactive)
(unless (texmathp) (user-error "Not inside math"))
(let ((current (car texmathp-why)))
  (LaTeX-modify-math
   (pcase current
     ("equation*" "equation")
     ("equation" "equation*")
     (_ "equation*")))))

```

A further example toggles between ‘equation’, ‘align’ and their starred forms:

```

(defun my-LaTeX-toggle-align ()
  "Toggle math environment at point between \"equation\" and \"align\"."
  (interactive)
  (unless (texmathp) (user-error "Not inside math"))
  (let ((current (car texmathp-why)))
    (LaTeX-modify-math
     (pcase current
       ("align*" "equation*")
       ("equation*" "align*")
       ("align" "equation")
       ("equation" "align")
       (_ "align*")))))

```

Such helper commands can be bound in `LaTeX-mode-map` as you see fit, e.g. by adding the following to your init file:

```
(keymap-set LaTeX-mode-map "C-c e" #'my-LaTeX-make-equation*)
```

See Section 2.1 [LaTeX-make-inline], page 12, for a built-in convenience wrapper that converts display constructs to inline math.

2.6 Completion

Emacs lisp programmers probably know the `lisp-complete-symbol` command which was bound to *M-TAB* until completion-at-point became the new standard completion facility (see below). Users of the wonderful `ispell` mode know and love the `ispell-complete-word` command from that package. Similarly, AUCTEX has a `TeX-complete-symbol` command, by default bound to *M-TAB* which is equivalent to *C-M-i*. Using `TeX-complete-symbol` makes it easier to type and remember the names of long L^AT_EX macros.

In order to use `TeX-complete-symbol`, you should write a backslash and the start of the macro. Typing *M-TAB* will now complete as much of the macro, as it unambiguously can. For example, if you type “`\renewc`” and then *M-TAB*, it will expand to “`\renewcommand`”. But there’s more: if point is just after ‘`\begin{`’, then `TeX-complete-symbol` will complete L^AT_EX environments, etc. This is controlled by `TeX-complete-list`.

TeX-complete-symbol

[Command]

(*M-TAB*) Complete T_EX symbol before point.

TeX-complete-list [Variable]

List of ways to complete the preceding text.

Each entry is a list with the following elements:

1. Regexp matching the preceding text or a predicate of arity 0 which returns `non-nil` and sets ‘match-data’ appropriately if it is applicable.
2. A number indicating the subgroup in the regexp containing the text.
3. A function returning an alist of possible completions.
4. Text to append after a successful completion.

Or alternatively:

1. Regexp matching the preceding text.
2. Function to do the actual completion.

More recent Emacs versions have a new completion mechanism. Modes may define and register custom `completion-at-point` functions and when the user invokes `completion-at-point` (usually bound to `M-TAB`), all such registered functions are consulted for checking for possible completions. Modern completion UIs like *company-mode* or *corfu* support this completion-at-point facility.

TeX--completion-at-point [Function]

AUCTEX’s completion-at-point function which is automatically added to `completion-at-point-functions` in `TeX` and `LaTeX` buffers.

It offers the same completion candidates as would `TeX-complete-symbol` (and is also controlled by `TeX-complete-list`) except that it doesn’t fall back on `ispell-complete-word` which would be awkward with completion UIs like *company-mode*.

LaTeX--arguments-completion-at-point [Function]

AUCTEX’s completion-at-point function inside arguments which is automatically added to `completion-at-point-functions` in `LaTeX` buffers.

It offers the completion candidates stored in the variables `TeX-symbol-list` and `LaTeX-environment-list` for single candidate, multiple candidates separated by commas, or key-value candidates separated by commas and/or equal signs.

Sometimes the list of offered candidates is enriched by annotations which are appended to the candidates themselves. For labels which are referenced, the annotations are controlled by the variable `LaTeX-label-annotation-max-length` and `RefTeX` being enabled in the buffer since the annotations are gathered from it.

LaTeX-label-annotation-max-length [User Option]

Controls the length of the annotation attached to a label, default is 30. Setting this variable to 0 disables annotation of labels.

A more direct way to insert a macro is with `TeX-insert-macro`, bound to `C-c C-m` which is equivalent to `C-c RET`. It has the advantage over completion that it knows about the argument of most standard `LaTeX` macros, and will prompt for them. It also knows about the type of the arguments, so it will for example give completion for the argument to ‘`\include`’. Some examples are listed below.

TeX-insert-macro [Command]
 (*C-c C-m* or *C-c RET*) Prompt (with completion) for the name of a \TeX macro, and if AUCT \TeX knows the macro, prompt for each argument.

As a default selection, AUCT \TeX will suggest the macro last inserted or, as the first choice the value of the variable `TeX-default-macro`.

TeX-default-macro [User Option]
 Default macro to insert when invoking `TeX-insert-macro` first time.

TeX-insert-macro-default-style [User Option]
 Specifies whether `TeX-insert-macro` will ask for all optional arguments.
 If set to the symbol `show-optional-args`, `TeX-insert-macro` asks for optional arguments of \TeX macros, unless the previous optional argument has been rejected. If set to `show-all-optional-args`, `TeX-insert-macro` asks for all optional arguments. `mandatory-args-only`, `TeX-insert-macro` asks only for mandatory arguments. When `TeX-insert-macro` is called with prefix argument (*C-u*), it's the other way round.

Note that for some macros, there are special mechanisms, e.g. `TeX-arg-cite-note-p` and `LaTeX-includegraphics-options-alist`.

A faster alternative is to enable the option `TeX-electric-escape`.

TeX-electric-escape [User Option]
 If this is non-`nil`, typing the \TeX escape character `\` will invoke the command `TeX-electric-macro`.
 In Texinfo mode, the command is invoked by `@` instead.

The difference between `TeX-insert-macro` and `TeX-electric-macro` is that space key `SPC` will complete and exit from the minibuffer in `TeX-electric-macro`. Use `TAB` if you merely want to complete.

TeX-electric-macro [Command]
 Prompt (with completion) for the name of a \TeX macro, and if AUCT \TeX knows the macro, prompt for each argument. Space (`SPC`) will complete and exit.

By default AUCT \TeX will put an empty set braces `{}` after a macro with no arguments to stop it from eating the next whitespace. This is suppressed inside math mode and can be disabled totally by setting `TeX-insert-braces` to `nil`.

TeX-insert-braces [User Option]
 If non-`nil`, append a empty pair of braces after inserting a macro with no arguments.

TeX-insert-braces-alist [User Option]
 Control the insertion of a pair of braces after a macro on a per macro basis.
 This variable is an alist. Each element is a cons cell, whose car is the macro name, and the cdr is non-`nil` or `nil`, depending on whether a pair of braces should be, respectively, appended or not to the macro.

If a macro has an element in this variable, AUCT \TeX will use its value to decide what to do, whatever the value of the variable `TeX-insert-braces`.

Completions work because AUCTeX can analyze T_EX files, and store symbols in Emacs Lisp files for later retrieval. See Section 5.5 [Automatic], page 77, for more information.

AUCTeX distinguishes normal and expert macros. By default, it will offer completion only for normal commands. This behavior can be controlled using the user option `TeX-complete-expert-commands`.

TeX-complete-expert-commands [User Option]

Complete macros and environments marked as expert commands.

Possible values are `nil`, `t`, or a list of style names.

`nil` Don't complete expert commands (default).

`t` Always complete expert commands.

`(styles ...)`
Only complete expert commands of *styles*.

AUCTeX will also make completion for many macro arguments, for example existing labels when you enter a `\ref` macro with `TeX-insert-macro` or `TeX-electric-macro`, and BibT_EX entries when you enter a `\cite` macro. For this kind of completion to work, parsing must be enabled as described in Section 5.3 [Parsing Files], page 70. For `\cite` you must also make sure that the BibT_EX files have been saved at least once after you enabled automatic parsing on save, and that the basename of the BibT_EX file does not conflict with the basename of one of T_EX files.

2.7 Marking Environments, Sections, or Texinfo Nodes

You can mark the current environment by typing `C-c .`, or the current section by typing `C-c *`.

In Texinfo documents you can type `C-M-h` to mark the current node.

When the region is set, the point is moved to its beginning and the mark to its end.

2.7.1 L^AT_EX Commands for Marking Environments and Sections

LaTeX-mark-section [Command]

`(C-c *)` Set mark at end of current logical section, and point at top.

With a non-`nil` prefix argument, mark only the region from the current section start to the next sectioning command. Thereby subsections are not being marked. Otherwise, any included subsections are also marked along with current section.

LaTeX-mark-environment [Command]

`(C-c .)` Set mark to the end of the current environment and point to the matching beginning.

If a prefix argument is given, mark the respective number of enclosing environments. The command will not work properly if there are unbalanced begin-end pairs in comments and verbatim environments.

2.7.2 Texinfo Commands for Marking Environments and Sections

Texinfo-mark-section [Command]

(*C-c **) Mark the current section, with inclusion of any containing node.

The current section is detected as starting by any of the structuring commands matched by the regular expression in the variable `outline-regexp` which in turn is a regular expression matching any element of the variable `texinfo-section-list`.

With a non-nil prefix argument, mark only the region from the current section start to the next sectioning command. Thereby subsections are not being marked. Otherwise, any included subsections are also marked.

Note that when the current section is starting immediately after a node command, then the node command is also marked as part of the section.

Texinfo-mark-environment [Command]

(*C-c .*) Set mark to the end of the current environment and point to the matching beginning.

If a prefix argument is given, mark the respective number of enclosing environments. The command will not work properly if there are unbalanced begin-end pairs in comments and verbatim environments.

Texinfo-mark-node [Command]

(*C-M-h*) Mark the current node. This is the node in which point is located. It is starting at the previous occurrence of the keyword `@node` and ending at next occurrence of the keywords `@node` or `@bye`.

2.8 Commenting

It is often necessary to comment out temporarily a region of \TeX or \LaTeX code. This can be done with the commands *C-c ;* and *C-c %*. *C-c ;* will comment out all lines in the current region, while *C-c %* will comment out the current paragraph. Type *C-c ;* again to uncomment all lines of a commented region, or *C-c %* again to uncomment all comment lines around point. These commands will insert or remove a single ‘%’ respectively.

TeX-comment-or-uncomment-region [Command]

(*C-c ;*) Add or remove ‘%’ from the beginning of each line in the current region. Uncommenting works only if the region encloses solely commented lines. If \AUCTEX should not try to guess if the region should be commented or uncommented the commands `TeX-comment-region` and `TeX-uncomment-region` can be used to explicitly comment or uncomment the region in concern.

TeX-comment-or-uncomment-paragraph [Command]

(*C-c %*) Add or remove ‘%’ from the beginning of each line in the current paragraph. When removing ‘%’ characters the paragraph is considered to consist of all preceding and succeeding lines starting with a ‘%’, until the first non-comment line.

In \docTeX document, all documentations are commented out. \AUCTEX inserts ‘%’ (with an accompanying space) at the beginning of line when you issue some commands including sectioning (*C-c C-s*) and inserting environments (*C-c C-e*), on a commented line. This behavior is controlled by `LaTeX-insert-into-comments`.

LaTeX-insert-into-comments

[User Option]

When this option is non-`nil`, some editing commands are aware of comment prefix at the beginning of line and insert it in the line created anew.

The default value of this option is `nil` and is set to `t` in `docTeX` mode.

2.9 Indenting

Indentation means the addition of whitespace at the beginning of lines to reflect special syntactical constructs. This makes it easier to see the structure of the document, and to catch errors such as a missing closing brace. Thus, the indentation is done for precisely the same reasons that you would indent ordinary computer programs.

Indentation is done by `LATEX` environments and by `TEX` groups, that is the body of an environment is indented by the value of `LaTeX-indent-level` (default 2). Also, items of an ‘itemize-like’ environment are indented by the value of `LaTeX-item-indent`, default -2. (Items are identified with the help of `LaTeX-item-regexp`.) If more environments are nested, they are indented ‘accumulated’ just like most programming languages usually are seen indented in nested constructs.

You can explicitly indent single lines, usually by pressing `TAB`, or marked regions by calling `indent-region` on it. If you have `auto-fill-mode` enabled and a line is broken while you type it, Emacs automatically cares about the indentation in the following line. If you want to have a similar behavior upon typing `RET`, you can customize the variable `TeX-newline-function` and change the default of `newline` which does no indentation to `newline-and-indent` which indents the new line or `reindent-then-newline-and-indent` which indents both the current and the new line.

There are certain `LATEX` environments which should be indented in a special way, like ‘`tabular`’ or ‘`verbatim`’. Those environments may be specified in the variable `LaTeX-indent-environment-list` together with their special indentation functions. Taking the ‘`verbatim`’ environment as an example you can see that `current-indentation` is used as the indentation function. This will stop `AUCTEX` from doing any indentation in the environment if you hit `TAB` for example.

There are environments in `LaTeX-indent-environment-list` which do not bring a special indentation function with them. This is due to the fact that first the respective functions are not implemented yet and second that filling will be disabled for the specified environments. This shall prevent the source code from being messed up by accidentally filling those environments with the standard filling routine. If you think that providing special filling routines for such environments would be an appropriate and challenging task for you, you are invited to contribute. (See Section 2.10 [Filling], page 33, for further information about the filling functionality.)

The check for the indentation function may be enabled or disabled by customizing the variable `LaTeX-indent-environment-check`.

For tabular-like environments, `AUCTEX` has a built-in function to indent according to preceding ‘&’ signs and assigns it to all known tabular-like environments in the default value of `LaTeX-indent-environment-list`.

As a side note with regard to formatting special environments: Newer Emacsen include `align.el` and therefore provide some support for formatting ‘`tabular`’ and ‘`tabbing`’ en-

vironments with the function `align-current` which will nicely align columns in the source code.

AUCTEX is able to format commented parts of your code just as any other part. This means L^AT_EX environments and T_EX groups in comments will be indented syntactically correct if the variable `LaTeX-syntactic-comments` is set to `t`. If you disable it, comments will be filled like normal text and no syntactic indentation will be done.

Following you will find a list of most commands and variables related to indenting with a small summary in each case:

TAB	<code>LaTeX-indent-line</code> will indent the current line.	
LFD		
C-j	<code>newline-and-indent</code> inserts a new line (much like RET) and moves the cursor to an appropriate position by the left margin. Most keyboards nowadays lack a linefeed key and C-j may be tedious to type. Therefore you can customize AUCTEX to perform indentation upon typing RET as well. The respective option is called <code>TeX-newline-function</code> .	
LaTeX-indent-environment-list		[User Option]
	List of environments with special indentation. The second element in each entry is the function to calculate the indentation level in columns.	
LaTeX-indent-level		[User Option]
	Number of spaces to add to the indentation for each ‘\begin’ not matched by a ‘\end’.	
LaTeX-item-indent		[User Option]
	Number of spaces to add to the indentation for ‘\item’'s in list environments.	
TeX-brace-indent-level		[User Option]
	Number of spaces to add to the indentation for each ‘{’ not matched by a ‘}’.	
LaTeX-syntactic-comments		[User Option]
	If non-nil comments will be filled and indented according to L ^A T _E X syntax. Otherwise they will be filled like normal text.	
TeX-newline-function		[User Option]
	Used to specify the function which is called when RET is pressed. This will normally be <code>newline</code> which simply inserts a new line. In case you want to have AUCTEX do indentation as well when you press RET, use the built-in functions <code>newline-and-indent</code> or <code>reindent-then-newline-and-indent</code> . The former inserts a new line and indents the following line, i.e. it moves the cursor to the right position and therefore acts as if you pressed LFD. The latter function additionally indents the current line. If you choose ‘Other’, you can specify your own fancy function to be called when RET is pressed.	

AUCTEX treats by default ‘\[…\]’ math mode as a regular environment and indents it accordingly. If you do not like such behavior you only need to remove ‘\[…\]’ from `LaTeX-begin-regexp` and `LaTeX-end-regexp` variables respectively.

A closely related topic is indenting of text enclosed in square brackets, parentheses and other pairs. AUCTEX offers two variables which control if indentation happens inside these pairs.

TeX-indent-open-delimiters [User Option]

This variable contains additional opening delimiters which increase indentation. For example add `[` to this variable to get text after a square bracket indented.

TeX-indent-close-delimiters [User Option]

This is the accompanying variable to **TeX-indent-open-delimiters** decreasing the indentation again. This variable should contain `]` if **TeX-indent-open-delimiters** is set like described above.

Note that this is an opt-in feature, both variables are initially set to an empty string. That is because it introduces non-trivial side effects to include `[` and `]` in **TeX-indent-open-delimiters** and **TeX-indent-close-delimiters**; if you only have an opening square bracket in your text without closing it, wrong indentation persists in the following text. For example, in math expression, half-open intervals are frequently written as `'[0,10)'` or `'[0,10['`. In such cases, you can put the closing part as a comment in the same line in order to have correct indentation after that:

```
$[0,10)$ % ]
$[0,10[$ % ]]
```

Another example is `'\left'-' \right'` pair in equations. Similar workarounds are available:

```
\begin{equation}
  \left[ % ]
    xyz
  \right] % [
  abc
\end{equation}
```

You can include parens `'()'` also in **TeX-indent-open-delimiters** and **TeX-indent-close-delimiters** to enable indent inside them. Be prepared for similar side effects when you do.

Note that commented curly braces `{` and `}` aren't counted when AUCTeX computes indentation.

In docTeX-mode, TeX code is enclosed in `'macrocode'` environment like this:

```
% \begin{macrocode}
\def\foo#1{%
  $#1$%
}
% \end{macrocode}
```

Sometimes, the code is long and one wants to insert comments inside the TeX code like this:

```
% \begin{macrocode}
\def\foo#1{%
% \end{macrocode}
% Comment the next line of code
% \begin{macrocode}
  $#1$%
```

```
}
%    \end{macrocode}
```

Usually, the comment inside the code interrupts the indentation. This behavior can be controlled by setting the variable `docTeX-indent-across-comments`.

docTeX-indent-across-comments [User Option]
If non-`nil`, indentation in `docTeX` is done across comments. This option is disabled by default.

2.10 Filling

Filling deals with the insertion of line breaks to prevent lines from becoming wider than what is specified in `fill-column`. The linebreaks will be inserted automatically if `auto-fill-mode` is enabled. In this case the source is not only filled but also indented automatically as you write it.

`auto-fill-mode` can be enabled for `AUCTeX` by calling `turn-on-auto-fill` in one of the hooks `AUCTeX` is running. See Section 5.1 [Modes and Hooks], page 68. As an example, if you want to enable `auto-fill-mode` in `LaTeX-mode`, put the following into your init file:

```
(add-hook 'LaTeX-mode-hook #'turn-on-auto-fill)
```

You can manually fill explicitly marked regions, paragraphs, environments, complete sections, or the whole buffer. (Note that manual filling in `AUCTeX` will indent the start of the region to be filled in contrast to many other Emacs modes.)

There are some syntactical constructs which are handled specially with regard to filling. These are so-called *code comments* and *paragraph commands*.

Code comments are comments preceded by code or text in the same line. Upon filling a region, code comments themselves will not get filled. Filling is done from the start of the region to the line with the code comment and continues after it. In order to prevent overfull lines in the source code, a linebreak will be inserted before the last non-comment word by default. This can be changed by customizing `LaTeX-fill-break-before-code-comments`. If you have overfull lines with code comments you can fill those explicitly by calling `LaTeX-fill-paragraph` or pressing `M-q` with the cursor positioned on them. This will add linebreaks in the comment and indent subsequent comment lines to the column of the comment in the first line of the code comment. In this special case `M-q` only acts on the current line and not on the whole paragraph.

Lines with `\par` are treated similarly to code comments, i.e. `\par` will be treated as paragraph boundary which should not be followed by other code or text. But it is not treated as a real paragraph boundary like an empty line where filling a paragraph would stop.

Paragraph commands like `\section` or `\noindent` (the list of commands is defined by `LaTeX-paragraph-commands`) are often to be placed in their own line(s). This means they should not be consecuted with any preceding or following adjacent lines of text. `AUCTeX` will prevent this from happening if you do not put any text except another macro after the end of the last brace of the respective macro. If there is other text after the macro, `AUCTeX` regards this as a sign that the macro is part of the following paragraph.

Here are some examples:

```
\begin{quote}
```

```

text text text text
\begin{quote}\label{foo}
text text text text

```

If you press *M-q* on the first line in both examples, nothing will change. But if you write

```

\begin{quote} text
text text text text

```

and press *M-q*, you will get

```

\begin{quote} text text text text text

```

Besides code comments and paragraph commands, another speciality of filling in AUCTeX involves commented lines. You should be aware that these comments are treated as islands in the rest of the L^AT_EX code if syntactic filling is enabled. This means, for example, if you try to fill an environment with `LaTeX-fill-environment` and have the cursor placed on a commented line which does not have a surrounding environment inside the comment, AUCTeX will report an error.

The relevant commands and variables with regard to filling are:

C-c C-q C-p

`LaTeX-fill-paragraph` will fill and indent the current paragraph.

M-q

Alias for *C-c C-q C-p*

C-c C-q C-e

`LaTeX-fill-environment` will fill and indent the current environment. This may e.g. be the ‘document’ environment, in which case the entire document will be formatted.

C-c C-q C-s

`LaTeX-fill-section` will fill and indent the current logical sectional unit.

C-c C-q C-r

`LaTeX-fill-region` will fill and indent the current region.

LaTeX-fill-break-at-separators

[User Option]

List of separators before or after which respectively linebreaks will be inserted if they do not fit into one line. The separators can be curly braces, brackets, switches for inline math (`‘$’`, `‘\('`, `‘\)’`) and switches for display math (`‘\[’`, `‘\]’`). Such formatting can be useful to make macros and math more visible or to prevent overfull lines in the L^AT_EX source in case a package for displaying formatted T_EX output inside the Emacs buffer, like `preview-latex`, is used.

LaTeX-fill-break-before-code-comments

[User Option]

Code comments are comments preceded by some other text in the same line. When a paragraph containing such a comment is to be filled, the comment start will be seen as a border after which no line breaks will be inserted in the same line. If the option `LaTeX-fill-break-before-code-comments` is enabled (which is the default) and the comment does not fit into the line, a line break will be inserted before the last non-comment word to minimize the chance that the line becomes overfull.

LaTeX-fill-excluded-macros

[User Option]

A list of macro names (without leading backslash) for whose arguments filling should be disabled. Typically, you will want to add macros here which have long, multi-line arguments. An example is `\pgfplotstabletypeset` from the `pgfplotstable` package which is used as shown in the following listing:

```
\pgfplotstabletypeset[skip first n=4]{%  
  XYZ Format,  
  Version 1.234  
  Date 2010-09-01  
  @author Mustermann  
  A B C  
  1 2 3  
  4 5 6  
}
```

3 Controlling Screen Display

It is often desirable to get visual help of what markup code in a text actually does without having to decipher it explicitly. For this purpose Emacs and AUCT_EX provide font locking (also known as syntax highlighting) which visually sets off markup code like macros or environments by using different colors or fonts. For example text to be typeset in italics can be displayed with an italic font in the editor as well, or labels and references get their own distinct color.

While font locking helps you grasp the purpose of markup code and separate markup from content, the markup code can still be distracting. AUCT_EX lets you hide those parts and show them again at request with its built-in support for hiding macros and environments which we call folding here.

Besides folding of macros and environments, AUCT_EX provides support for Emacs' outline mode which lets you narrow the buffer content to certain sections of your text by hiding the parts not belonging to these sections.

Moreover, you can focus in a specific portion of the code by narrowing the buffer to the desired region. AUCT_EX provides also functions to narrow the buffer to the current group and to L^AT_EX environments.

AUCT_EX also provides some WYSIWYG features.

First, you can customize `font-latex-fontify-script` to enable special formatting of \sim superscripts and $_$ subscripts (see Section 3.1 [Font Locking], page 36).

Secondly, AUCT_EX with GNU Emacs 25 or later can display certain math macros using Unicode characters, e.g., `\alpha` as α . This is called prettification and is lightweight and reasonable robust (see Section 3.5 [Prettifying], page 49).

A more accurate approach is provided by `preview-latex`, a subsystem of AUCT_EX, see Section “Introduction” in *The preview-latex Manual*. This system uses L^AT_EX to generate images that are then displayed in your buffer. It is extremely accurate but can be fragile with some packages (like older pgf versions).

Please note that you can use prettification and `preview-latex` together.

3.1 Font Locking

Font locking is supposed to improve readability of the source code by highlighting certain keywords with different colors or fonts. It thereby lets you recognize the function of markup code to a certain extent without having to read the markup command. For general information on controlling font locking with Emacs' Font Lock mode, see Section “Font Lock Mode” in *GNU Emacs Manual*.

TeX-install-font-lock

[User Option]

Once font locking is enabled globally or for the major modes provided by AUCT_EX, the font locking patterns and functionality of `font-latex` are activated by default since this variable is initialized to `font-latex-setup`. You can switch to a different font locking scheme or disable font locking in AUCT_EX by customizing the variable `TeX-install-font-lock`.

Besides `font-latex` AUCT_EX ships with a scheme which is derived from Emacs' default L^AT_EX mode and activated by choosing `tex-font-setup`. Be aware that this scheme is

not coupled with AUCTeX's style system and not the focus of development. Therefore and due to font-latex being much more feature-rich the following explanations will only cover font-latex.

In case you want to hook in your own fontification scheme, you can choose **other** and insert the name of the function which sets up your font locking patterns. If you want to disable fontification in AUCTeX completely, choose **ignore**.

font-latex provides many options for customization which are accessible with *M-x customize-group RET font-latex RET*. For this description the various options are explained in conceptional groups.

3.1.1 Fontification of macros

Highlighting of macros can be customized by adapting keyword lists which can be found in the customization group **font-latex-keywords**.

Three types of macros can be handled differently with respect to fontification:

1. Commands of the form `'\foo[bar]{baz}'` which consist of the macro itself, optional arguments in square brackets and mandatory arguments in curly braces. For the command itself the face **font-lock-keyword-face** will be used and for the optional arguments the face **font-lock-variable-name-face**. The face applied to the mandatory argument depends on the macro class represented by the respective built-in variables.
2. Declaration macros of the form `'{\foo text}'` which consist of the macro which may be enclosed in a TeX group together with text to be affected by the macro. In case a TeX group is present, the macro will get the face **font-lock-keyword-face** and the text will get the face configured for the respective macro class. If no TeX group is present, the latter face will be applied to the macro itself.
3. Simple macros of the form `'\foo'` which do not have any arguments or groupings. The respective face will be applied to the macro itself.

Customization variables for `'\foo[bar]{baz}'` type macros allow both the macro name and the sequence of arguments to be specified. The latter is done with a string which can contain the characters

- '*' indicating the existence of a starred variant for the macro,
- '[' for optional arguments in brackets,
- '{' for mandatory arguments in braces,
- '\' for mandatory arguments consisting of a single macro and
- '|' as a prefix indicating that two alternatives are following.

For example the specifier for `'\documentclass'` would be `'[{'` because the macro has one optional followed by one mandatory argument. The specifier for `'\newcommand'` would be `'*|{\[[]{'` because there is a starred variant, the mandatory argument following the macro name can be a macro or a TeX group which can be followed by two optional arguments and the last token is a mandatory argument in braces.

Customization variables for the `'{\foo text}'` and `'\foo'` types are simple lists of strings where each entry is a macro name (without the leading backslash).

General macro classes

`font-latex` provides keyword lists for different macro classes which are described in the following table:

`font-latex-match-function-keywords`

Keywords for macros defining or related to functions, like ‘`\newcommand`’.

Type: ‘`\macro[...]{...}`’

Face: `font-lock-function-name-face`

`font-latex-match-reference-keywords`

Keywords for macros defining or related to references, like ‘`\ref`’.

Type: ‘`\macro[...]{...}`’

Face: `font-lock-constant-face`

`font-latex-match-textual-keywords`

Keywords for macros specifying textual content, like ‘`\caption`’.

Type: ‘`\macro[...]{...}`’

Face: `font-lock-type-face`

`font-latex-match-variable-keywords`

Keywords for macros defining or related to variables, like ‘`\setlength`’.

Type: ‘`\macro[...]{...}`’

Face: `font-lock-variable-name-face`

`font-latex-match-warning-keywords`

Keywords for important macros, e.g. affecting line or page break, like ‘`\clearpage`’.

Type: ‘`\macro`’

Face: `font-latex-warning-face`

Sectioning commands

Sectioning commands are macros like ‘`\chapter`’ or ‘`\section`’. For these commands there are two fontification schemes which may be selected by customizing the variable `font-latex-fontify-sectioning`.

`font-latex-fontify-sectioning`

[User Option]

Per default sectioning commands will be shown in a larger, proportional font, which corresponds to a number for this variable. The font size varies with the sectioning level, e.g. ‘`\part`’ (`font-latex-sectioning-0-face`) has a larger font than ‘`\paragraph`’ (`font-latex-sectioning-5-face`). Typically, values from 1.05 to 1.3 for `font-latex-fontify-sectioning` give best results, depending on your font setup. If you rather like to use the base font and a different color, set the variable to the symbol ‘`color`’. In this case the face `font-lock-type-face` will be used to fontify the argument of the sectioning commands.

You can make `font-latex` aware of your own sectioning commands by adding them to the keyword lists: `font-latex-match-sectioning-0-keywords` (`font-latex-sectioning-0-face`) ... `font-latex-match-sectioning-5-keywords` (`font-latex-sectioning-5-face`).

Related to sectioning there is special support for slide titles which may be fontified with the face `font-latex-slide-title-face`. You can add macros which should appear in this face by customizing the variable `font-latex-match-slide-title-keywords`.

Commands for changing fonts

L^AT_EX provides various macros for changing fonts or font attributes. For example, you can select an italic font with ‘`\textit{...}`’ or bold with ‘`\textbf{...}`’. An alternative way to specify these fonts is to use special macros in T_EX groups, like ‘`{\itshape ...}`’ for italics and ‘`{\bfseries ...}`’ for bold. As mentioned above, we call the former variants commands and the latter declarations.

Besides the macros for changing fonts provided by L^AT_EX there is an infinite number of other macros—either defined by yourself for logical markup or defined by macro packages—which affect the font in the typeset text. While L^AT_EX’s built-in macros and macros of packages known by AUC_TE_X are already handled by `font-latex`, different keyword lists per type style and macro type are provided for entering your own macros which are listed in the table below.

`font-latex-match-bold-command-keywords`

Keywords for commands specifying a bold type style.

Face: `font-latex-bold-face`

`font-latex-match-italic-command-keywords`

Keywords for commands specifying an italic font.

Face: `font-latex-italic-face`

`font-latex-match-underline-command-keywords`

Keywords for commands specifying an underlined text.

Face: `font-latex-underline-face`

`font-latex-match-math-command-keywords`

Keywords for commands specifying a math font.

Face: `font-latex-math-face`

`font-latex-match-type-command-keywords`

Keywords for commands specifying a typewriter font.

Face: `font-lock-type-face`

`font-latex-match-bold-declaration-keywords`

Keywords for declarations specifying a bold type style.

Face: `font-latex-bold-face`

`font-latex-match-italic-declaration-keywords`

Keywords for declarations specifying an italic font.

Face: `font-latex-italic-face`

`font-latex-match-type-declaration-keywords`

Keywords for declarations specifying a typewriter font.

Face: `font-latex-type-face`

Deactivating defaults of built-in keyword classes

`font-latex` ships with predefined lists of keywords for the classes described above. You can disable these defaults per class by customizing the variable `font-latex-deactivated-keyword-classes`. This is a list of strings for keyword classes to be deactivated. Valid entries are "warning", "variable", "biblatexnoarg", "biblatex", "reference", "function", "function-noarg", "sectioning-0", "sectioning-1", "sectioning-2", "sectioning-3", "sectioning-4", "sectioning-5", "slide-title", "textual", "bold-command", "italic-command", "underline-command", "math-command", "type-command", "bold-declaration", "italic-declaration" or "type-declaration".

You can also get rid of certain keywords only. For example if you want to remove highlighting of footnotes as references you can put the following stanza into your init file:

```
(with-eval-after-load 'font-latex
  (setq-default
    font-latex-match-reference-keywords-local
    (remove (assoc-string "footnote"
                        font-latex-match-reference-keywords-local)
            font-latex-match-reference-keywords-local)))
```

But note that this means fiddling with `font-latex`'s internals and is not guaranteed to work in future versions of `font-latex`.

User-defined keyword classes

In case the customization options explained above do not suffice for your needs, you can specify your own keyword classes by customizing the variable `font-latex-user-keyword-classes`.

font-latex-user-keyword-classes [User Option]

Every keyword class consists of four parts, a name, a list of keywords, a face and a specifier for the type of macros to be highlighted.

When adding new entries, you have to use unique values for the class names, i.e. they must not clash with names of the built-in keyword classes or other names given by you. Additionally the names must not contain spaces.

The list of keywords defines which commands and declarations should be covered by the keyword class. A keyword can either be a simple command name omitting the leading backslash or a list consisting of the command name and a string specifying the sequence of arguments for the command.

The face argument can either be an existing face or face attributes made by you.

There are three alternatives for the type of keywords—"Command with arguments", "Declaration inside `TEX` group" and "Command without arguments"—which correspond with the macro types explained above.

3.1.2 Fontification of quotes

Text in quotation marks is displayed with the face `font-latex-string-face`. Besides the various forms of opening and closing double and single quotation marks, so-called guillemets (`<<`, `>>`) can be used for quoting. Because there are two styles of using them—French style: `<< text >>`; German style: `>>text<<`—you can customize the variable `font-latex-quotes`

to tell font-latex which type you are using if the correct value cannot be derived from document properties.

font-latex-quotes [User Option]

The default value of **font-latex-quotes** is ‘auto’ which means that font-latex will try to derive the correct type of quotation mark matching from document properties like the language option supplied to the babel L^AT_EX package.

If the automatic detection fails for you and you mostly use one specific style you can set it to a specific language-dependent value as well. Set the value to ‘german’ if you are using >>German quotes<< and to ‘french’ if you are using << French quotes >>. font-latex will recognize the different ways these quotes can be given in your source code, i.e. (“<”, “>”), (<<, >>) and the respective 8-bit variants.

If you set **font-latex-quotes** to nil, quoted content will not be fontified.

3.1.3 Fontification of mathematical constructs

In L^AT_EX mathematics can be indicated by a variety of different methods: toggles (like dollar signs), macros and environments. Math constructs known by font-latex are displayed with the face **font-latex-math-face**. Support for dollar signs and shorthands like ‘\(...\)’ or ‘\[...\]’ is built-in and not customizable. Support for other math macros and environments can be adapted by customizing the variables **font-latex-match-math-command-keywords** and **texmathp-tex-commands** respectively. It is no longer recommended to customize **font-latex-math-environments**.

To convert your customization in **font-latex-math-environments** into **texmathp-tex-commands**, please register your own math environments, together with starred variants if any, as entries of **env-on** type in **texmathp-tex-commands**, then clear out **font-latex-math-environments**. You have to restart Emacs for this new customization to take effect for fontification.

In order to make math constructs more readable, font-latex displays subscript and superscript parts in a smaller font and raised or lowered respectively. This fontification feature can be controlled with the variables **font-latex-fontify-script** and **font-latex-script-display**.

font-latex-fontify-script [User Option]

If non-nil, fontify subscript and superscript strings. Concretely, this means that the scripts are raised or lowered.

Another possibility is setting this variable to the symbol **multi-level**. In this case, in a formula x^{y^z} , y is raised above and smaller than x , and z is raised above and smaller than y . With many script levels, the text might become too small to be readable. (See **font-latex-fontify-script-max-level** below.)

Lastly, you can set this variable to **invisible** whose behavior is like **multi-level**, and in addition the super-/subscript characters \wedge and $_$ are not displayed.

font-latex-fontify-script-max-level [User Option]

Maximum scriptification level for which script faces are applied.

The faces **font-latex-superscript-face** and **font-latex-subscript-face** define custom **:height** values < 1.0. Therefore, scripts are displayed with a slightly smaller

font than normal math text. If `font-latex-fontify-script` is `multi-level` or `invisible`, the font size becomes too small to be readable after a few levels. This option allows to specify the maximum level after which the size of the script text won't be shrunk anymore.

For example, in the expression $x^{\{y^{\{z^a b\}}\}}$, x has scriptification level 0, y has level 1, z has level 2, and both a and b have scriptification level 3.

If `font-latex-fontify-script-max-level` was 2, then z , a , and b would have the same font size. If it was 3 or more, then a and b were smaller than z just in the same way as z is smaller than y and y is smaller than x .

The script characters ‘`^`’ and ‘`_`’ themselves are also fontified with an own face named `font-latex-script-char-face`.

`font-latex-script-display` [User Option]

Display specification for subscript and superscript content. The `car` is used for subscript, the `cdr` is used for superscript. The feature is implemented using so-called display properties. For information on what exactly to specify for the values, see Section “Other Display Specifications” in *GNU Emacs Lisp Reference Manual*.

3.1.4 Verbatim macros and environments

Usually it is not desirable to have content to be typeset verbatim highlighted according to \LaTeX syntax. Therefore this content will be fontified uniformly with the face `font-latex-verbatim-face`.

`font-latex` differentiates three different types of verbatim constructs for fontification. Macros with special characters like `|` as delimiters, macros with braces, and environments. Which macros and environments are recognized is controlled by the variables `LaTeX-verbatim-macros-with-delims`, `LaTeX-verbatim-macros-with-braces`, and `LaTeX-verbatim-environments` respectively.

3.1.5 Faces used by font-latex

In case you want to change the colors and fonts used by `font-latex` please refer to the faces mentioned in the explanations above and use `M-x customize-face RET <face> RET`. All faces defined by `font-latex` are accessible through a customization group by typing `M-x customize-group RET font-latex-highlighting-faces RET`.

3.1.6 Known fontification problems

In certain cases the fontification machinery fails to interpret buffer contents correctly. This can lead to color bleed, i.e. large parts of a buffer get fontified with an inappropriate face. A typical situation for this to happen is the use of a dollar sign (`$`) in a verbatim macro or environment. If `font-latex` is not aware of the verbatim construct, it assumes the dollar sign to be a toggle for mathematics and fontifies the following buffer content with the respective face until it finds a closing dollar sign or till the end of the buffer.

As a remedy you can make the verbatim construct known to `font-latex` (see Section 3.1.4 [Verbatim content], page 42). If this is not possible, you can insert a commented dollar sign (`%$`) at the next suitable end of line as a quick workaround. In `docTeX` documents, `^^A$` is also available for similar purpose.

3.2 Folding Macros and Environments

A popular complaint about markup languages like \TeX and \LaTeX is that there is too much clutter in the source text and that one cannot focus well on the content. There are macros where you are only interested in the content they are enclosing, like font specifiers where the content might already be fontified in a special way by font locking. Or macros the content of which you only want to see when actually editing it, like footnotes or citations. Similarly you might find certain environments or comments distracting when trying to concentrate on the body of your document.

With \AUCTeX 's folding functionality you can collapse those items and replace them by a fixed string, the content of one of their arguments, or a mixture of both. If you want to make the original text visible again in order to view or edit it, move point sideways onto the placeholder (also called display string) or left-click with the mouse pointer on it. The macro or environment will unfold automatically, stay open as long as point is inside of it and collapse again once you move point out of it. (Note that folding of environments currently does not work in every \AUCTeX mode.)

In order to use this feature, you have to activate `TeX-fold-mode` which will activate the auto-reveal feature and the necessary commands to hide and show macros and environments. You can activate the mode in a certain buffer by typing the command `M-x TeX-fold-mode RET` or using the keyboard shortcut `C-c C-o C-f`. If you want to use it every time you edit a \LaTeX document, add it to a hook:

```
(add-hook 'LaTeX-mode-hook (lambda ()
                             (TeX-fold-mode 1)))
```

If it should be activated in all \AUCTeX modes, use `TeX-mode-hook` instead of `LaTeX-mode-hook`.

Once the mode is active there are several commands available to hide and show macros, environments and comments:

TeX-fold-buffer [Command]
(`C-c C-o C-b`) Hide all foldable items in the current buffer according to the setting of `TeX-fold-type-list`.

If you want to have this done automatically every time you open a file, add it to a hook and make sure the function is called after font locking is set up for the buffer. The following code should accomplish this:

```
(add-hook 'find-file-hook #'TeX-fold-buffer t)
```

The command can be used any time to refresh the whole buffer and fold any new macros and environments which were inserted after the last invocation of the command.

TeX-fold-type-list [User Option]
List of symbols determining the item classes to consider for folding. This can be macros, environments and comments. Per default only macros and environments are folded.

TeX-fold-force-fontify [User Option]
In order for all folded content to get the right faces, the whole buffer has to be fontified before folding is carried out. `TeX-fold-buffer` therefore will force fontification of

unfontified regions. As this will prolong the time folding takes, you can prevent forced fontification by customizing the variable `TeX-fold-force-fontify`.

TeX-fold-auto [User Option]

By default, a macro inserted with `TeX-insert-macro` (*C-c C-m*) will not be folded. Set this variable to a non-`nil` value to automatically fold macros as soon as they are inserted.

TeX-fold-preserve-comments [User Option]

By default items found in comments will be folded. If your comments often contain unfinished code this might lead to problems. Give this variable a non-`nil` value and foldable items in your comments will be left alone.

TeX-fold-unfold-around-mark [User Option]

When this variable is non-`nil` and there is an active region, text around the mark will be kept unfolded.

TeX-fold-region-functions [User Option]

This variable is a list of functions which allow the user, or external packages, to fold additional `LATEX` constructs beyond those supported by default. By default, it is configured to support folding for verbatim environments and quotes.

TeX-fold-region [Command]

(*C-c C-o C-r*) Hide all configured macros in the marked region.

TeX-fold-paragraph [Command]

(*C-c C-o C-p*) Hide all configured macros in the paragraph containing point.

TeX-fold-section [Command]

(*C-c C-o C-s*) Hide all configured macros in the section containing point.

TeX-fold-macro [Command]

(*C-c C-o C-m*) Hide the macro on which point currently is located. If the name of the macro is found in `TeX-fold-macro-spec-list`, the respective display string will be shown instead. If it is not found, the name of the macro in square brackets or the default string for unspecified macros (`TeX-fold-unspec-macro-display-string`) will be shown, depending on the value of the variable `TeX-fold-unspec-use-name`.

TeX-fold-env [Command]

(*C-c C-o C-e*) Hide the environment on which point currently is located. The behavior regarding the display string is analogous to `TeX-fold-macro` and determined by the variables `TeX-fold-env-spec-list` and `TeX-fold-unspec-env-display-string` respectively.

TeX-fold-math [Command]

Hide the math macro on which point currently is located. If the name of the macro is found in `TeX-fold-math-spec-list`, the respective display string will be shown instead. If it is not found, the name of the macro in square brackets or the default string for unspecified macros (`TeX-fold-unspec-macro-display-string`) will be shown, depending on the value of the variable `TeX-fold-unspec-use-name`.

- TeX-fold-comment** [Command]
 (*C-c C-o C-c*) Hide the comment point is located on.
- TeX-fold-clearout-buffer** [Command]
 (*C-c C-o b*) Permanently unfold all macros and environments in the current buffer.
- TeX-fold-clearout-region** [Command]
 (*C-c C-o r*) Permanently unfold all macros and environments in the marked region.
- TeX-fold-clearout-paragraph** [Command]
 (*C-c C-o p*) Permanently unfold all macros and environments in the paragraph containing point.
- TeX-fold-clearout-section** [Command]
 (*C-c C-o s*) Permanently unfold all macros and environments in the section containing point.
- TeX-fold-clearout-item** [Command]
 (*C-c C-o i*) Permanently show the macro or environment on which point currently is located. In contrast to temporarily opening the macro when point is moved sideways onto it, the macro will be permanently unfolded and will not collapse again once point is leaving it.
- TeX-fold-dwim** [Command]
 (*C-c C-o C-o*) Hide or show items according to the current context. If there is folded content, unfold it. If there is a marked region, fold all configured content in this region. If there is no folded content but a macro or environment, fold it.

In case you want to use a different prefix than *C-c C-o* for these commands you can customize the variable `TeX-fold-command-prefix`. (Note that this will not change the key binding for activating the mode.)

The commands above will only take macros or environments into consideration which are specified in the variables `TeX-fold-macro-spec-list` or `TeX-fold-env-spec-list` respectively.

- TeX-fold-macro-spec-list** [User Option]

List of replacement specifiers and macros to fold. The specifier can be a string, an integer or a function symbol.

If you specify a string, it will be used as a display replacement for the whole macro. Numbers in braces, brackets, parens or angle brackets will be replaced by the respective macro argument. For example ‘{1}’ will be replaced by the first mandatory argument of the macro. One can also define alternatives within the specifier which are used if an argument is not found. Alternatives are separated by ‘||’. They are most useful with optional arguments. As an example, the default specifier for ‘\item’ is ‘[1]:||*’ which means that if there is an optional argument, its value is shown followed by a colon. If there is no optional argument, only an asterisk is used as the display string.

If you specify a number as the first element, the content of the respective mandatory argument of a `LATEX` macro will be used as the placeholder.

If the first element is a function symbol, the function will be called with all mandatory arguments of the macro, and with point positioned at the beginning of the macro. The result of the function call will be used as a replacement for the macro. Such functions typically return a string, but may also return the symbol `abort` to indicate that the macro should not be folded.

The placeholder is made by copying the text from the buffer together with its properties, i.e. its face as well. If fontification has not happened when this is done (e.g. because of lazy font locking) the intended fontification will not show up. As a workaround you can leave Emacs idle a few seconds and wait for stealth font locking to finish before you fold the buffer. Or you just re-fold the buffer with `TeX-fold-buffer` when you notice a wrong fontification.

TeX-fold-env-spec-list [User Option]

List of display strings or argument numbers and environments to fold. Argument numbers refer to the ‘`\begin`’ statement. That means if you have e.g. ‘`\begin{tabularx}{\linewidth}{XXX} ... \end{tabularx}`’ and specify 3 as the argument number, the resulting display string will be “XXX”.

TeX-fold-math-spec-list [User Option]

List of display strings and math macros to fold.

The variables `TeX-fold-macro-spec-list`, `TeX-fold-env-spec-list`, and `TeX-fold-math-spec-list` apply to any AUCTeX mode. If you want to make settings which are only applied to L^AT_EX mode, you can use the mode-specific variables `LaTeX-fold-macro-spec-list`, `LaTeX-fold-env-spec-list`, and `LaTeX-fold-math-spec-list`.

TeX-fold-unspec-macro-display-string [User Option]

Default display string for macros which are not specified in `TeX-fold-macro-spec-list`.

TeX-fold-unspec-env-display-string [User Option]

Default display string for environments which are not specified in `TeX-fold-env-spec-list`.

TeX-fold-unspec-use-name [User Option]

If non-`nil` the name of the macro or environment surrounded by square brackets is used as display string, otherwise the defaults specified in `TeX-fold-unspec-macro-display-string` or `TeX-fold-unspec-env-display-string` respectively.

When you hover with the mouse pointer over folded content, its original text will be shown in a tooltip or the echo area depending on Tooltip mode being activate. In order to avoid exorbitantly big tooltips and to cater for the limited space in the echo area the content will be cropped after a certain amount of characters defined by the variable `TeX-fold-help-echo-max-length`.

TeX-fold-help-echo-max-length [User Option]

Maximum length of original text displayed in a tooltip or the echo area for folded content. Set it to zero in order to disable this feature.

TeX-fold-auto-reveal [User Option]

This option determines the auto-reveal behavior when the point enters the folded portion of the buffer. Possible values and the corresponding behaviors are:

- **t**: AUCTEX always opens the folded expression and reveal the original source text.
- **nil**: AUCTEX never reveals.
- *symbol*: The value of *symbol* is used as a boolean flag. If *symbol* isn't bound as a variable, it is treated as **nil**.
- Cons cell (*function . arguments*): AUCTEX calls the *function* with arbitrary number of *arguments*. The return value is regarded as a boolean flag.

The default behavior is to reveal when the point enters the folded portion via one of the commands specified in **TeX-fold-auto-reveal-commands**.

TeX-fold-auto-reveal-commands [User Option]

This is a list of commands, consulted under the default behavior of **TeX-fold-auto-reveal**. By default, it consists of the commands **left**, **right**, **C-b**, **C-f** or mouse click under standard key binding.

TeX-fold-alert-color [User Option]

This is the color used to fold '**\alert{...}**' macros (for the replacement specifier given by the default value of **TeX-fold-macro-spec-list**).

TeX-fold-begin-end-spec-list [User Option]

List of replacement specifiers for '**\begin{env}**' and '**\end{env}**' macros. This option is used only when the replacement specifiers for '**begin**' and '**end**' macros in **TeX-fold-macro-spec-list** are set to their default values: **TeX-fold-begin-display** and **TeX-fold-end-display**, respectively.

Each item in the list consists of two elements. The first element is a cons cell (*BEGIN . END*), where *BEGIN* and *END* are the display specifications for '**\begin{...}**' and '**\end{...}**' macros, respectively. Each specification can be either:

- A string, which will be used directly as the fold display string.
- A function symbol, which will be called with two arguments: the (unabbreviated) environment name and a list of the remaining mandatory macro arguments. The function should return a string to be used as the fold display.

The second element is a list of environment types. Each type can be:

- A string representing the environment name (e.g., **"remark"**).
- A list with the first element being the environment name and the remaining elements being any abbreviated forms of that name (e.g., (**"remark" "rem" "rmk"**)).

For example:

```
(((">" . "<") ("itemize" "enumerate" "description" "frame"))
((TeX-fold-format-theorem-environment . "*")
 ("note" ("theorem" "thm"))))
```

In this example:

- ‘`\begin{itemize}`’, ‘`\begin{enumerate}`’, ‘`\begin{description}`’ and ‘`\begin{frame}`’ will fold to ‘>’, while their corresponding ‘`\end{...}`’ macros will fold to ‘<’.
- ‘`\begin{note}`’ will fold to ‘Note’.
- ‘`\begin{theorem}`’ and ‘`\begin{thm}`’ will fold to ‘Theorem’.
- If an optional argument is provided, e.g., ‘`\begin{theorem}[Foo]`’, it will fold to ‘Theorem (Foo)’.
- All corresponding ‘`\end{...}`’ macros for ‘note’, ‘theorem’, and ‘thm’ will fold to ‘*’.

To disable folding for ‘`\begin{...}`’ and ‘`\end{...}`’ macros, you can either set this variable to `nil`, or remove the display specifications for ‘begin’ and ‘end’ from `TeX-fold-macro-spec-list`.

TeX-fold-bib-file [User Option]

The default folding behavior for ‘`\cite{...}`’ macros that point to a BibTeX entry is to replace them with a string of the form ‘`[XYZ99]`’, formed using the authors’ last names and the publication year. If AUCTeX cannot find the required BibTeX entries in any bib files included in the current document, then, as a backup, it searches the files specified in `TeX-fold-bib-file`. This may be useful when using ‘`\thebibliography{...}`’ rather than BibTeX, or when working in non-file buffers.

TeX-fold-open-quote [User Option]

Folded version of opening quote. This string is used to replace opening quotes when folding L^AT_EX quotes.

TeX-fold-close-quote [User Option]

Folded version of closing quote. This string is used to replace closing quotes when folding L^AT_EX quotes.

TeX-fold-quotes-on-insert [User Option]

If non-`nil`, automatically fold L^AT_EX quotes when they are inserted. This option is consulted by `TeX-insert-quote` when determining whether to fold newly inserted quotes.

3.3 Outlining the Document

AUCTeX supports the standard outline minor mode using L^AT_EX/ConT_EXt sectioning commands as header lines. See Section “Outline Mode” in *GNU Emacs Manual*.

You can add your own headings by setting the variable `TeX-outline-extra`.

TeX-outline-extra [User Option]

List of extra T_EX outline levels.

Each element is a list with two entries. The first entry is the regular expression matching a header, and the second is the level of the header. A ‘`^`’ is automatically prepended to the regular expressions in the list, so they must match text at the beginning of the line.

See `LaTeX-section-list` or `ConTeXt-interface-section-list` for existing header levels.

The following example add ‘`\item`’ and ‘`\bibliography`’ headers, with ‘`\bibliography`’ at the same outline level as ‘`\section`’, and ‘`\item`’ being below ‘`\subparagraph`’.

```
(setq TeX-outline-extra
      '((([" \t"]*\\\\\\\\(bib\\)?item\\b" 7)
          (("\\\\\\bibliography\\b" 2))))
```

3.4 Narrowing

Sometimes you want to focus your attention to a limited region of the code. You can do that by restricting the text addressable by editing commands and hiding the rest of the buffer with the narrowing functions, see Section “Narrowing” in *GNU Emacs Manual*. In addition, AUCTeX provides a couple of other commands to narrow the buffer to a group, i.e. a region enclosed in a pair of curly braces, and to L^AT_EX environments.

TeX-narrow-to-group [Command]
(*C-x n g*) Make text outside current group invisible.

LaTeX-narrow-to-environment count [Command]
(*C-x n e*) Make text outside current environment invisible. With optional argument *count* keep visible that number of enclosing environments.

Like other standard narrowing functions, the above commands are disabled. Attempting to use them asks for confirmation and gives you the option of enabling them; if you enable the commands, confirmation will no longer be required for them.

3.5 Prettifying

Emacs 25 is able to prettify symbols in programming language buffers, see Section “Misc for Programs” in *GNU Emacs Manual*. The canonical example is to display `(lambda () ...)` as $(\lambda () \dots)$ in Lisp buffers.

AUCTeX can use this feature in order to display certain math macros and Greek letters using their Unicode representation, too. For example, the T_EX code `\alpha \times \beta` will be displayed as $\alpha \times \beta$. When point is on one of the characters, it’ll be unprettified automatically, meaning you see the verbatim text again. For this behavior however you need to set `prettify-symbols-unprettify-at-point` to `t` or `right-edge` which will unprettify the symbol when point moves into or near it.

To enable prettification in AUCTeX, simply add this to your init file:

```
(add-hook 'TeX-mode-hook #'prettify-symbols-mode)
```

If you enabled prettification globally with `(global-prettify-symbols-mode)`, then it’s automatically enabled in AUCTeX, too.

You can also add custom symbol unicode-character pairs for prettification by adding to `tex--prettify-symbols-alist`. Note that this variable is part of Emacs’ stock `tex-mode.el` and used by that and AUCTeX.

4 Starting Processors, Viewers and Other Programs

The most powerful features of AUCT_EX may be those allowing you to run T_EX, L^AT_EX, ConT_EXt and other external commands like BibT_EX and `makeindex` from within Emacs, viewing and printing the results, and moreover allowing you to *debug* your documents.

AUCT_EX comes with a special tool bar for T_EX and L^AT_EX which provides buttons for the most important commands. You can enable or disable it by customizing the options `plain-TeX-enable-toolbar` and `LaTeX-enable-toolbar` in the `TeX-tool-bar` customization group. You can also customize the buttons by the options `TeX-bar-TeX-buttons`, `TeX-bar-TeX-all-button-alist`, `TeX-bar-LaTeX-buttons` and `TeX-bar-LaTeX-button-alist`.

4.1 Executing Commands

Formatting the document with T_EX, L^AT_EX or ConT_EXt, viewing with a previewer, printing the document, running BibT_EX, making an index, or checking the document with `lacheck` or `chktex` all require running an external command.

4.1.1 Starting a Command on a Document or Region

There are two ways to run an external command, you can either run it on the current document with `TeX-command-master`, or on the current region with `TeX-command-region`. A special case of running T_EX on a region is `TeX-command-buffer` which differs from `TeX-command-master` if the current buffer is not its own master file.

TeX-command-master [Command]
 (*C-c C-c*) Query the user for a command, and run it on the master file associated with the current buffer. The name of the master file is controlled by the variable `TeX-master`. The available commands are controlled by the variable `TeX-command-list`.

TeX-command-region [Command]
 (*C-c C-r*) Query the user for a command, and run it on the contents of the selected region. The region contents are written into the region file, after extracting the header and trailer from the master file. If `mark` is inactive (which can happen with Transient Mark mode), use the old region. See also the command `TeX-pin-region` about how to fix a region.

The name of the region file is controlled by the variable `TeX-region`. The name of the master file is controlled by the variable `TeX-master`. The header is all text up to the line matching the regular expression `TeX-header-end`. The trailer is all text from the line matching the regular expression `TeX-trailer-start`. The available commands are controlled by the variable `TeX-command-list`.

TeX-command-buffer [Command]
 (*C-c C-b*) Query the user for a command, and apply it to the contents of the current buffer. The buffer contents are written into the region file, after extracting the header and trailer from the master file. The command is then actually run on the region file. See above for details.

LaTeX-command-section [Command]

(*C-c C-z*) Query the user for a command, and apply it to the current section (or part, chapter, subsection, paragraph, or subparagraph). What makes the current section is determined by `LaTeX-command-section-level` which can be enlarged/shrunk using `LaTeX-command-section-change-level` (*C-c M-z*). The given numeric prefix arg is added to the current value of `LaTeX-command-section-level`. By default, `LaTeX-command-section-level` is initialized with the current document's `LaTeX-largest-level`. The buffer contents are written into the region file, after extracting the header and trailer from the master file. The command is then actually run on the region file. See `TeX-command-region` for details.

It is also possible to compile automatically the whole document until it is ready with a single command: `TeX-command-run-all`.

TeX-command-run-all [Command]

(*C-c C-a*) Compile the current document until an error occurs or it is finished. If compilation finishes successfully, run the viewer at the end.

Here are some relevant variables.

TeX-region [User Option]

The name of the file for temporarily storing the text when formatting the current region.

TeX-kill-process-without-query [User Option]

This boolean option controls whether AUCTeX should ask user before aborting a running process for a TeX document. It can be set as a file-local variable.

TeX-header-end [User Option]

A regular expression matching the end of the header. By default, this is `'\begin{document}'` in LaTeX mode and `'%**end of header'` in plain TeX mode.

TeX-trailer-start [User Option]

A regular expression matching the start of the trailer. By default, this is `'\end{document}'` in LaTeX mode and `'\bye'` in plain TeX mode.

If you want to change the values of `TeX-header-end` and `TeX-trailer-start` you can do this for all files by setting the variables in a mode hook or per file by specifying them as file variables (see Section “File Variables” in *The Emacs Editor*).

TeX-pin-region [Command]

(*C-c C-t C-r*) If you don't have a mode like Transient Mark mode active, where marks get disabled automatically, the region would need to get properly set before each call to `TeX-command-region`. If you fix the current region with *C-c C-t C-r*, then it will get used for more commands even though mark and point may change. An explicitly activated mark, however, will always define a new region when calling `TeX-command-region`.

If the last process you started was on the region, the commands described in Section 4.3 [Debugging], page 62, and Section 4.5 [Control], page 65, will work on that process, otherwise they will work on the process associated with the current document.

Don't run more than one process at the same time. AUCTeX doesn't support simultaneous typeset including region typeset. Wait for the previous process to finish before you start a new process, in particular when you are editing multiple documents in parallel. This limitation applies for preview by preview-latex as well.

4.1.2 Selecting and Executing a Command

Once you started the command selection with `C-c C-c`, `C-c C-r` or `C-c C-b` you will be prompted for the type of command. AUCTeX will try to guess which command is appropriate in the given situation and propose it as default. Usually this is a processor like 'TeX' or 'LaTeX' if the document was changed or a viewer if the document was just typeset. Other commands can be selected in the minibuffer with completion support by typing TAB.

The available commands are defined by the variable `TeX-command-list`. Per default it includes commands for typesetting the document (e.g. 'LaTeX'), for viewing the output ('View'), for printing ('Print'), for generating an index ('Index') or for spell checking ('Spell') to name but a few. You can also add your own commands by adding entries to `TeX-command-list`. Refer to its doc string for information about its syntax. You might also want to look at `TeX-expand-list` to learn about the expanders you can use in `TeX-command-list`.

Note that the default of the variable occasionally changes. Therefore it is advisable to add to the list rather than overwriting it. You can do this with a call to `add-to-list` in your init file. For example, if you wanted to add a command for running a program called 'foo' on the master or region file, you could do this with the following form.

```
(with-eval-after-load 'tex
  (add-to-list 'TeX-command-list
    ("Foo" "foo %s" TeX-run-command t t :help "Run foo")
    t))
```

As mentioned before, AUCTeX will try to guess what command you want to invoke. If you want to use another command than 'TeX', 'LaTeX' or whatever processor AUCTeX thinks is appropriate for the current mode, set the variable `TeX-command-default`. You can do this for all files by setting it in a mode hook or per file by specifying it as a file variable (see Section "File Variables" in *The Emacs Editor*).

TeX-command-default [User Option]

The default command to run in this buffer. Must be an entry in `TeX-command-list`.

In case you use biblatex in a document, when automatic parsing is enabled AUCTeX checks the value of 'backend' option given to biblatex at load time to decide whether to use BibTeX or Biber for bibliography processing. Should AUCTeX fail to detect the right backend, you can use the file local `LaTeX-biblatex-use-Biber` variable.

LaTeX-biblatex-use-Biber [Variable]

If this boolean variable is set as file local, it tells to AUCTeX whether to use Biber with biblatex. In this case, the autodetection of the biblatex backend will be overridden.

You may want to set locally this variable if automatic parsing is not enabled.

After confirming a command to execute, AUCTeX will try to save any buffers related to the document, and check if the document needs to be reformatted. If the variable `TeX-save-query` is non-`nil`, AUCTeX will query before saving each file. By default AUCTeX

will check emacs buffers associated with files in the current directory, in one of the `TeX-macro-private` directories, and in the `TeX-macro-global` directories. You can change this by setting the variable `TeX-check-path`.

`TeX-check-path`

[User Option]

Directory path to search for dependencies.

If `nil`, just check the current file. Used when checking if any files have changed.

When performing spell checking on a document or a region (invoked through AUCTeX's 'Spell' command or `M-x ispell RET`), you want the spell checking program to skip certain macro arguments and environments, most notably the arguments of referencing macros and the contents of verbatim environments. The skipped parts are controlled by variable `ispell-tex-skip-alists` provided by `ispell.el`. AUCTeX has a library which can be added to this variable depending on the value of `TeX-ispell-extend-skip-list` which is set to `t` by default.

`TeX-ispell-extend-skip-list`

[User Option]

This boolean option controls whether AUCTeX activates its extension for skipping certain macro arguments and environments when spell checking.

When non-`nil`, AUCTeX loads the file `tex-ispell.el` and adds its content to `ispell-tex-skip-alists`. This library can and will never be complete, but the interface can be used to add selected and private macro names within your init file or on a file local basis.

`ispell-tex-skip-alists` has the following structure:

```
(defvar ispell-tex-skip-alists
  '((; First list
    ("\\\\addcontentsline"          ispell-tex-arg-end 2)
    ("\\\\([aA]lph\\\\|arabic\\\\)" ispell-tex-arg-end)
    ("\\\\makebox"                  ispell-tex-arg-end 0)
    ("\\\\documentclass" . "\\\\begin{document}"))
    (; Second list
    ("\\(figure\\\\|table\\\\)\\\\*?" ispell-tex-arg-end 0)
    ("list"                      ispell-tex-arg-end 2)
    ("verbatim\\\\*?" . "\\\\end{verbatim\\\\*?}"))
    "Lists of regions to be skipped in TeX mode.
First list is used raw.
Second list has key placed inside \\begin{.}")
```

Each item is an alist and the structure of it is described in `ispell-skip-region-alist`:

```
(defvar ispell-skip-region-alist
  '(...))
"Alist expressing beginning and end of regions not to spell check.
The alist key must be a regular expression.
Valid forms include:
(KEY) - just skip the key.
(KEY . REGEXP) - skip to the end of REGEXP.
```

```

                REGEXP may be string or symbol.
    (KEY REGEXP) - skip to end of REGEXP.  REGEXP must be a string.
    (KEY FUNCTION ARGS) - FUNCTION called with ARGS
                        returns end of region.")

```

Let's go through the first list of `ispell-tex-skip-alist`s line by line:

```
("\\\\\\addcontentsline"          ispell-tex-arg-end 2)
```

`key` is the string "`\\\\\\addcontentsline`", `function` is `ispell-tex-arg-end` called with `args`, here 2. `ispell-tex-arg-end` is a function provided by `ispell.el` which skips as many subsequent optional arguments in square brackets as it sees and then skips `args` number of mandatory arguments in braces. Omitting `args` means skip 1 mandatory argument. In practice, when you have something like this in your document:

```
\addcontentsline{toc}{chapter}{Some text}
```

The first two arguments are left out and 'Some text' will be spell checked. For the next line

```
("\\\\\\([aA]lph\\\\|arabic\\\\)"  ispell-tex-arg-end)
```

the name of the counter as argument is skipped. Next line is

```
("\\\\\\makebox"                  ispell-tex-arg-end 0)
```

where only optional arguments are skipped, the first mandatory argument is checked, e.g.

```
\makebox[0pt][l]{Some text}
```

Finally, the next line

```
("\\\\\\documentclass" . "\\\\\\begin{document}"))
```

ensures that the entire preamble of a document is discarded. Second list works the same; it is more convenient for environments since `key` is wrapped inside `\begin{}`.

AUCTEX provides two functions to add items to `car` and `cdr` of `ispell-tex-arg-end`, namely `TeX-ispell-skip-setcar` and `TeX-ispell-skip-setcdr`. The argument of these functions is exactly as in `ispell-tex-skip-alist`s. Additions can be done via `init` file, e.g.:

```

(with-eval-after-load 'tex-ispell
  (TeX-ispell-skip-setcar
    '("\\\\\\mymacro" ispell-tex-arg-end)))
  (TeX-ispell-skip-setcdr
    '(("myverbatim" . "\\\\\\end{myverbatim}"))))

```

Another possibility is to use file local additions at the end of your TEX file, e.g.:

```

%%% Local Variables:
%%% mode: LaTeX
%%% TeX-master: t
%%% eval: (TeX-ispell-skip-setcar '("\\\\\\mymacro" . "{[-0-9]+}"))
%%% End:

```

Finally, AUCTEX provides a function called `TeX-ispell-tex-arg-end` which sees more arguments than `ispell-tex-arg-end`. Refer to its doc string for more information.

AUCT_EX also provides a facility to skip the argument of in-line verbatim macros like ‘\Verb’ from `fancyvrb.sty` or ‘\mintinline’ from `minted.sty`. Characters delimiting the verbatim text are stored in `TeX-ispell-verb-delimiters`.

TeX-ispell-verb-delimiters [User Option]

String with delimiters recognized for in-line verbatim macros. This variable is initialized to ‘!|#"*/+^~’. Since this string is used to build a character alternative inside a regular expression, special characters ‘^’ and ‘~’ should come last. Other characters like opening brace ‘{’, asterisk ‘*’ or at sign ‘@’ should be avoided as they are not recognized by `font-latex.el`.

4.1.3 Options for T_EX Processors

There are some options you can customize affecting which processors are invoked or the way this is done and which output they produce as a result. These options control if DVI or PDF output should be produced, if T_EX should be started in interactive or nonstop mode, if source specials or a SyncT_EX file should be produced for making inverse and forward search possible or which T_EX engine should be used instead of regular T_EX, like PDFT_EX, Omega or XeT_EX, and the style error messages are printed with.

TeX-PDF-mode [Command]

(*C-c C-t C-p*) This command toggles the PDF mode of AUCT_EX, a buffer-local minor mode which is enabled by default. You can customize `TeX-PDF-mode` to give it a different default or set it as a file local variable on a per-document basis. This option usually results in calling either PDFT_EX or ordinary T_EX.

TeX-DVI-via-PDFTeX [User Option]

If this is set, DVI will also be produced by calling PDFT_EX, setting `\pdfoutput=0`. This makes it possible to use PDFT_EX features like character protrusion even when producing DVI files. Contemporary T_EX distributions do this anyway, so that you need not enable the option within AUCT_EX.

TeX-interactive-mode [Command]

(*C-c C-t C-i*) This command toggles the interactive mode of AUCT_EX, a global minor mode. You can customize `TeX-interactive-mode` to give it a different default. In interactive mode, T_EX will pause with an error prompt when errors are encountered and wait for the user to type something.

TeX-source-correlate-mode [Command]

(*C-c C-t C-s*) Toggles support for forward and inverse search. Forward search refers to jumping to the place in the previewed document corresponding to where point is located in the document source and inverse search to the other way round. See Section 4.2.2 [I/O Correlation], page 60.

You can permanently activate `TeX-source-correlate-mode` by customizing the variable `TeX-source-correlate-mode`. There is a bunch of customization options for the mode, use *M-x customize-group* RET *TeX-view* RET to find out more.

AUCT_EX is aware of three different means to do I/O correlation: source specials (only DVI output), the `pdfsync` L^AT_EX package (only PDF output) and SyncT_EX.

The choice between source specials and SyncTeX can be controlled with the variable `TeX-source-correlate-method`.

Should you use source specials it has to be stressed *very* strongly however, that source specials can cause differences in page breaks and spacing, can seriously interfere with various packages and should thus *never* be used for the final version of a document. In particular, fine-tuning the page breaks should be done with source specials switched off.

Sometimes you are requested, by journal rules or packages, to compile the document into DVI output. Thus, if you want a PDF document in the end you can either use XeTeX engine, see below for information about how to set engines, or compile the document with `tex` and then convert to PDF with `dvips-ps2pdf` before viewing it. In addition, current Japanese TeX engines cannot generate PDF directly so they rely on DVI-to-PDF converters. Usually `dvipdfmx` command is used for this purpose. You can use the `TeX-PDF-from-DVI` variable to let AUCTeX know you want to generate the final PDF by converting a DVI file.

TeX-PDF-from-DVI

[User Option]

This option controls if and how to produce a PDF file by converting a DVI file.

When `TeX-PDF-mode` is non-nil, if `TeX-PDF-from-DVI` is non-nil too the document is compiled to DVI instead of PDF. When the document is ready, `C-c C-c` will suggest to run the converter to PDF or an intermediate format.

If non-nil, `TeX-PDF-from-DVI` should be the name of the command in `TeX-command-list`, as a string, used to convert the DVI file to PDF or to an intermediate format. Values currently supported are:

- "Dvips": the DVI file is converted to PS with `dvips`. After successfully running it, `ps2pdf` will be the default command to convert the PS file to PDF.
- "Dvipdfmx": the DVI file is converted to PDF with `dvipdfmx`.

(case is significant; note the uppercase ‘D’ in both strings) When the PDF file is finally ready, the next suggested command will be ‘View’ to open the viewer.

This option can also be set as a file local variable, in order to use this conversion on a per-document basis.

Recall the whole sequence of `C-c C-c` commands can be replaced by the single `C-c C-a`.

AUCTeX also allows you to easily select different TeX engines for processing, either by using the entries in the ‘TeXing Options’ submenu below the ‘Command’ menu or by calling the function `TeX-engine-set`. These eventually set the variable `TeX-engine` which you can also modify directly.

TeX-engine

[User Option]

This variable allows you to choose which TeX engine should be used for typesetting the document, i.e. the executables which will be used when you invoke the ‘TeX’ or ‘LaTeX’ commands. The value should be one of the symbols defined in `TeX-engine-alist-builtin` or `TeX-engine-alist`. The symbols ‘default’, ‘xetex’, ‘luatex’ and ‘omega’ are available from the built-in list.

Note that **TeX-engine** is buffer-local, so setting the variable directly or via the above mentioned menu or function will not take effect in other buffers. If you want to activate an engine for all AUCTeX modes, set **TeX-engine** in your init file, e.g. by using *M-x customize-option* RET. If you want to activate it for a certain AUCTeX mode only, set the variable in the respective mode hook. If you want to activate it for certain files, set it through file variables (see Section “File Variables” in *The Emacs Editor*).

Should you need to change the executable names related to the different engine settings, there are some variables you can tweak. Those are **TeX-command**, **LaTeX-command**, **TeX-Omega-command**, **LaTeX-Omega-command**, **ConTeXt-engine** and **ConTeXt-Omega-engine**. The rest of the executables is defined directly in **TeX-engine-alist-builtin**. If you want to override an entry from that, add an entry to **TeX-engine-alist** that starts with the same symbol as that the entry in the built-in list and specify the executables you want to use instead. You can also add entries to **TeX-engine-alist** in order to add support for engines not covered per default.

TeX-engine-alist [User Option]

Alist of TeX engines and associated commands. Each entry is a list with a maximum of five elements. The first element is a symbol used to identify the engine. The second is a string describing the engine. The third is the command to be used for plain TeX. The fourth is the command to be used for LaTeX. The fifth is the command to be used for the `--engine` parameter of ConTeXt’s ‘texexec’ program. Each command can either be a variable or a string. An empty string or `nil` means there is no command available.

In some systems, Emacs cannot inherit the `PATH` environment variable from the shell and thus AUCTeX may not be able to run TeX commands. Before running them, AUCTeX checks if it is able to find those commands and will warn you in case it fails. You can skip this test by changing the option **TeX-check-TeX**.

TeX-check-TeX [User Option]

If non-`nil`, AUCTeX will check if it is able to find a working TeX distribution before running TeX, LaTeX, ConTeXt, etc. It actually checks if can run **TeX-command** command or the shell returns a command not found error. The error code returned by the shell in this case can be set in **TeX-check-TeX-command-not-found** option.

In addition, AUCTeX searches for a line similar to

```
LaTeX2e <2022-11-01> patch level 1
```

in the console log to check whether `latex` command fails or not. If there isn’t such a line when running LaTeX, AUCTeX warns the problem and resets the next default command to “LaTeX”. If this check doesn’t suit for your use case, you can customize the **TeX-LaTeX-sentinel-banner-regexp** option:

TeX-LaTeX-sentinel-banner-regexp [User Option]

When a LaTeX run doesn’t output a banner line matching this regexp, AUCTeX considers that it failed.

Some LaTeX packages requires the document to be compiled with a specific engine. Notable examples are ‘fontspec’ and ‘polyglossia’ packages, which require LuaTeX and

XeTeX engines. If you try to compile a document which loads one of such packages and the set engine is not one of those allowed you will be asked to select a different engine before running the `LATEX` command. If you do not want to be warned by AUCTeX in these cases, customize the option `TeX-check-engine`.

TeX-check-engine [User Option]

This boolean option controls whether AUCTeX should check the correct engine has been set before running `LATEX` commands.

As shown above, AUCTeX handles in a special way most of the main options that can be given to the TeX processors. When you need to pass to the TeX processor arbitrary options not handled by AUCTeX, you can use the file local variable `TeX-command-extra-options`.

TeX-command-extra-options [User Option]

String with the extra options to be given to the TeX processor. For example, if you need to enable the shell escape feature to compile a document, add the following line to the list of local variables of the source file:

```
%%% TeX-command-extra-options: "-shell-escape"
```

By default this option is not safe as a file-local variable because a specially crafted document compiled with shell escape enabled can be used for malicious purposes.

You can customize AUCTeX to show the processor output as it is produced.

TeX-show-compilation [User Option]

If non-nil, the output of TeX compilation is shown in another window.

You can instruct TeX to print error messages in the form ‘file:line:error’ which is similar to the way many compilers format them.

TeX-file-line-error [User Option]

If non-nil, TeX will produce ‘file:line:error’ style error messages.

ConTeXt users can choose between Mark II and Mark IV versions. This is controlled by `ConTeXt-Mark-version` option.

ConTeXt-Mark-version [User Option]

This variable specifies which version of Mark should be used. Values currently supported are "II", the default, and "IV". It can be set globally using customization interface or on a per-file basis, by specifying it as a file variable.

4.2 Viewing the Formatted Output

AUCTeX allows you to start external programs for previewing the formatted output of your document.

4.2.1 Starting Viewers

Viewers are normally invoked by pressing `C-c C-c` once the document is formatted, which will propose the ‘View’ command, or by activating the respective entry in the Command menu. Alternatively you can type `C-c C-v` which calls the function `TeX-view`.

TeX-view [Command]

(*C-c C-v*) Start a viewer without confirmation. The viewer is started either on a region or the master file, depending on the last command issued. This is especially useful for jumping to the location corresponding to point in the viewer when using **TeX-source-correlate-mode**.

AUCTeX will try to guess which type of viewer (DVI, PostScript or PDF) has to be used and what options are to be passed over to it. This decision is based on the output files present in the working directory as well as the class and style options used in the document. For example, if there is a DVI file in your working directory, a DVI viewer will be invoked. In case of a PDF file it will be a PDF viewer. If you specified a special paper format like ‘**a5paper**’ or use the ‘**landscape**’ option, this will be passed to the viewer by the appropriate options. Especially some DVI viewers depend on this kind of information in order to display your document correctly. In case you are using ‘**pstricks**’ or ‘**psfrag**’ in your document, a DVI viewer cannot display the contents correctly and a PostScript viewer will be invoked instead.

The association between the tests for the conditions mentioned above and the viewers is made in the variable **TeX-view-program-selection**. Therefore this variable is the starting point for customization if you want to use other viewers than the ones suggested by default.

TeX-view-program-selection [User Option]

This is a list of predicates and viewers which is evaluated from front to back in order to find out which viewer to call under the given conditions. In the first element of each list item you can reference one or more predicates defined in **TeX-view-predicate-list** or **TeX-view-predicate-list-builtin**. In the second element you can reference a viewer defined in **TeX-view-program-list** or **TeX-view-program-list-builtin**. The viewer of the first item with a positively evaluated predicate is selected.

So **TeX-view-program-selection** only contains references to the actual implementations of predicates and viewer commands respectively which can be found elsewhere. AUCTeX comes with a set of preconfigured predicates and viewer commands which are stored in the variables **TeX-view-predicate-list-builtin** and **TeX-view-program-list-builtin** respectively. If you are not satisfied with those and want to overwrite one of them or add your own definitions, you can do so via the variables **TeX-view-predicate-list** and **TeX-view-program-list**.

TeX-view-predicate-list [User Option]

This is a list of predicates for viewer selection and invocation. The first element of each list item is a symbol and the second element a Lisp form to be evaluated. The form should return **nil** if the predicate is not fulfilled.

A built-in predicate from **TeX-view-predicate-list-builtin** can be overwritten by defining a new predicate with the same symbol.

TeX-view-program-list [User Option]

This is a list of viewer specifications each consisting of a symbolic name and either a command line or a function to be invoked when the viewer is called. If a command line is used, parts of it can be conditionalized by prefixing them with predicates from **TeX-view-predicate-list** or **TeX-view-predicate-list-builtin**. (See the doc

string for the exact format to use.) The command line can also contain placeholders as defined in `TeX-expand-list` and `TeX-expand-list-builtin` which are expanded before the viewer is called.

The third element of each item is a string, or a list of strings, with the name of the executable, or executables, needed to open the output file in the viewer. Placeholders defined in `TeX-expand-list` and `TeX-expand-list-builtin` can be used here. This element is optional and is used to check whether the viewer is actually available on the system.

A built-in viewer spec from `TeX-view-program-list-builtin` can be overwritten by defining a new viewer spec with the same name.

After the viewer is called via either the ‘`View`’ command or the key stroke `C-c C-v`, the window system focus goes and stays on the viewer. If you prefer that the focus is pulled back to Emacs immediately after that and you are using evince-compatible viewer, customize the option `TeX-view-evince-keep-focus`.

TeX-view-evince-keep-focus [User Option]

When this option is non-`nil` and the viewer is compatible with evince, the focus is pulled back to Emacs immediately after the viewer is invoked or refreshed from within AUCTeX.

Note that the viewer selection and invocation as described above will only work if certain default settings in AUCTeX are intact. For one, the whole viewer selection machinery will only be triggered if there is no ‘`%V`’ expander in `TeX-expand-list`. So if you have trouble with the viewer invocation you might check if there is an older customization of the variable in place. In addition, the use of a function in `TeX-view-program-list` only works if the ‘`View`’ command in `TeX-command-list` makes use of the hook `TeX-run-discard-or-function`.

4.2.2 Forward and Inverse Search

Forward and inverse search refer to the correlation between the document source in the editor and the typeset document in the viewer. Forward search allows you to jump to the place in the previewed document corresponding to a certain line in the document source and inverse search vice versa.

AUCTeX supports three methods for forward and inverse search: source specials (only DVI output), the pdfsync L^AT_EX package (only PDF output) and SyncTeX (any type of output). If you want to make use of forward and inverse searching with source specials or SyncTeX, switch on `TeX-source-correlate-mode`. See Section 4.1.3 [Processor Options], page 55, on how to do that. The use of the pdfsync package is detected automatically if document parsing is enabled. Customize the variable `TeX-source-correlate-method` to select the method to use.

TeX-source-correlate-method [User Option]

Method to use for enabling forward and inverse search. This can be ‘`source-specials`’ if source specials should be used, ‘`synctex`’ if SyncTeX should be used, or ‘`auto`’ if AUCTeX should decide.

When the variable is set to ‘auto’, AUCT_EX will always use SyncT_EX if your `latex` processor supports it, source specials otherwise. You must make sure your viewer supports the same method.

It is also possible to specify a different method depending on the output, either DVI or PDF, by setting the variable to an alist of the kind

```
((dvi . '<source-specials or synctex>')
 (pdf . '<source-specials or synctex>'))
```

in which the CDR of each entry is a symbol specifying the method to be used in the corresponding mode. The default value of the variable is

```
((dvi . source-specials)
 (pdf . synctex))
```

which is compatible with the majority of viewers.

Forward search happens automatically upon calling the viewer, e.g. by typing `C-c C-v` (TeX-view). This will open the viewer or bring it to front and display the output page corresponding to the position of point in the source file. AUCT_EX will automatically pass the necessary command line options to the viewer for this to happen.

You can also make special mouse event do forward search at the clicked position. Use `TeX-source-correlate-map`¹ and `TeX-view-mouse` like this:

```
(with-eval-after-load 'tex
 (define-key TeX-source-correlate-map [C-down-mouse-1]
 #'TeX-view-mouse))
```

This example binds `C-down-mouse-1`, which usually opens a concise menu to select buffer, to the command to do forward search.

Upon opening the viewer you will be asked if you want to start a server process (Gnuserv or Emacs server) which is necessary for inverse search. This happens only if there is no server running already. You can customize the variable `TeX-source-correlate-start-server` to inhibit the question and always or never start the server respectively.

TeX-source-correlate-start-server [User Option]

If `TeX-source-correlate-mode` is active and a viewer is invoked, the default behavior is to ask if a server process should be started. Set this variable to `t` if the question should be inhibited and the server should always be started. Set it to `nil` if the server should never be started. Inverse search will not be available in the latter case.

Inverse search, i.e. jumping to the part of your document source in Emacs corresponding to a certain position in the viewer, is triggered from the viewer, typically by a mouse click. Refer to the documentation of your viewer to find out how it has to be configured and what you have to do exactly. In `xdvi` you normally have to use `C-down-mouse-1`.

Note that inverse search with the Evince PDF viewer or its MATE fork Atril might fail in raising the Emacs frame after updating point in your document’s buffer. There is simply no way to raise the Emacs frame reliably across different operating systems and

¹ The keymap name is `TeX-source-correlate-map`, not `TeX-source-correlate-mode-map`. Actually, this keymap isn’t implemented as minor mode map of `TeX-source-correlate-mode`, in order that its bindings don’t affect buffers outside of AUCT_EX.

different window managers with their different focus stealing policies. If the Emacs frame is not raised after performing an inverse search from Evince or Atril, you can customize the following option.

TeX-raise-frame-function [User Option]

A function that will be called after performing an inverse search from Evince or Atril in order to raise the current Emacs frame.

If your Emacs frame is already raised in that situation, just leave this variable set to its default value `raise-frame`. Otherwise, here are some alternative settings that work for some users.

```
;; Alternative 1: For some users, `x-focus-frame' works.
(setq TeX-raise-frame-function #'x-focus-frame)

;; Alternative 2: Under GNOME 3.20 (and probably others), it
;; seems some focus stealing prevention policy prohibits that
;; some window gets the focus immediately after the user has
;; clicked in some other window. Here waiting a bit before
;; issuing the request seems to work.
(setq TeX-raise-frame-function
  (lambda ()
    (run-at-time 0.5 nil #'x-focus-frame)))

;; Alternative 3: Use the external wmctrl tool in order to
;; force Emacs into the focus.
(setq TeX-raise-frame-function
  (lambda ()
    (call-process
      "wmctrl" nil nil nil "-i" "-R"
      (frame-parameter (selected-frame) 'outer-window-id))))
```

4.3 Catching the errors

Once you've formatted your document you may 'debug' it, i.e. browse through the errors (La)TeX reported. You may also have a look at a nicely formatted list of all errors and warnings reported by the compiler.

TeX-next-error *arg* *reparse* [Command]

(*C-c `*) Go to the next error reported by TeX. The view will be split in two, with the cursor placed as close as possible to the error in the top view. In the bottom view, the error message will be displayed along with some explanatory text.

An optional numeric *arg*, positive or negative, specifies how many error messages to move. A negative *arg* means to move back to previous error messages, see also `TeX-previous-error`.

The optional *reparse* argument makes AUCTeX reparse the error message buffer and start the debugging from the first error. This can also be achieved by calling the function with a prefix argument (*C-u*).

TeX-previous-error *arg* [Command]
 (*M-g p*) Go to the previous error reported by T_EX. An optional numeric *arg* specifies how many error messages to move backward. This is like calling **TeX-next-error** with a negative argument.

The command **TeX-previous-error** works only if AUCT_EX can parse the whole T_EX log buffer. This is controlled by the **TeX-parse-all-errors** variable.

TeX-parse-all-errors [User Option]
 If **t**, AUCT_EX automatically parses the whole output log buffer right after running a T_EX command, in order to collect all warnings and errors. This makes it possible to navigate back and forth between the error messages using **TeX-next-error** and **TeX-previous-error**. This is the default. If **nil**, AUCT_EX does not parse the whole output log buffer and **TeX-previous-error** cannot be used.

As default, AUCT_EX will display a special help buffer containing the error reported by T_EX along with the documentation. There is however an ‘expert’ option, which allows you to display the real T_EX output.

TeX-display-help [User Option]
 If **t**, AUCT_EX will automatically display a help text whenever an error is encountered using **TeX-next-error** (*C-c `*). If **nil**, a terse information about the error is displayed in the echo area. If **expert** AUCT_EX will display the output buffer with the raw T_EX output.

4.3.1 Controlling warnings to be reported

Normally AUCT_EX will only report real errors, but you may as well ask it to report ‘bad boxes’ and warnings as well.

TeX-toggle-debug-bad-boxes [Command]
 (*C-c C-t C-b*) Toggle whether AUCT_EX should stop at bad boxes (i.e. overfull and underfull boxes) as well as normal errors. The boolean option **TeX-debug-bad-boxes** is set accordingly.

TeX-toggle-debug-warnings [Command]
 (*C-c C-t C-w*) Toggle whether AUCT_EX should stop at warnings as well as normal errors. The boolean option **TeX-debug-warnings** is set accordingly.

While many users desire to have warnings reported after compilation, there are certain warnings that are considered unimportant and users want to ignore them. For a more fine-grained control of what kinds of warnings should be shown after compilation, AUCT_EX provides other options.

TeX-ignore-warnings [User Option]
 Controls which warnings are to be ignored.
 It can be a regexp matching the message of the warnings to be ignored.
 More advanced users can set also this option to a symbol with the name of a custom function taking as arguments all the information of the warning listed in **TeX-error-list** variable, except the last one about whether to ignore the warning. See the code of **TeX-warning** function and the documentation of **TeX-error-list** for more details.

TeX-toggle-suppress-ignored-warnings [Command]

(*C-c C-t C-x*) Toggle whether AUCTeX should actually hide the ignored warnings specified with **TeX-ignore-warnings**. The boolean option **TeX-suppress-ignored-warnings** is set accordingly. If this is **nil**, all warnings are shown, even those matched by **TeX-ignore-warnings**, otherwise these are hidden.

Note that **TeX-debug-warnings** takes the precedence: if it is **nil**, all warnings are hidden in any case.

4.3.2 List of all errors and warnings

When the option **TeX-parse-all-errors** is non-**nil**, you will be also able to open an overview of all errors and warnings reported by the TeX compiler.

TeX-error-overview [Command]

Show an overview of the errors and warnings occurred in the last TeX run.

In this window you can visit the error on which point is by pressing **RET**, and visit the next or previous issue by pressing **n** or **p** respectively. A prefix argument to these keys specifies how many errors to move forward or backward. You can visit an error also by clicking on its message. Jump to error point in the source code with **j**, and use **l** to see the error in the log buffer. In addition, you can toggle visibility of bad boxes, generic warnings, and ignored warnings with **b**, **w**, and **x**, respectively (see Section 4.3.1 [Ignoring warnings], page 63, for details). Press **q** to quit the overview.

TeX-error-overview-open-after-TeX-run [User Option]

When this boolean variable is non-**nil**, the error overview will be automatically opened after running TeX if there are errors or warnings to show.

The error overview is opened in a new window of the current frame by default, but you can change this behavior by customizing the option **TeX-error-overview-setup**.

TeX-error-overview-setup [User Option]

Controls the frame setup of the error overview. The possible value is: **separate-frame**; with a **nil** value the current frame is used instead.

The parameters of the separate frame can be set with the **TeX-error-overview-frame-parameters** option.

If the display does not support multi frame, the current frame will be used regardless of the value of this variable.

4.4 Checking for problems

Running TeX or L^ATeX will only find regular errors in the document, not examples of bad style. Furthermore, description of the errors may often be confusing. The utilities **lacheck** and **chktex** can be used to find style errors, such as forgetting to escape the space after an abbreviation or using ‘...’ instead of ‘\ldots’ and other similar problems. You start **lacheck** with *C-c C-c Check* **RET** and **chktex** with *C-c C-c ChkTeX* **RET**. The result will be a list of errors in the ***compilation*** buffer. You can go through the errors with *C-x `* (**next-error**, see Section “Compilation” in *The Emacs Editor*), which will move point to the location of the next error.

Alternatively, you may want in-buffer notation. AUCTeX provides support for this using the Flymake package in Emacs 26 or newer (see Section “Using Flymake” in *GNU Flymake* for details). To enable, call `M-x flymake-mode RET` in the buffer or enable it in all buffers by adding this to your init file:

```
(add-hook 'LaTeX-mode-hook #'flymake-mode)
```

Note that AUCTeX currently only provides support for using `chktex` as the flymake backend. Error messages produced by `chktex` can be controlled by setting the variable `LaTeX-flymake-chktex-options`.

LaTeX-flymake-chktex-options [User Option]

List of strings passed to `chktex` program as additional options. This option can be used to pass a specific warning number to `chktex` like `-w41`.

Each of the two utilities `lacheck` and `chktex` will find some errors the other doesn't, but `chktex` is more configurable, allowing you to create your own errors. You may need to install the programs before using them. You can get `lacheck` from <https://www.ctan.org/pkg/lacheck> and `chktex` from <https://www.ctan.org/pkg/chktex>. TeX Live contains both.

4.5 Controlling the output

A number of commands are available for controlling the output of an application running under AUCTeX

TeX-kill-job [Command]

(`C-c C-k`) Kill currently running external application. This may be either of TeX, LaTeX, previewer, BibTeX, etc.

TeX-recenter-output-buffer [Command]

(`C-c C-l`) Recenter the output buffer so that the bottom line is visible.

TeX-home-buffer [Command]

(`C-c ^`) Go to the ‘master’ file in the document associated with the current buffer, or if already there, to the file where the current process was started.

Additionally, output files produced by AUCTeX can be placed in a separate directory.

TeX-output-dir [User Option]

Set this option to the path of a directory where output files will be placed. The output files include those that are produced by applications running under AUCTeX, temporary files related to region processing and the preview-latex files. If a relative path is specified, it is interpreted as being relative to the master file in a multifile document.

This is a buffer local variable and must be set separately for all documents and all files in a multifile document. For example,

```
%%% Local Variables:
%%% mode: LaTeX
%%% TeX-output-dir: "build"
%%% End:
```

Alternatively, you may use `setq-default` to set the default value of this option or set it as a directory local variable (see Section “Directory Variables” in *The Emacs Editor*).

Note that a non-`nil` value of `TeX-output-dir` might be incompatible with some `TeX` commands and macros. In particular, the `LaTeX` macro ‘`\include`’ is known to not work with this option. Some `TeX` packages which produce intermediary files might also be incompatible. A possible workaround for those packages is to append the value of `TeX-output-dir` to the environment variables `TEXINPUTS` and `BIBINPUTS`.

4.6 Cleaning intermediate and output files

TeX-clean

[Command]

Remove generated intermediate files. In case a prefix argument is given, remove output files as well.

Canonical access to the function is provided by the ‘`Clean`’ and ‘`Clean All`’ entries in `TeX-command-list`, invocable with `C-c C-c` or the Command menu.

The patterns governing which files to remove can be adapted separately for each `AUCTeX` mode by means of the following variables:

- `plain-TeX-clean-intermediate-suffixes`
- `plain-TeX-clean-output-suffixes`
- `LaTeX-clean-intermediate-suffixes`
- `LaTeX-clean-output-suffixes`
- `docTeX-clean-intermediate-suffixes`
- `docTeX-clean-output-suffixes`
- `Texinfo-clean-intermediate-suffixes`
- `Texinfo-clean-output-suffixes`
- `ConTeXt-clean-intermediate-suffixes`
- `ConTeXt-clean-output-suffixes`
- `AmSTeX-clean-intermediate-suffixes`
- `AmSTeX-clean-output-suffixes`

TeX-clean-confirm

[User Option]

Control if deletion of intermediate and output files has to be confirmed before it is actually done. If non-`nil`, ask before deleting files.

4.7 Documentation about macros and packages

TeX-documentation-texdoc

[Command]

(`C-c ?`) Get documentation about the packages installed on your system, using `texdoc` to find the manuals. The function will prompt for the name of packages. If point is on a word, this will be suggested as default.

If the command is called with a prefix argument, you will be shown a list of manuals of the given package among to choose.

The command can be invoked by the key binding mentioned above as well as the ‘Find Documentation...’ entry in the mode menu.

Note that this command assumes T_EX Live (<https://tug.org/texlive/>), not MiK_TE_X (<https://miktex.org/>); according to ‘Texdoc’ site (<https://tug.org/texdoc/>),

A command named `texdoc` is also available in MiK_TE_X, but it is merely a shortcut for an independent program, `mthelp`.

Thus it isn’t sure whether this command works for MiK_TE_X or not.

5 Customization and Extension

5.1 Modes and Hooks

AUCTEX supports a wide variety of derivatives and extensions of T_EX. Besides plain T_EX those are L^AT_EX, AMS-T_EX, ConT_EXt, Texinfo and docT_EX. For each of them there is a separate major mode in AUCTEX and each major mode runs `text-mode-hook`, `TeX-mode-hook` as well as a hook special to the mode in this order. (As an exception, Texinfo mode does not run `TeX-mode-hook`.) The following table provides an overview of the respective mode functions and hooks.

Type	Mode function	Hook
Plain T _E X	<code>plain-TeX-mode</code>	<code>plain-TeX-mode-hook</code>
L ^A T _E X	<code>LaTeX-mode</code>	<code>LaTeX-mode-hook</code>
AMS-T _E X	<code>AmSTeX-mode</code>	<code>AmSTeX-mode-hook</code>
ConT _E Xt	<code>ConTeXt-mode</code>	<code>ConTeXt-mode-hook</code>
Texinfo	<code>Texinfo-mode</code>	<code>Texinfo-mode-hook</code>
DocT _E X	<code>docTeX-mode</code>	<code>docTeX-mode-hook</code>

If you need to make a customization via a hook which is only relevant for one of the modes listed above, put it into the respective mode hook, if it is relevant for any AUCTEX mode, add it to `TeX-mode-hook` and if it is relevant for all text modes, append it to `text-mode-hook`.

Now docT_EX mode is child of L^AT_EX mode, so docT_EX mode runs `LaTeX-mode-hook` as well. Similarly, AmST_EX mode is child of plain T_EX mode and runs `plain-TeX-mode-hook` as well.

Other useful hooks are listed below.

TeX-after-compilation-finished-functions [Variable]

Hook which is run after the T_EX/L^AT_EX processor has successfully finished compiling your document. (See Chapter 4 [Processing], page 50, for finding out how to compile your document.) Each function in the hook is run with the compiled output document as its argument.

This is useful for automatically refreshing the viewer after re-compilation especially when using Emacs viewers such as DocView or PDF Tools. The function `TeX-revert-document-buffer` can be added to the hook for this purpose.

5.2 Multifile Documents

You may wish to spread a document over many files (as you are likely to do if there are multiple authors, or if you have not yet discovered the power of the outline commands (see Section 3.3 [Outline], page 48)). This can be done by having a “master” file in which you include the various files with the T_EX macro ‘`\input`’ or the L^AT_EX macro ‘`\include`’. These files may also include other files themselves. However, to format the document you must run the commands on the top level master file.

When you, for example, ask AUCTEX to run a command on the master file, it has no way of knowing the name of the master file. By default, it will assume that the current file

is the master file. If you insert the following in your init file (`init.el` or `.emacs`), AUCT_EX will use a more advanced algorithm.

```
(setq-default TeX-master nil) ; Query for master file.
```

In this case, AUCT_EX will ask for the name of the master file associated with the buffer. To avoid asking you again, AUCT_EX will automatically insert the name of the master file as a file variable (see Section “File Variables” in *The Emacs Editor*). You can also insert the file variable yourself, by putting the following text at the end of your files.

```
%%% Local Variables:
%%% TeX-master: "master"
%%% End:
```

You should always set this variable to the name of the top level document. If you always use the same name for your top level documents, you can set `TeX-master` in your init file such as `init.el` or `.emacs`.

```
(setq-default TeX-master "master") ; All master files called "master".
```

TeX-master [User Option]

The master file associated with the current buffer. If the file being edited is actually included from another file, then you can tell AUCT_EX the name of the master file by setting this variable. If there are multiple levels of nesting, specify the top level file.

If this variable is `nil`, AUCT_EX will query you for the name.

If the variable is `t`, then AUCT_EX will assume the file is a master file itself.

If the variable is `shared`, then AUCT_EX will query for the name, but will not change the file.

If the variable is `dwim`, AUCT_EX will try to avoid querying by attempting to “do what I mean”; and then change the file.

TeX-one-master [User Option]

Regular expression matching ordinary T_EX files.

You should set this variable to match the name of all files, for which it is a good idea to append a `TeX-master` file variable entry automatically. When AUCT_EX adds the name of the master file as a file variable, it does not need to ask next time you edit the file.

If you dislike AUCT_EX automatically modifying your files, you can set this variable to “<none>”. By default, AUCT_EX will modify any file with an extension of ‘`.tex`’, ‘`.ltx`’, ‘`.texi`’ or ‘`.dtx`’.

TeX-master-file-ask [Command]

(`C-c _`) Query for the name of a master file and add the respective File Variables (see Section “File Variables” in *The Emacs Editor*) to the file for setting this variable permanently.

AUCT_EX will not ask for a master file when it encounters existing files. This function shall give you the possibility to insert the variable manually.

AUCT_EX keeps track of macros, environments, labels, and style files that are used in a given document. For this to work with multifile documents, AUCT_EX has to have a place

to put the information about the files in the document. This is done by having an **auto** subdirectory placed in the directory where your document is located. Each time you save a file, AUCTeX will write information about the file into the **auto** directory. When you load a file, AUCTeX will read the information in the **auto** directory about the file you loaded *and the master file specified by TeX-master*. Since the master file (perhaps indirectly) includes all other files in the document, AUCTeX will get information from all files in the document. This means that you will get from each file, for example, completion for all labels defined anywhere in the document.

AUCTeX will create the **auto** directory automatically if **TeX-auto-save** is non-**nil**. Without it, the files in the document will not know anything about each other, except for the name of the master file. See Section 5.5.3 [Automatic Local], page 79.

TeX-save-document [Command]
(*C-c C-d*) Save all buffers known to belong to the current document.

TeX-save-query [User Option]
If non-**nil**, then query the user before saving each file with **TeX-save-document**.

5.3 Automatic Parsing of TeX Files

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. You are encouraged to enable them by adding the following lines to your init file such as **init.el** or **.emacs**.

```
(setq TeX-parse-self t) ; Enable parse on load.
(setq TeX-auto-save t) ; Enable parse on save.
```

The latter command will make AUCTeX store the parsed information in an **auto** subdirectory in the directory each time the TeX files are stored, see Section 5.5.3 [Automatic Local], page 79. If AUCTeX finds the pre-parsed information when loading a file, it will not need to reparse the buffer. The information in the **auto** directory is also useful for multifile documents, see Section 5.2 [Multifile], page 68, since it allows each file to access the parsed information from all the other files in the document. This is done by first reading the information from the master file, and then recursively the information from each file stored in the master file.

The variables can also be set on a per file basis, by changing the file local variables.

```
%%% Local Variables:
%%% TeX-parse-self: t
%%% TeX-auto-save: t
%%% End:
```

Even when you have disabled the automatic parsing, you can force the generation of style information by pressing *C-c C-n*. This is often the best choice, as you will be able to decide when it is necessary to reparse the file.

TeX-parse-self [User Option]
Parse file after loading it if no style hook is found for it.

TeX-auto-save [User Option]
Automatically save style information when saving the buffer.

TeX-normal-mode *arg* [Command]
 (*C-c C-n*) Remove all information about this buffer, and apply the style hooks again.
 Save buffer first including style information. With optional argument, also reload the style hooks.

When AUCT_EX saves your buffer, it can optionally convert all tabs in your buffer into spaces. Tabs confuse AUCT_EX's error message parsing and so should generally be avoided. However, tabs are significant in some environments, and so by default AUCT_EX does not remove them. To convert tabs to spaces when saving a buffer, insert the following in your init file such as `init.el` or `.emacs`:

```
(setq TeX-auto-untabify t)
```

TeX-auto-untabify [User Option]
 Automatically remove all tabs from a file before saving it.

Instead of disabling the parsing entirely, you can also speed it significantly up by limiting the information it will search for (and store) when parsing the buffer. You can do this by setting the default values for the buffer local variables `TeX-auto-regexp-list` and `TeX-auto-parse-length` in your init file such as `init.el` or `.emacs`.

```
;; Only parse LaTeX class and package information.
(setq-default TeX-auto-regexp-list 'LaTeX-auto-minimal-regexp-list)
;; The class and package information is usually near the beginning.
(setq-default TeX-auto-parse-length 2000)
```

This example will speed the parsing up significantly, but AUCT_EX will no longer be able to provide completion for labels, macros, environments, or bibitems specified in the document, nor will it know what files belong to the document.

These variables can also be specified on a per file basis, by changing the file local variables.

```
%%% Local Variables:
%%% TeX-auto-regexp-list: TeX-auto-full-regexp-list
%%% TeX-auto-parse-length: 999999
%%% End:
```

TeX-auto-regexp-list [User Option]
 List of regular expressions used for parsing the current file.

TeX-auto-parse-length [User Option]
 Maximal length of T_EX file that will be parsed.

The pre-specified lists of regexps are defined below. You can use these before loading AUCT_EX by quoting them, as in the example above.

TeX-auto-empty-regexp-list [Constant]
 Parse nothing

LaTeX-auto-minimal-regexp-list [Constant]
 Only parse L^AT_EX class and packages.

LaTeX-auto-label-regexp-list [Constant]
 Only parse L^AT_EX labels.

<code>LaTeX-auto-index-regexp-list</code>	[Constant]
Only parse \LaTeX index and glossary entries.	
<code>LaTeX-auto-class-regexp-list</code>	[Constant]
Only parse macros in \LaTeX classes and packages.	
<code>LaTeX-auto-pagestyle-regexp-list</code>	[Constant]
Only parse \LaTeX pagestyles.	
<code>LaTeX-auto-counter-regexp-list</code>	[Constant]
Only parse \LaTeX counters.	
<code>LaTeX-auto-length-regexp-list</code>	[Constant]
Only parse \LaTeX lengths.	
<code>LaTeX-auto-savebox-regexp-list</code>	[Constant]
Only parse \LaTeX saveboxes.	
<code>LaTeX-auto-regexp-list</code>	[Constant]
Parse common \LaTeX commands.	
<code>plain-TeX-auto-regexp-list</code>	[Constant]
Parse common plain \TeX commands.	
<code>TeX-auto-full-regexp-list</code>	[Constant]
Parse all \TeX and \LaTeX commands that \AUCTeX can use.	

5.4 Language Support

\TeX and Emacs are usable for European (Latin, Cyrillic, Greek) based languages. Some \LaTeX and Emacs Lisp packages are available for easy typesetting and editing documents in European languages.

All Emacs versions supported by current \AUCTeX can handle CJK (Chinese, Japanese, and Korean) languages by default.

In most cases, special versions of \TeX engines are needed for high-quality typesetting of CJK languages: \CTeX and \ChinaTeX for Chinese, ASCII \pTeX , \upTeX and \NTTjTeX for Japanese, \HLaTeX and \kTeX for Korean. They are necessary as well when you want to typeset documents saved in their domestic encodings such as ‘**Shift-JIS**’. Currently, \AUCTeX offers native support for \pTeX , \upTeX and \jTeX only.

If you don’t need fine tuning in the result with respect to the typesetting rules of their respective national standards, most unicode based \TeX engines, e.g. \LuaTeX and \XeTeX , can handle CJK languages by default if they are encoded in UTF-8. The CJK- \LaTeX package is provided for supporting CJK scripts in a standard \LaTeX document.

5.4.1 Using \AUCTeX with European Languages

5.4.1.1 Typing and Displaying Non-ASCII Characters

First you will need a way to write non-ASCII characters. You can either use macros, or teach \TeX about the ISO character sets. I prefer the latter, it has the advantage that the usual standard emacs word movement and case change commands will work.

Recommended encoding for \LaTeX document is UTF-8. Recent $\text{\LaTeX}2\text{e}$ has native support for UTF-8. If your $\text{\LaTeX}2\text{e}$ is not recent enough, just add `\usepackage[utf8]{inputenc}`.

You can still use ISO 8859 Latin 1 encoding with `\usepackage[latin1]{inputenc}`.

To be able to display non-ASCII characters you will need an appropriate font. All Emacs versions supported by current \LaTeX can display 8-bit characters, provided that suitable fonts are installed.

A compromise is to use an European character set when editing the file, and convert to \TeX macros when reading and writing the files.

`iso-cvt.el`

Much like `iso-tex.el` but is bundled with Emacs 19.23 and later.

X-Symbol a much more complete package for Emacs that can also handle a lot of mathematical characters and input methods.

5.4.1.2 Style Files for Different Languages

\LaTeX supports style files for several languages. Each style file may modify \LaTeX to better support the language, and will run a language specific hook that will allow you to for example change ispell dictionary, or run code to change the keyboard remapping. The following will for example choose a Danish dictionary for documents including `\usepackage[danish]{babel}`. This requires parsing to be enabled, see Section 5.3 [Parsing Files], page 70.

```
(add-hook 'TeX-language-dk-hook
  (lambda () (ispell-change-dictionary "danish")))
```

The following style files are recognized:

brazilian

brazil Runs style hook `TeX-language-pt-br-hook`. Gives `'` word syntax, makes the `"` key inserts `` ``` or `' '` depending on context. Typing `"` twice will insert a literal `'`. Typing `-` twice will insert `"=`, three times `--`.

bulgarian

Runs style hook `TeX-language-bg-hook`. Gives `'` word syntax, makes the `"` key insert a literal `'`. Typing `"` twice will insert `` ``` or `' '` depending on context. Typing `-` twice will insert `"=`, three times `--`.

czech

Runs style hook `TeX-language-cz-hook`. Pressing `"` will insert `\uv{}` and `}` depending on context.

danish

Runs style hook `TeX-language-dk-hook`. Pressing `"` will insert `` ``` and `' '` depending on context. Typing `-` twice will insert `"=`, i.e. a hyphen string allowing hyphenation in the composing words.

dutch

Runs style hook `TeX-language-nl-hook`.

english	
australian	
canadian	
newzealand	Runs style hook <code>TeX-language-en-hook</code> .
frenchb	
francais	Runs style hook <code>TeX-language-fr-hook</code> . Pressing " will insert ‘\og’ and ‘\fg’ depending on context. Note that the language name for customizing <code>TeX-quote-language-alist</code> is ‘french’.
german	
ngerman	Runs style hook <code>TeX-language-de-hook</code> . Gives ‘”’ word syntax, makes the " key insert a literal ‘”’. Pressing the key twice will give you opening or closing German quotes (‘”’ or ‘”’). Typing - twice will insert ‘=’, three times ‘--’.
icelandic	Runs style hook <code>TeX-language-is-hook</code> . Gives ‘”’ word syntax, makes the " key insert a literal ‘”’. Typing " twice will insert ‘”’ or ‘”’ depending on context. Typing - twice will insert ‘=’, three times ‘--’.
italian	Runs style hook <code>TeX-language-it-hook</code> . Pressing " will insert ‘<’ and ‘>’ depending on context.
polish	Runs style hook <code>TeX-language-pl-hook</code> . Gives ‘”’ word syntax and makes the " key insert a literal ‘”’. Pressing " twice will insert ‘”’ or ‘”’ depending on context.
polski	Runs style hook <code>TeX-language-pl-hook</code> . Makes the " key insert a literal ‘”’. Pressing " twice will insert ‘,’ or ‘,’’ depending on context.
portuguese	
portuges	Runs style hook <code>TeX-language-pt-hook</code> . Gives ‘”’ word syntax, makes the " key inserts ‘<’ or ‘>’ depending on context. Typing " twice will insert a literal ‘”’. Typing - twice will insert ‘=’, three times ‘--’. Note that the language name for customizing <code>TeX-quote-language-alist</code> is ‘portuguese’.
slovak	Runs style hook <code>TeX-language-sk-hook</code> . Pressing " will insert ‘\uv{’ and ‘}’ depending on context.
swedish	Runs style hook <code>TeX-language-sv-hook</code> . Pressing " will insert ‘”’. Typing - twice will insert ‘=’, three times ‘--’.

Replacement of language-specific hyphen strings like ‘=’ with dashes does not require to type - three times in a row. You can put point after the hyphen string anytime and trigger the replacement by typing -.

In case you are not satisfied with the suggested behavior of quote and hyphen insertion you can change it by customizing the variables `TeX-quote-language-alist` and `LaTeX-babel-hyphen-language-alist` respectively.

TeX-quote-language-alist [User Option]

Used for overriding the default language-specific quote insertion behavior. This is an alist where each element is a list consisting of four items. The first item is the name

of the language in concern as a string. See the list of supported languages above. The second item is the opening quotation mark. The third item is the closing quotation mark. Opening and closing quotation marks can be specified directly as strings or as functions returning a string. The fourth item is a boolean controlling quote insertion. It should be non-`nil` if the special quotes should only be used after inserting a literal ‘”’ character first, i.e. on second key press.

LaTeX-babel-hyphen-language-alist [User Option]

Used for overriding the behavior of hyphen insertion for specific languages. Every element in this alist is a list of three items. The first item should specify the affected language as a string. The second item denotes the hyphen string to be used as a string. The third item, a boolean, controls the behavior of hyphen insertion and should be non-`nil` if the special hyphen should be inserted after inserting a literal ‘-’ character, i.e. on second key press.

The defaults of hyphen insertion are defined by the variables `LaTeX-babel-hyphen` and `LaTeX-babel-hyphen-after-hyphen` respectively.

LaTeX-babel-hyphen [User Option]

String to be used when typing -. This usually is a hyphen alternative or hyphenation aid provided by ‘babel’ and the related language style files, like ‘=’, ‘~’ or ‘-’.

Set it to an empty string or `nil` in order to disable language-specific hyphen insertion.

LaTeX-babel-hyphen-after-hyphen [User Option]

Control insertion of hyphen strings. If non-`nil` insert normal hyphen on first key press and swap it with the language-specific hyphen string specified in the variable `LaTeX-babel-hyphen` on second key press. If `nil` do it the other way round.

5.4.2 Using AUCT_EX with Japanese T_EX

To write Japanese text with AUCT_EX, you need the versions of T_EX and Emacs that support Japanese. AUCT_EX supports three Japanese T_EX engines by default: NTT jT_EX, ASCII pT_EX and upT_EX.

Activate `japanese-plain-TeX-mode` or `japanese-LaTeX-mode` to use the Japanese T_EX engines. If it doesn’t work, send mail to Masayuki Ataka masayuki.ataka@gmail.com or Ikumi Keita ikumikeita@jcom.home.ne.jp, who currently concern with stuff related to Japanese in AUCT_EX. None of the primary AUCT_EX maintainers understand Japanese, so they cannot help you.

It is recommended to enable `TeX-parse-self` for typical Japanese L^AT_EX users. When enabled, `japanese-LaTeX-mode` selects the suitable Japanese T_EX engine automatically based on the class file name (such as `jbook`, `jsarticle` and `tjreport`) and its option. See Section 5.3 [Parsing Files], page 70.

It is important to select the suitable Japanese T_EX engine because the selected engine determines the command name such as `platex` and `uptex` to typeset the document. If you find that wrong command is used, check the value of `TeX-engine` on that buffer. If the value does not suit the current document, change the value by the ‘TeXing Options’ submenu below the ‘Command’ menu. See Section 4.1.3 [Processor Options], page 55.

To make the selected engine to persist across Emacs sessions, there are two ways from which you can choose one according to your needs:

1. If you use a specific engine (almost) exclusively, customize the option `japanese-TeX-engine-default`.

`japanese-TeX-engine-default` [User Option]
 The default TeX-engine in Japanese T_EX mode.
 The default value is ‘ptex’.

2. If you want to set the engine on a per file basis, use the file local variables to set TeX-engine.

Here is a sample code to set TeX-engine to ‘uptex’:

```
%%% Local Variables:
%%% mode: japanese-LaTeX
%%% TeX-engine: uptex
%%% End:
```

In the both cases above, the valid value is one of ‘ptex’, ‘jtex’ and ‘uptex’.

You can override the command names associated with the above three engines or define your own engine by customizing `TeX-engine-alist`. See Section 4.1.3 [Processor Options], page 55.

It is sometimes necessary to use an engine which differs from the one AUCT_EX selects automatically. For example, even when you want to use `j-article` document class deliberately with ASCII p_LA_TE_X, AUCT_EX selects NTT j_LA_TE_X command if `TeX-parse-self` is enabled, because `j-article` originally belongs to NTT j_LA_TE_X. In such cases, use the file local variable method above to select the engine you intend to use.

If you usually use AUCT_EX in Japanese, setting the following variables is useful.

`TeX-default-mode` [User Option]
 Mode to enter for a new file when it cannot be determined whether the file is plain T_EX or L_AT_EX or what.

If you want to enter Japanese L_AT_EX mode whenever this may happen, set the variable like this:

```
(setq TeX-default-mode 'japanese-LaTeX-mode)
```

`japanese-LaTeX-default-style` [User Option]
 The default style/class when creating a new Japanese L_AT_EX document.
 The default value is “jarticle”.

It is recommended also for Japanese users to customize the option `TeX-PDF-from-DVI` to “Dvipdfmx”. See Section 4.1.3 [Processor Options], page 55.

There are three customize options with regard to the encoding of Japanese text.

`japanese-TeX-use-kanji-opt-flag` [User Option]
 If non-nil, AUCT_EX adds `-kanji` option to the typesetting command when TeX-engine is ‘ptex’.

Usually AUCT_EX guesses the right coding systems for input to and output from the Japanese T_EX process, but you can override them by the following two customize options.

TeX-japanese-process-input-coding-system [User Option]

If non-`nil`, used for encoding input to Japanese T_EX process. When `nil`, AUCT_EX tries to choose suitable coding system.

TeX-japanese-process-output-coding-system [User Option]

If non-`nil`, used for decoding output from Japanese T_EX process. When `nil`, AUCT_EX tries to choose suitable coding system.

The former customize options `japanese-TeX-command-default`, `japanese-LaTeX-command-default` and `japanese-TeX-command-list` are removed from AUCT_EX. Use `japanese-TeX-engine-default` instead. If you need to customize the executable file name such as `"latex"`, the options for them, or both, customize `TeX-engine-alist`.

The following two additional font commands are available in L^AT_EX mode buffer.

C-c C-f g Insert **gothic font** command `'\textgt{★}'` or `'\mathgt{★}'` depending on the context.

C-c C-f m Insert mincho font command `'\textmc{★}'` or `'\mathmc{★}'` depending on the context.

Although they are meaningful only with `'ptex'` and `'uptex'` engines, it won't matter in buffers with other engines.

See `tex-jp.el` for more information.

5.5 Automatic Customization

Since AUCT_EX is so highly customizable, it makes sense that it is able to customize itself. The automatic customization consists of scanning T_EX files and extracting symbols, environments, and things like that.

The automatic customization is done on three different levels. The global level is the level shared by all users at your site, and consists of scanning the standard T_EX style files, and any extra styles added locally for all users on the site. The private level deals with those style files you have written for your own use, and use in different documents. You may have a `~/lib/TeX/` directory where you store useful style files for your own use. The local level is for a specific directory, and deals with writing customization for the files for your normal T_EX documents.

If compared with the environment variable `TEXINPUTS`, the global level corresponds to the directories built into T_EX. The private level corresponds to the directories you add yourself, except for `.`, which is the local level.

By default AUCT_EX will search for customization files in all the global, private, and local style directories, but you can also set the path directly. This is useful if you for example want to add another person's style hooks to your path. Please note that all matching files found in `TeX-style-path` are loaded, and all hooks defined in the files will be executed.

TeX-style-path [User Option]

List of directories to search for AUCT_EX style files.

By default, when AUCTeX searches a directory for files, it will recursively search through subdirectories.

TeX-file-recurse [User Option]

Whether to search TeX directories recursively: `nil` means do not recurse, a positive integer means go that far deep in the directory hierarchy, `t` means recurse indefinitely.

By default, AUCTeX will ignore files named `.`, `..`, `SCCS`, `RCS`, and `CVS`.

TeX-ignore-file [User Option]

Regular expression matching file names to ignore.

These files or directories will not be considered when searching for TeX files in a directory.

5.5.1 Automatic Customization for the Site

Assuming that the automatic customization at the global level was done when AUCTeX was installed, your choice is now: will you use it? If you use it, you will benefit by having access to all the symbols and environments available for completion purposes. The drawback is slower load time when you edit a new file and perhaps too many confusing symbols when you try to do a completion.

You can disable the automatic generated global style hooks by setting the variable `TeX-auto-global` to `nil`.

TeX-macro-global [User Option]

Directories containing the site's TeX style files.

TeX-style-global [User Option]

Directory containing hand generated TeX information.

These correspond to TeX macros shared by all users of a site.

TeX-auto-global [User Option]

Directory containing automatically generated information.

For storing automatic extracted information about the TeX macros shared by all users of a site.

5.5.2 Automatic Customization for a User

You should specify where you store your private TeX macros, so AUCTeX can extract their information. The extracted information will go to the directories listed in `TeX-auto-private`

Use *M-x TeX-auto-generate* RET to extract the information.

TeX-macro-private [User Option]

Directories where you store your personal TeX macros. The value defaults to the directories listed in the `TEXINPUTS` and `BIBINPUTS` environment variables or to the respective directories in `$TEXMFHOME` of `kpsewhich` setting if no results can be obtained from the environment variables.

TeX-auto-private [User Option]
 List of directories containing automatically generated AUCTeX style files. These correspond to the personal T_EX macros.

TeX-auto-generate *tex auto* [Command]
 (*M-x TeX-auto-generate* RET) Generate style hook for *tex* and store it in *auto*. If *tex* is a directory, generate style hooks for all files in the directory.

TeX-style-private [User Option]
 List of directories containing hand generated AUCTeX style files. These correspond to the personal T_EX macros.

5.5.3 Automatic Customization for a Directory

AUCTeX can update the style information about a file each time you save it if **TeX-auto-save** option is enabled. Saved information will be stored in the directory **TeX-auto-local**, set to "auto" by default.

The advantage of doing this is that macros, labels, etc. defined in any file in a multifile document will be known in all the files in the document. The disadvantage is that saving will be slower. To disable, set **TeX-auto-local** to **nil**.

TeX-style-local [User Option]
 Directory containing hand generated T_EX information.
 These correspond to T_EX macros found in the current directory.

TeX-auto-local [User Option]
 Directory containing automatically generated T_EX information.
 These correspond to T_EX macros found in the current directory.

TeX-auto-save-aggregate [User Option]
 When non-**nil**, save parsed information in **auto** subdirectory of master directory.
 Otherwise, save in each **auto** subdirectory of the parsed file.
 Subdirectory name is actually taken from **TeX-auto-local**.

5.6 Writing Your Own Style Support

See Section 5.5 [Automatic], page 77, for a discussion about automatically generated global, private, and local style files. The hand generated style files are equivalent, except that they by default are found in **style** directories instead of **auto** directories.

If you write some useful support for a public T_EX style file, please send it to us.

5.6.1 A Simple Style File

Here is a simple example of a style file.

```
;;; book.el - Special code for book style.

(TeX-add-style-hook
 "book"
 (lambda ()
```

```
(LaTeX-largest-level-set "part"))
TeX-dialect)
```

The example is from the AUCT_{EX} sources and is loaded for any L_AT_EX document using the book document class (or style before L_AT_EX2_ε). (Note that the above code is much simplified for explanatory purpose.) The file specifies that the largest kind of section in such a document is ‘**part**’. The interesting thing to notice is that the style file defines an (anonymous) function, and adds it to the list of loaded style hooks by calling `TeX-add-style-hook`.

The first time the user indirectly tries to access some style-specific information, such as the largest sectioning command available, the style hooks for all files directly or indirectly read by the current document are executed. The actual files will only be evaluated once, but the hooks will be called for each buffer using the style file.

Note that the basename of the style file and the name of the style hook should usually be identical.

TeX-add-style-hook *style hook* &optional *dialect-expr* [Function]

Add *hook* to the list of functions to run when we use the T_EX file *style* and the current dialect is one in the set derived from *dialect-expr*. When *dialect-expr* is omitted, then *hook* is allowed to be run whatever the current dialect is.

dialect-expr may be one of:

- A symbol indicating a singleton containing one basic T_EX dialect, this symbol shall be selected among:

`:latex` For all files in L_AT_EX mode, or any mode derived thereof.

`:bibtex` For all files in BibT_EX mode, or any mode derived thereof.

`:texinfo` For all files in Texinfo mode.

`:plain-tex`

For all files in plain-T_EX mode, or any mode derived thereof.

`:context` For all files in ConT_EXt mode.

`:classopt`

For class options of L_AT_EX document. This is provided as pseudo-dialect for style hooks associated with class options.

- A logical expression like:

(or *dialect-expression1* ... *dialect-expression_n*)

For union of the sets of dialects corresponding to *dialect-expression1* through *dialect-expression_n*

(and *dialect-expression1* ... *dialect-expression_n*)

For intersection of the sets of dialects corresponding to *dialect-expression1* through *dialect-expression_n*

(nor *dialect-expression1* ... *dialect-expression_n*)

For complement of the union sets of dialects corresponding to *dialect-expression1* through *dialect-expression_n* relatively to the set of all supported dialects

(not *dialect-expr*)

For complement set of dialect corresponding to *dialect-expr* relatively to the set of all supported dialects

In case of adding a style hook for L^AT_EX, when calling function `TeX-add-style-hook` it is thought more futureproof for argument *dialect-expr* to pass constant `TeX-dialect` currently defined to `:latex`, rather than passing `:latex` directly.

`TeX-dialect`

[Constant]

Default dialect for use with function `TeX-add-style-hook` for argument *dialect-expr* when the hook is to be run only on L^AT_EX file, or any mode derived thereof.

5.6.2 Adding Support for Macros

The most common thing to define in a style hook is new symbols (T_EX macros). Most likely along with a description of the arguments to the function, since the symbol itself can be defined automatically.

Here are a few examples from `latex.el`.

```
(TeX-add-style-hook
 "latex"
 (lambda ()
  (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)
   '("newtheorem" TeX-arg-define-environment
     [ TeX-arg-environment "Numbered like" ]
     t [ TeX-arg-counter "Within counter" ]))))
```

`TeX-add-symbols` *symbol* ...

[Function]

Add each *symbol* to the list of known symbols.

Each argument to `TeX-add-symbols` is a list describing one symbol. The head of the list is the name of the symbol, the remaining elements describe each argument.

If there are no additional elements, the symbol will be inserted with point inside braces. Otherwise, each argument of this function should match an argument of the T_EX macro. What is done depends on the argument type.

If a macro is defined multiple times, AUCT_EX will choose the one with the longest definition (i.e. the one with the most arguments).

Thus, to overwrite

```
'("tref" 1) ; one argument
```

you can specify

```
'("tref" TeX-arg-ref ignore) ; two arguments
```

`ignore` is a function that does not do anything, so when you insert a ‘`tref`’ you will be prompted for a label and no more.

You can use the following types of specifiers for arguments:

string Use the string as a prompt to prompt for the argument.

number	Insert that many braces, leave point inside the first. 0 and -1 are special. 0 means that no braces are inserted. -1 means that braces are inserted around the macro and an active region (e.g. ‘ <code>\tiny foo</code> ’). If there is no active region, no braces are inserted.
nil	Insert empty braces.
t	Insert empty braces, leave point between the braces.
other symbols	Call the symbol as a function. You can define your own hook, or use one of the predefined argument hooks.
list	If the car is a string, insert it as a prompt and the next element as initial input. Otherwise, call the car of the list with the remaining elements as arguments.
vector	Optional argument. If it has more than one element, parse it as a list, otherwise parse the only element as above. Use square brackets instead of curly braces, and is not inserted on empty user input.

A lot of argument hooks have already been defined. The first argument to all hooks is a flag indicating if it is an optional argument. It is up to the hook to determine what to do with the remaining arguments, if any. Typically the next argument is used to overwrite the default prompt.

TeX-arg-conditional

Implements if *expr then else*. If *expr* evaluates to true, parse *then* as an argument list, else parse *else* as an argument list.

TeX-arg-literal

Insert its arguments into the buffer. Used for specifying extra syntax for a macro.

TeX-arg-free

Parse its arguments but use no braces when they are inserted.

TeX-arg-eval

Evaluate arguments and insert the result in the buffer.

TeX-arg-label

Prompt for a label completing with known labels. If RefTeX is active, prompt for the reference format.

TeX-arg-ref

Prompt for a label completing with known labels. If RefTeX is active, do not prompt for the reference format. Usually, reference macros should use this function instead of **TeX-arg-label**.

TeX-arg-index-tag

Prompt for an index tag. This is the name of an index, not the entry.

TeX-arg-index

Prompt for an index entry completing with known entries.

TeX-arg-length

Prompt for a L^AT_EX length completing with known lengths.

TeX-arg-macro

Prompt for a \TeX macro with completion.

TeX-arg-date

Prompt for a date, defaulting to the current date. The format of the date is specified by the **TeX-date-format** option. If you want to change the format when the ‘**babel**’ package is loaded with a specific language, set **TeX-date-format** inside the appropriate language hook (for details see Section 5.4.1 [European], page 72).

TeX-arg-version

Prompt for the version of a file, using as initial input the current date.

TeX-arg-environment

Prompt for a \LaTeX environment with completion.

TeX-arg-cite

Prompt for a \BibTeX citation. If the variable **TeX-arg-cite-note-p** is non-`nil`, ask also for optional note in citations.

TeX-arg-counter

Prompt for a \LaTeX counter completing with known counters.

TeX-arg-savebox

Prompt for a \LaTeX savebox completing with known saveboxes.

TeX-arg-file

Prompt for a filename in the current directory, and use it with the extension.

TeX-arg-file-name

Prompt for a filename and use as initial input the name of the file being visited in the current buffer, with extension.

TeX-arg-file-name-sans-extension

Prompt for a filename and use as initial input the name of the file being visited in the current buffer, without extension.

TeX-arg-input-file

Prompt for the name of an input file in \TeX ’s search path, and use it without the extension. Run the style hooks for the file. (Note that the behavior (type of prompt and inserted file name) of the function can be controlled by the variable **TeX-arg-input-file-search**.)

TeX-arg-define-label

Prompt for a label completing with known labels. Add label to list of defined labels.

TeX-arg-define-length

Prompt for a \LaTeX length completing with known lengths. Add length to list of defined lengths.

TeX-arg-define-macro

Prompt for a \TeX macro with completion. Add macro to list of defined macros.

TeX-arg-define-environment

Prompt for a \LaTeX environment with completion. Add environment to list of defined environments.

TeX-arg-define-cite

Prompt for a \BibTeX citation.

TeX-arg-define-counter

Prompt for a \LaTeX counter.

TeX-arg-define-savebox

Prompt for a \LaTeX savebox.

TeX-arg-document

Prompt for a \LaTeX document class, using `LaTeX-default-style` as default value and `LaTeX-default-options` as default list of options. If the variable `TeX-arg-input-file-search` is `t`, you will be able to complete with all \LaTeX classes available on your system, otherwise classes listed in the variable `LaTeX-style-list` will be used for completion. It is also provided completion for options of many common classes.

LaTeX-arg-usepackage

Prompt for \LaTeX packages. If the variable `TeX-arg-input-file-search` is `t`, you will be able to complete with all \LaTeX packages available on your system. It is also provided completion for options of many common packages.

TeX-arg-bibstyle

Prompt for a \BibTeX style file completing with all style available on your system.

TeX-arg-bibliography

Prompt for \BibTeX database files completing with all databases available on your system.

TeX-arg-corner

Prompt for a \LaTeX side or corner position with completion.

TeX-arg-lr

Prompt for a \LaTeX side with completion.

TeX-arg-tb

Prompt for a \LaTeX side with completion.

TeX-arg-pagestyle

Prompt for a \LaTeX pagestyle with completion.

TeX-arg-verb

Prompt for delimiter and text.

TeX-arg-verb-delim-or-brace

Prompt for delimiter and text. This function is similar to `TeX-arg-verb`, but is intended for macros which take their argument enclosed in delimiters or in braces.

TeX-arg-pair

Insert a pair of numbers, use arguments for prompt. The numbers are surrounded by parentheses and separated with a comma.

TeX-arg-size

Insert width and height as a pair. No arguments.

TeX-arg-coordinate

Insert x and y coordinates as a pair. No arguments.

LaTeX-arg-author

Prompt for document author, using **LaTeX-default-author** as initial input.

TeX-read-hook

Prompt for a \LaTeX hook and return it.

TeX-arg-hook

Prompt for a \LaTeX hook and insert it as a \TeX macro argument.

TeX-read-key-val

Prompt for a ‘key=value’ list of options and return them.

TeX-arg-key-val

Prompt for a ‘key=value’ list of options and insert it as a \TeX macro argument.

TeX-arg-space

Insert one or more spaces.

TeX-arg-set-exit-mark

Set **TeX-exit-mark** to current point or a given position.

If you add new hooks, you can assume that point is placed directly after the previous argument, or after the macro name if this is the first argument. Please leave point located after the argument you are inserting. If you want point to be located somewhere else after all hooks have been processed, set the value of **TeX-exit-mark**. It will point nowhere, until the argument hook sets it.

Some packages provide macros that are rarely useful to non-expert users. Those should be marked as expert macros using **TeX-declare-expert-macros**.

TeX-declare-expert-macros *style macros*...

[Function]

Declare *macros* as expert macros of *style*.

Expert macros are completed depending on **TeX-complete-expert-commands**.

5.6.3 Adding Support for Environments

Adding support for environments is very much like adding support for \TeX macros, except that each environment normally only takes one argument, an environment hook. The example is again a short version of **latex.el**.

```
(TeX-add-style-hook
 "latex"
 (lambda ()
  (LaTeX-add-environments
   '("document" LaTeX-env-document)))
```

```
'("enumerate" LaTeX-env-item)
'("itemize" LaTeX-env-item)
'("list" LaTeX-env-list)))))
```

It is completely up to the environment hook to insert the environment, but the function `LaTeX-insert-environment` may be of some help. The hook will be called with the name of the environment as its first argument, and extra arguments can be provided by adding them to a list after the hook.

For simple environments with arguments, for example defined with ‘`\newenvironment`’, you can make AUCTEX prompt for the arguments by giving the prompt strings in the call to `LaTeX-add-environments`. The fact that an argument is optional can be indicated by wrapping the prompt string in a vector.

For example, if you have defined a `loop` environment with the three arguments *from*, *to*, and *step*, you can add support for them in a style file.

```
%% loop.sty

\newenvironment{loop}[3]{...}{...}
;; loop.el

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

If an environment is defined multiple times, AUCTEX will choose the one with the longest definition. Thus, if you have an `enumerate` style file, and want it to replace the standard L^AT_EX `enumerate` hook above, you could define an `enumerate.el` file as follows, and place it in the appropriate style directory.

```
(TeX-add-style-hook
 "latex"
 (lambda ()
  (LaTeX-add-environments
   '("enumerate" LaTeX-env-enumerate foo))))

(defun LaTeX-env-enumerate (environment &optional _ignore) ...)
```

The symbol `foo` will be passed to `LaTeX-env-enumerate` as the second argument, but since we only added it to overwrite the definition in `latex.el` it is just ignored.

`LaTeX-add-environments env ...` [Function]

Add each *env* to list of loaded environments.

`LaTeX-insert-environment env [extra]` [Function]

Insert environment of type *env*, with optional argument *extra*.

Following is a list of available hooks for `LaTeX-add-environments`:

`LaTeX-env-item`

Insert the given environment and the first item.

LaTeX-env-item-args

Insert the given environment plus further arguments, and the first item. You can use this as a hook in case you want to specify multiple complex arguments just like in elements of `TeX-add-symbols`. Here is an example from `enumitem.el` in order to prompt for a ‘key=value’ list to be inserted as an optional argument to the ‘itemize’ environment:

```
(LaTeX-add-environments
  ("itemize" LaTeX-env-item-args
    [TeX-arg-key-val (LaTeX-enumitem-key-val-options)]))
```

LaTeX-env-figure

Insert the given figure-like environment with a caption and a label.

LaTeX-env-array

Insert the given array-like environment with position and column specifications.

LaTeX-env-label

Insert the given environment with a label.

LaTeX-env-label-args

Insert the given environment with a label and further arguments to the environment.

LaTeX-env-list

Insert the given list-like environment, a specifier for the label and the first item.

LaTeX-env-minipage

Insert the given minipage-like environment with position and width specifications.

LaTeX-env-tabular*

Insert the given tabular*-like environment with width, position and column specifications.

LaTeX-env-picture

Insert the given environment with width and height specifications.

LaTeX-env-bib

Insert the given environment with a label for a bibitem.

LaTeX-env-args

Insert the given environment with arguments. You can use this as a hook in case you want to specify multiple complex arguments just like in elements of `TeX-add-symbols`. This is most useful if the specification of arguments to be prompted for with strings and strings wrapped in a vector as described above is too limited.

Here is an example from `listings.el` which calls a function with one argument in order to prompt for a ‘key=value’ list to be inserted as an optional argument of the ‘lstlisting’ environment:

```
(LaTeX-add-environments
  ("lstlisting" LaTeX-env-args
    [TeX-arg-key-val (LaTeX-listings-key-val-options)]))
```

Some packages provide environments that are rarely useful to non-expert users. Those should be marked as expert environments using `LaTeX-declare-expert-environments`.

`LaTeX-declare-expert-environments` *style environments...* [Function]

Declare *environments* as expert environments of *style*.

Expert environments are completed depending on `TeX-complete-expert-commands`.

5.6.4 Adding or Examining Other Information

5.6.4.1 Adding bibliographies in style hooks

You can also specify bibliographical databases and labels in the style file. This is probably of little use, since this information will usually be automatically generated from the `TeX` file anyway.

`LaTeX-add-bibliographies` *bibliography ...* [Function]

Add each *bibliography* to list of loaded bibliographies.

`LaTeX-add-labels` *label ...* [Function]

Add each *label* to the list of known labels.

5.6.4.2 Examining Package/Class Options

In `LaTeX` documents, style hooks can find the package names and those options given as optional argument(s) of ‘`\usepackage`’ in `LaTeX-provided-package-options`.

`LaTeX-provided-package-options` [Variable]

Buffer local variable holding alist of options provided to `LaTeX` packages. Each element is a cons cell (*package . option-list*). For example, its value will be

```
((("babel" . ("german"))
  ("geometry" . ("a4paper" "top=2cm" "left=2.5cm" "right=2.5cm"))
  ...)
```

You can examine whether there is a specific package-option pair by `LaTeX-provided-package-options-member`.

`LaTeX-provided-package-options-member` *package option* [Function]

Return non-`nil` if *option* has been given to *package*. The value is actually the tail of the list of options given to *package*.

There are similar facilities for class names and those options given in `\documentclass` declaration.

`LaTeX-provided-class-options` [Variable]

Buffer local variable holding alist of options provided to `LaTeX` classes. Each element is a cons cell (*class . option-list*). For example, its value will be

```
((("book" . ("a4paper" "11pt" "openany" "fleqn"))
  ...)
```

`LaTeX-provided-class-options-member` *class option* [Function]

Return non-`nil` if *option* has been given to *class*. The value is actually the tail of the list of options given to *class*.

LaTeX-match-class-option *regex* [Function]
 Check if a documentclass option matching *regex* is active. Return first found class option matching *regex*, or `nil` if not found.

These functions are also useful to implement customized predicate(s) in `TeX-view-predicate-list`. See Section 4.2.1 [Starting Viewers], page 58.

5.6.4.3 Adding Support for Option Completion

When the user inserts ‘`\usepackage`’ by *C-c C-m*, AUCTeX asks for the optional arguments after the package name is given. The style file of that package can provide completion support for the optional arguments.

LaTeX-package-name-package-options [Variable]
 List of optional arguments available for the package.

Here is an excerpt from ‘`acronym.el`’:

```
(defvar LaTeX-acronym-package-options
  '("footnote" "nohyperlinks" "printonlyused" "withpage"
    "smaller" "dua" "nolist")
  "Package options for the acronym package.")
```

When the package accepts key-value style optional arguments, more sophisticated completion support is needed. The package style file can provide dynamic completion support by custom elisp function.

LaTeX-package-name-package-options [Function]
 This function should ask the user for optional arguments and return them as a string, instead of built-in option query facility. When this function is defined, AUCTeX calls it with no argument.

Here is an excerpt from ‘`acro.el`’:

```
(defun LaTeX-acro-package-options ()
  "Prompt for package options for the acro package."
  (TeX-read-key-val t LaTeX-acro-package-options-list))
```

As you can see in the above example, a utility function `TeX-read-key-val` is available to read key-value pair(s) from users.

Note that `defvar` or `defun` of `LaTeX-package-name-package-options` should be at the top level of the style file and not inside the style hook, because the style hook is not yet called when the user inputs the optional arguments in response to *C-c C-m*.

There are similar facilities for class options. When the user inserts ‘`\documentclass`’ by *C-c C-e*, the respective class style file can provide completion support for the optional arguments.

LaTeX-classname-class-options [Variable]
 List of optional arguments available for the class.

LaTeX-classname-class-options [Function]
 Which see.

In case the utility function `TeX-read-key-val` is used, the variable containing the key-value pair(s) should be called `LaTeX-packagename-package-options-list` or `LaTeX-classname-class-options-list` accordingly.

5.6.5 Automatic Extraction of New Things

The automatic `TeX` information extractor works by searching for regular expressions in the `TeX` files, and storing the matched information. You can add support for new constructs to the parser, something that is needed when you add new commands to define symbols.

For example, in the file `macro.tex` I define the following macro.

```
\newcommand{\newmacro}[5]{%
  \def#1{#3\index{#4@#5~cite{#4}}\nocite{#4}}%
  \def#2{#5\index{#4@#5~cite{#4}}\nocite{#4}}%
}
```

`AUCTEX` will automatically figure out that ‘`newmacro`’ is a macro that takes five arguments. However, it is not smart enough to automatically see that each time we use the macro, two new macros are defined. We can specify this information in a style hook file.

```
;;; macro.el --- Special code for my own macro file.

;;; Code:

(defvar TeX-newmacro-regexp
  '("\\\\newmacro{\\\\\\\\([a-zA-Z]+\\\\)}{\\\\\\\\\\\\\\\\([a-zA-Z]+\\\\)}"
    (1 2) TeX-auto-multi)
  "Matches \\newmacro definitions.")

(defvar TeX-auto-multi nil
  "Temporary for parsing \\newmacro definitions.")

(defun TeX-macro-cleanup ()
  "Move symbols from `TeX-auto-multi' to `TeX-auto-symbol'."
  (mapc (lambda (list)
    (mapc (lambda (symbol)
      (setq TeX-auto-symbol
        (cons symbol TeX-auto-symbol)))
      list))
    TeX-auto-multi))

(defun TeX-macro-prepare ()
  "Clear `TeX-auto-multi' before use."
  (setq TeX-auto-multi nil))

(add-hook 'TeX-auto-prepare-hook #'TeX-macro-prepare)
(add-hook 'TeX-auto-cleanup-hook #'TeX-macro-cleanup)

(TeX-add-style-hook
 "macro"
```

```

(lambda ()
  (TeX-auto-add-regexp TeX-newmacro-regexp)
  (TeX-add-symbols '("newmacro"
                     TeX-arg-macro
                     (TeX-arg-macro "Capitalized macro: \\")
                     t
                     "BibTeX entry: "
                     nil))))

;;; macro.el ends here

```

When this file is first loaded, it adds a new entry to `TeX-newmacro-regexp`, and defines a function to be called before the parsing starts, and one to be called after the parsing is done. It also declares a variable to contain the data collected during parsing. Finally, it adds a style hook which describes the ‘`newmacro`’ macro, as we have seen it before.

So the general strategy is: Add a new entry to `TeX-newmacro-regexp`. Declare a variable to contain intermediate data during parsing. Add hook to be called before and after parsing. In this case, the hook before parsing just initializes the variable, and the hook after parsing collects the data from the variable, and adds them to the list of symbols found.

TeX-auto-regexp-list [Variable]

List of regular expressions matching `TeX` macro definitions.

The list has the following format `((regexp match table) . . .)`, that is, each entry is a list with three elements.

regexp. Regular expression matching the macro we want to parse.

match. A number or list of numbers, each representing one parenthesized subexpression matched by *regexp*.

table. The symbol table to store the data. This can be a function, in which case the function is called with the argument *match*. Use `TeX-match-buffer` to get match data. If it is not a function, it is presumed to be the name of a variable containing a list of match data. The matched data (a string if *match* is a number, a list of strings if *match* is a list of numbers) is put in front of the table.

TeX-auto-prepare-hook *nil* [Variable]

List of functions to be called before parsing a `TeX` file.

TeX-auto-cleanup-hook *nil* [Variable]

List of functions to be called after parsing a `TeX` file.

Appendix A Copying, Changes, Development, FAQ, Texinfo Mode

A.1 Copying this Manual

The full license text can be read here:

A.1.1 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a

textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you

follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that

copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

A.2 Changes and New Features

Recent changes are listed at ELPA AUCT_EX page (<https://elpa.gnu.org/packages/auctex.html>).

Older Versions

See the file `history.texi` for older changes. It is available at AUCT_EX Git repository.

List of Contributors

The following people have contributed to AUCT_EX (in alphabetical order): Ivan Andrus, Ralf Angeli, Masayuki Ataka, Mohammad Hossein Bateni, Fabrice Ben Hamouda, Thomas Baumann, Vincent Belaïche, Berend de Boer, Alex Branham, Uwe Brauer, Ken Brown, Joshua Buhl, Jean-François Burnol, Patrice Dumas, Arash Esbati, Vangelis Evangelou, Werner Fink, Miguel Frasson, David Fussner, Peter S. Galbraith, Mosè Giordano, Andrea Greselin, Patrick Gundlach, Abdul-Lateef Haji-Ali, Jobst Hoffmann, Tassilo Horn, Yvon Hevel, Orlando Iovino, Mads Jensen, Arne Jørgensen, Philip Kaludercic, David Kastrup, Ikumi Keita, Philip Kime, Oleh Krehel, Joost Kremers, Frank Küster, Jan-Åke Larsson, Matthew Leach, Brian Leung, Antoine Levitt, Leo Liu, Vladimir Lomov, Stefan Monnier, Paul Nelson, Dan Nicolaescu, Piet van Oostrum, Pieter Pareit, Nicolas Richard, Florent Rougon, Santiago Saavedra, Davide G. M. Salvetti, Rüdiger Sonderfeld, Holger Sparr, Mike Sperber, Reiner Steib, Christian Schlauer, Augusto Ritter Stoffel, Shiro Takeda, Mark Tretin, Artem Yurchenko, Tony Zorman (Please accept our apologies if we forgot somebody).

A.3 Future Development

The following sections describe future development of AUCT_EX. Besides mid-term goals, bug reports and requests we cannot fix or honor right away are being gathered here. If you

have some time for Emacs Lisp hacking, you are encouraged to try to provide a solution to one of the following problems. If you don't know Lisp, you may help us to improve the documentation. It might be a good idea to discuss proposed changes on the mailing list of AUCTEX first.

A.3.1 Mid-term Goals

- Integration of preview-latex into AUCTEX

As of AUCTEX 11.81 preview-latex is a part of AUCTEX in the sense that the installation routines were merged and preview-latex is being packaged with AUCTEX.

Further integration will happen at the backend. This involves folding of error parsing and task management of both packages which will ease development efforts and avoid redundant work.

- Error help catalogs

Currently, the help for errors is more or less hardwired into `latex.el`. For supporting error help in other languages, it would be sensible to instead arrange error messages in language-specific files, make a common info file from all such catalogs in a given language and look the error texts up in an appropriate index. The user would then specify a preference list of languages, and the errors would be looked up in the catalogs in sequence until they were identified.

- Combining 'docTeX' with RefTeX

Macro cross references should also be usable for document navigation using RefTeX.

- Fix remove-style feature

Currently `TeX-remove-style` implementation isn't good. It is common practice that major mode functions directly add macros and environments via `TeX-add-symbols` and `LaTeX-add-environments`, but those macros and environments are lost once `TeX-remove-style` runs. It is necessary to run major mode function, by e.g. `normal-mode`, again to recover them, but that makes no point in running `TeX-remove-style` itself because major mode function kills all buffer-local variables.

As of AUCTEX 12.3, `TeX-remove-style` is no longer used by any other codes.

- Factor out syntax propertization from `font-latex.el`

Syntax propertization is implemented in `font-latex.el`. This means that features which depend on syntax parse don't work well for `tex-font.el` users and those who disable font lock.

Hence syntax propertization should be factored out from `font-latex.el` and implemented as a major mode facility.

(Texinfo mode is an exception because it already has its own syntax propertize function, which just copies the one available in Emacs built-in texinfo mode.)

- Add documentation

Following entries should be included in the document:

- Variable `TeX-translate-location-hook`
- How to use `TeX-auto-add-type`, as well as functions and variables generated by that macro. They should be covered in the node Section 5.6.5 [Hacking the Parser], page 90.
- Usage of ConTeXt mode

A.3.2 Wishlist

- Enable syntactic font lock for verbatim constructs in ‘docTeX’ mode
In ‘docTeX’ mode buffer, ‘%’ sign at the line beginning hinders syntactic font lock of verbatim constructs. This should be improved.
- Simplify tool bar implementation
The library `toolbar-x.el` was developed as an abstraction layer to absorb difference between XEmacs and GNU Emacs. Now that XEmacs is no longer supported, the library, together with `tex-bar.el` as a whole, can be much simplified (or even unified).
- Documentation lookup for macros
A parser could gather information about which macros are defined in which L^AT_EX packages and store the information in a hashtable which can be used in a backend for TeX-doc in order to open the matching documentation for a given macro. The information could also be used to insert an appropriate ‘\usepackage’ statement if the user tries to insert a macro for which the respective package has not been requested yet.
- Improvements to error reporting
Fringe indicators for errors in the main text would be nice.
- A math entry grid
A separate frame with a table of math character graphics to click on in order to insert the respective sequence into the buffer (cf. the “grid” of ‘x-symbol’).
- Crossreferencing support
It would be nice if you could index process your favorite collection of .dtx files (such as the L^AT_EX source), just call a command on arbitrary control sequence, and get either the DVI viewer opened right at the definition of that macro (using Source Specials), or the source code of the .dtx file.
- Better plain T_EX support
For starters, LaTeX-math-mode is not very L^AT_EX-specific in the first place, and similar holds for indentation and formatting.
- Page count when compiling should (optionally) go to modeline of the window where the compilation command was invoked, instead of the output window. Suggested by Karsten Tinnfeld `tinnfeld@irb.informatik.uni-dortmund.de`.
- Command to insert a macrodefinition in the preamble, without moving point from the current location. Suggested by "Jeffrey C. Ely" `ely@nwu.edu`.
- A database of all commands defined in all stylefiles. When a command or environment gets entered that is provided in one of the styles, insert the appropriate \usepackage in the preamble.
- A way to add and overwrite math mode entries in style files, and to decide where they should be. Suggested by Remo Badii `Remo.Badii@psi.ch`.
- Create template for (first) line of tabular environment.
- I think prompting for the master is the intended behavior. It corresponds to a ‘shared’ value for TeX-master.
There should probably be a ‘none’ value which wouldn’t query for the master, but instead disable all features that relies on TeX-master.

This default value for `TeX-master` could then be controlled with mapping based on the extension.

- Use index files (when available) to speed up `C-c C-m include` RET.
- Option not to calculate very slow completions like for `C-c C-m include` RET.
- Font menu should be created from `TeX-font-list`.
- Installation procedure written purely in emacs lisp.
- Included PostScript files should also be counted as part of the document.
- A nice hierarchical by-topic organization of all officially documented `LATEX` macros, available from the menu bar.
- `TeX-command-default` should be set from the master file, if not set locally. Suggested by Peter Whaite `peta@cim.mcgill.ca`.
- Make `AUCTEX` work with ‘crypt++’. Suggested by Chris Moore `Chris.Moore@src.bae.co.uk`.
- Make `AUCTEX` work with ‘longlines’. This would also apply to `preview-latex`, though it might make sense to unify error processing before attempting this.
- The ‘Spell’ command should apply to all files in a document. Maybe it could try to restrict to files that have been modified since last spell check? Suggested by Ravinder Bhumbra `rbhumbra@ucsd.edu`.
- Make `.` check for abbreviations and sentences ending with capital letters.
- Use Emacs 19 minibuffer history to choose between previewers, and other stuff. Suggested by John Interrante `interran@uluru.Stanford.EDU`.
- Documentation of variables that can be set in a style hook.

We need a list of what can safely be done in an ordinary style hook. You can not set a variable that `AUCTEX` depends on, unless `AUCTEX` knows that it has to run the style hooks first.

Here is the start of such a list.

```
LaTeX-add-environments
TeX-add-symbols
LaTeX-add-labels
LaTeX-add-bibliographies
LaTeX-largest-level
```

- Outline should be (better) supported in `TEX` mode.

At least, support headers, trailers, as well as `TeX-outline-extra`.

- `TeX-header-start` and `TeX-trailer-end`.

We might want these, just for fun (and outlines)

- Plain `TEX` and `LATEX` specific header and trailer expressions.

We should have a way to globally specify the default value of the header and trailer regexps.

- Get closer to original `TeX-mode` keybindings.

A third initialization file (`tex-mode.el`) containing an emulator of the standard `TeX-mode` would help convince some people to change to `AUCTEX`.

- Use markers in `TeX-error-list` to remember buffer positions in order to be more robust with regard to line numbers and changed files.
- Finish the Texinfo mode. For one thing, many Texinfo mode commands do not accept braces around their arguments.
- Hook up the letter environment with `bbdb.el`.

A.3.3 Bugs

- The style hooks automatically generated by parsing files for `example.dtx`, `example.sty`, `example.drv` and `example.bib` all clash. Bad. Clash with hand-written style hooks should be removed by dialect discrimination — to be checked.
- `C-c `` should always stay in the current window, also when it finds a new file.
- Do not overwrite emacs warnings about existing auto-save files when loading a new file.
- Maybe the regexp for matching a \TeX symbol during parsing should be `"\\[a-zA-Z]+\\|\\.\\|\\|"` — `thiemann@informatik.uni-tuebingen.de` Peter Thiemann.
- AUCT \TeX should not parse verbatim environments.
- Make `‘`’` check for math context in `LaTeX-math-mode`. and simply self insert if not in a math context.
- Make `TeX-insert-dollar` more robust. Currently it can be fooled by `‘\mbox’`es and escaped double dollar for example.
- Correct indentation for tabbing, table, and math environments.

A.4 Frequently Asked Questions

1. Something is not working correctly. What should I do?

If you are having trouble with upgrading ELPA package, first have a look at the dedicated entry below. If that doesn't resolve your issue, then come back here and proceed. Well, you might have guessed it, the first place to look is in the available documentation packaged with AUCT \TeX . This could be the `README` file or `NEWS.org` file in case you are experiencing problems after an upgrade, the section about bugs in the manual in case you encountered a bug or the relevant sections in the manual for other related problems.

If this did not help, you can send a bug report to the AUCT \TeX bug reporting list by using the command `M-x TeX-submit-bug-report RET`. But before you do this, you can try to get more information about the problem at hand which might also help you locate the cause of the error yourself.

First, you can try to generate a so-called backtrace which shows the functions involved in a program error. In order to do this, start Emacs with the command line `‘emacs --debug-init’` and/or put the line

```
(setq debug-on-error t)
```

as the first line into your init file. After Emacs has started, you can load a file which triggers the error and a new window should pop up showing the backtrace. If you get such a backtrace, please include it in the bug report.

Second, you can try to figure out if something in your personal or site configuration triggers the error by starting Emacs without such customizations. You can do this by invoking Emacs with the following command line:

```
emacs -q -no-site-file --eval "(progn (setq package-load-list '((auctex
t))) (package-initialize))"
```

Use ‘runemacs’ instead of ‘emacs’ on MS Windows.

The `--eval` option activates only AUCTEX among all installed ELPA packages.

After you have started Emacs like this, you can load the file triggering the error. If everything is working now, you know that you have to search either in the site configuration file or your personal init file for statements related to the problem.

2. ELPA upgrade fails. What should I do?

In general, ELPA upgrade can fail in a Emacs session you are running, especially when AUCTEX major version increases. This entry covers the following cases in such casual upgrade.

- Upgrade stops with error and doesn’t complete.
- Upgrade falls in infinite loop and doesn’t terminate. You have to interrupt it with `C-g` and upgrade doesn’t complete.
- It looks like upgrade finishes successfully but afterwards AUCTEX breaks with mysterious error like:

```
TeX-command-expand: Wrong type argument: stringp, nil
```

These issues involve byte compilation failure. In the third case, the generated `.elc` files are actually corrupted. Thus the point is to have sane byte compilation.

The first thing which is worth trying is:

1. Terminate the current running Emacs session and restart Emacs.
2. Type `M-x package-recompile RET auctex RET`.
3. Restart your Emacs again.

If you are lucky enough, this will basically resolve the issue. There is still old ‘auctex-X.Y.Z’ directory left behind under `~/.emacs.d/elpa/`, so delete it manually to avoid future trouble.

If the above prescription doesn’t work, then try:

1. Uninstall AUCTEX once.
2. Restart your Emacs and (before doing anything else) reinstall AUCTEX.

In theory, this recipe will circumvent all caveats in ELPA upgrade of AUCTEX. Find and delete old ‘auctex-X.Y.Z’ directory remaining under `~/.emacs.d/elpa/`.

3. What versions of Emacs are supported?

AUCTEX was tested with GNU Emacs 28.1. Older versions may work but are unsupported.

4. Why doesn’t the completion, style file, or multifile stuff work?

It must be enabled first, insert this in your init file:

```
(setq-default TeX-master nil)
(setq TeX-parse-self t)
```

```
(setq TeX-auto-save t)
```

Read also the chapters about parsing and multifile documents in the manual. See Section 5.3 [Parsing Files], page 70, and Section 5.2 [Multifile], page 68.

5. Why doesn't `TeX-save-document` work?

`TeX-check-path` has to contain `"/"` somewhere.

6. Why is the information in `foo.tex` forgotten when I save `foo.bib`?

For various reasons, `AUCTEX` ignores the extension when it stores information about a file, so you should use unique base names for your files. E.g. rename `foo.bib` to `foob.bib`.

7. Why doesn't `AUCTEX` signal when processing a document is done?

If the message in the minibuffer stays "Type 'C-c C-l' to display results of compilation.", you probably have a misconfiguration in your init file (`.emacs`, `init.el` or similar). To track this down either search in the `*Messages*` buffer for an error message or put `(setq debug-on-error t)` as the first line into your init file, restart Emacs and open a `LaTeX` file. Emacs will complain loudly by opening a debugging buffer as soon as an error occurs. The information in the debugging buffer can help you find the cause of the error in your init file.

8. Why does `TeX-next-error (C-c `)` fail?

If `TeX-file-line-error` is set to `nil` (not the default), these sort of failures might be related to the fact that when writing the log file, `TeX` puts information related to a file, including error messages, between a pair of parentheses. In this scenario `AUCTEX` determines the file where the error happened by parsing the log file and counting the parentheses. This can fail when there are other, unbalanced parentheses present.

Activating so-called `'file:line:error'` messages for the log file usually solves this issue, as these kind of messages are easier to parse; however, they may lack some details. Activation can be done either in the configuration of your `TeX` system (consult its manual to see where this is) or by simply keeping the variable `TeX-file-line-error` to the default value of `non-nil`.

9. What does `'AUC'` stand for?

`AUCTEX` came into being at Aalborg University in Denmark. Back then the Danish name of the university was Aalborg Universitetscenter; `'AUC'` for short.

A.5 Features specific to `AUCTEX`'s Texinfo major mode

`AUCTEX` includes a major mode for editing Texinfo files. This major mode is not the same mode as the native Texinfo mode (see Section "Texinfo Mode" in *Texinfo*) of Emacs, although they have the same name. However, `AUCTEX` still relies on a number of functions from the native Texinfo mode.

The following text describes which functionality is offered by `AUCTEX` and which by the native Texinfo mode. This should enable you to decide when to consult the `AUCTEX` manual and when the manual of the native mode. And in case you are a seasoned user of the native mode, the information should help you to swiftly get to know the `AUCTEX`-specific commands.

A.5.1 How AUCT_EX and the native mode work together

In a nutshell the split between AUCT_EX Texinfo mode, and native Texinfo mode is as follows:

- Most of the editing (environment creation, commenting, font command insertions) and/or processing commands (e.g. compiling or printing) which are available in other AUCT_EX modes are also handled by AUCT_EX in Texinfo mode.
- Texinfo-related features (e.g. info node linkage or menu creation) rely on the commands provided by the native Texinfo mode. AUCT_EX provides the key bindings to reach these functions, keeping the same keys as in native Texinfo whenever possible, or similar ones otherwise.

A.5.2 Where the native mode is superseded

This section is directed to users of the native Texinfo mode switching to AUCT_EX. It follows the summary of the native mode (see Section “Texinfo Mode Summary” in *Texinfo*) and lists which of its commands are no longer of use.

Insert commands

In the native Texinfo mode, frequently used Texinfo commands can be inserted with key bindings of the form `C-c C-c k` where *k* differs for each Texinfo command; *c* inserts `@code`, *d* inserts `@dfn`, *k* `@kbd`, etc.

In AUCT_EX commands are inserted with the key binding `C-c C-m` instead which prompts for the macro to be inserted. For font selection commands (like `@b`, `@i`, or `@emph`) and a few related ones (like `@var`, `@key` or `@code`) there are bindings which insert the respective macros directly. They have the form `C-c C-f k` or `C-c C-f C-k` and call the function `TeX-font`. Type `C-c C-f RET` to get a list of supported commands.

Note that the prefix argument is not handled the same way by AUCT_EX.

Insert braces

In AUCT_EX braces can be inserted with the same key binding as in the native Texinfo mode: `C-c {`. But AUCT_EX uses its own function for the feature: `TeX-insert-braces`.

Insert environments

The native Texinfo mode does not insert full environments. Instead, it provides the function `texinfo-insert-@end` (mapped to `C-c C-c e`) for closing an open environment with a matching `@end` statement.

In AUCT_EX you can insert full environments, i.e. both the opening and closing statements, with the function `Texinfo-environment` (mapped to `C-c C-e`).

Insert nodes

Node insertion command `texinfo-insert-@node` is available in the native Texinfo mode (mapped to `C-c C-c n`). It only inserts the string ‘`@node`’ (with suitable newlines).

AUCT_EX provides its own node insertion command `Texinfo-insert-node` (mapped to `C-c C-s`), which asks for the next, previous and upper nodes with completion, in addition to the node name you are going to insert.

Format info files with `makeinfo` and `TeX`

In the native Texinfo mode there are various functions and bindings to format a region or the whole buffer for info or to typeset the respective text. For example, there is `makeinfo-buffer` (mapped to `C-c C-m C-b`) which runs `makeinfo` on the buffer or there is `texinfo-tex-buffer` (mapped to `C-c C-t C-b`) which runs `TeX` on the buffer in order to produce a DVI file.

In AUCTeX different commands for formatting or typesetting can be invoked through the function `TeX-command-master` (mapped to `C-c C-c`). After typing `C-c C-c`, you can select the desired command, e.g. ‘Makeinfo’, ‘Makeinfo HTML’, ‘Texi2dvi’ or ‘TeX’, through a prompt in the mini buffer. Note that you can make, say ‘Makeinfo’, the default by adding this statement in your init file:

```
(add-hook 'Texinfo-mode-hook
  (lambda () (setq TeX-command-default "Makeinfo")))
```

Note also that `C-c C-c Makeinfo RET` is not completely functionally equivalent to `makeinfo-buffer` as the latter will display the resulting info file in Emacs, showing the node corresponding to the position in the source file, just after a successful compilation. This is why, while using AUCTeX, invoking `makeinfo-buffer` might still be more convenient.

Note also that in the case of a multifile document, `C-c C-c` in AUCTeX will work on the whole document (provided that the file variable `TeX-master` is set correctly), while `makeinfo-buffer` in the native mode will process only the current buffer, provided that the `@setfilename` statement is provided.

Produce indexes and print

The native Texinfo mode provides the binding `C-c C-t C-i` (`texinfo-texindex`) for producing an index and the bindings `C-c C-t C-p` (`texinfo-tex-print`) and `C-c C-t C-q` (`tex-show-print-queue`) for printing and showing the printer queue. These are superseded by the respective commands available through `C-c C-c` (`TeX-command-master`) in AUCTeX: ‘Texindex’, ‘Print’, and ‘Queue’.

Kill jobs The command `C-c C-t C-k` (`tex-kill-job`) in the native mode is superseded by `C-c C-k` (`TeX-kill-job`) in AUCTeX.

A.5.3 Where key bindings are mapped to the native mode

This node follows the native Texinfo mode summary (see Section “Texinfo Mode Summary” in *Texinfo*) and lists only those commands to which AUCTeX provides a keybinding.

Basically all commands of the native mode related to producing menus and interlinking nodes are mapped to same or similar keys in AUCTeX, while a few insertion commands are mapped to AUCTeX-like keys.

@item insertion

The binding `C-c C-c i` for the insertion of `@item` in the native mode is mapped to `M-RET` or `C-c C-j` in AUCTeX, similar to other AUCTeX modes.

@end insertion

The binding `C-c C-c e` for closing a `@foo` command by a corresponding `@end foo` statement in the native mode is mapped to `C-c J` in AUCTeX, similar to other AUCTeX modes.

Move out of balanced braces

The binding `C-c }` (`up-list`) is available both in the native mode and in AUCTEX. (This is because the command is not implemented in either mode but a native Emacs command.) However, in AUCTEX, you cannot use `C-c]` for this, as it is used for `@end` insertion.

Update pointers

The bindings `C-c C-u C-n` (`texinfo-update-node`) and `C-c C-u C-e` (`texinfo-every-node-update`) from the native mode are available in AUCTEX as well.

Update menus

The bindings `C-c C-u m` (`texinfo-master-menu`), `C-c C-u C-m` (`texinfo-make-menu`), and `C-c C-u C-a` (`texinfo-all-menus-update`) from the native mode are available in AUCTEX as well. The command `texinfo-start-menu-description`, bound to `C-c C-c C-d` in the native mode, is bound to `C-c C-u C-d` in AUCTEX instead.

A.5.4 Which native mode key bindings are missing

The following command from the native commands might still be useful when working with AUCTEX, however, it is not accessible with a key binding any longer.

Show the section structure

The command `texinfo-show-structure` (`C-c C-s`) from the native mode does not have a key binding in AUCTEX. The binding is used by AUCTEX for inserting `@node`.

Indices

Key Index

"		
"	12
\$		
\$	13
(
(.....	14
[
[.....	14
^		
^	23
-		
-	23
{		
{	14
C		
C-c %	29
C-c *	28, 29
C-c	28, 29
C-c ;	29
C-c ?	66
C-c]	20
C-c ^	65
C-c _	69
C-c `	62
C-c {	14
C-c ~	23
C-c C-a	51
C-c C-b	50
C-c C-c	50
C-c C-d	70
C-c C-e	19, 106
C-c C-f	16
C-c C-f C-b	9, 15
C-c C-f C-c	9, 16
C-c C-f C-e	9, 15
C-c C-f C-f	9, 16
C-c C-f C-i	9, 15
C-c C-f C-l	16
C-c C-f C-m	15
C-c C-f C-n	16
C-c C-f C-r	9, 16
C-c C-f C-s	9, 15
C-c C-f C-t	9, 16
C-c C-f C-w	16
C-c C-f g	77
C-c C-f m	77
C-c C-k	65, 107
C-c C-l	65
C-c C-m	27
C-c C-n	71
C-c C-o b	45
C-c C-o C-b	43
C-c C-o C-c	45
C-c C-o C-e	44
C-c C-o C-f	43
C-c C-o C-m	44
C-c C-o C-o	45
C-c C-o C-p	44
C-c C-o C-r	44
C-c C-o C-s	44
C-c C-o i	45
C-c C-o p	45
C-c C-o r	45
C-c C-o s	45
C-c C-q C-e	34
C-c C-q C-p	34
C-c C-q C-r	34
C-c C-q C-s	34
C-c C-r	50
C-c C-s	17, 106
C-c C-t C-b	63
C-c C-t C-i	10, 55
C-c C-t C-o	10
C-c C-t C-p	10, 55
C-c C-t C-r	51
C-c C-t C-s	10, 55
C-c C-t C-w	63
C-c C-t C-x	64
C-c C-v	59
C-c C-z	51
C-c LFD	22
C-j	31
C-M-a	20
C-M-e	20
C-M-h	29
C-x n e	49
C-x n g	49

L

LFD 31

M

M-g p 63
 M-q 34
 M-RET 22
 M-TAB 25

Function Index**A**

align-current 30
 AmSTeX-mode 68
 auto-fill-mode 33

C

ConTeXt-mode 68

D

docTeX-mode 68

F

font-latex-setup 36

I

indent-region 30

J

japanese-LaTeX-mode 75
 japanese-plain-TeX-mode 75

L

LaTeX--arguments-completion-at-point 26
 LaTeX-add-bibliographies 88
 LaTeX-add-environments 86
 LaTeX-add-labels 88
 LaTeX-arg-author 85
 LaTeX-arg-usepackage 84
 LaTeX-classname-class-options 89
 LaTeX-close-environment 20
 LaTeX-command-section 51
 LaTeX-declare-expert-environments 88
 LaTeX-env-args 87
 LaTeX-env-array 87
 LaTeX-env-bib 87
 LaTeX-env-figure 87
 LaTeX-env-item 86
 LaTeX-env-item-args 87

T

TAB 31

LaTeX-env-label 87
 LaTeX-env-label-args 87
 LaTeX-env-list 87
 LaTeX-env-minipage 87
 LaTeX-env-picture 87
 LaTeX-env-tabular* 87
 LaTeX-environment 19
 LaTeX-fill-environment 34
 LaTeX-fill-paragraph 34
 LaTeX-fill-region 34
 LaTeX-fill-section 34
 LaTeX-find-matching-begin 20
 LaTeX-find-matching-end 20
 LaTeX-indent-line 31
 LaTeX-insert-environment 86
 LaTeX-insert-item 22
 LaTeX-make-inline 14
 LaTeX-mark-environment 28
 LaTeX-mark-section 28
 LaTeX-match-class-option 89
 LaTeX-math-mode 23
 LaTeX-mode 68
 LaTeX-modify-math 24
 LaTeX-narrow-to-environment 49
 LaTeX-packagename-package-options 89
 LaTeX-provided-class-options-member 88
 LaTeX-provided-package-options-member 88
 LaTeX-section 17
 LaTeX-section-heading 17
 LaTeX-section-label 18
 LaTeX-section-section 18
 LaTeX-section-title 18
 LaTeX-section-toc 18

P

plain-TeX-mode 68

T

tex-font-setup	36	TeX-comment-or-uncomment-region	29
TeX--completion-at-point	26	TeX-complete-symbol	25
TeX-add-style-hook	80	TeX-declare-expert-macros	85
TeX-add-symbols	81	TeX-documentation-texdoc	66
TeX-arg-bibliography	84	TeX-electric-macro	27
TeX-arg-bibstyle	84	TeX-error-overview	64
TeX-arg-cite	83	TeX-fold-buffer	43
TeX-arg-conditional	82	TeX-fold-clearout-buffer	45
TeX-arg-coordinate	85	TeX-fold-clearout-item	45
TeX-arg-corner	84	TeX-fold-clearout-paragraph	45
TeX-arg-counter	83	TeX-fold-clearout-region	45
TeX-arg-date	83	TeX-fold-clearout-section	45
TeX-arg-define-cite	84	TeX-fold-comment	45
TeX-arg-define-counter	84	TeX-fold-dwim	45
TeX-arg-define-environment	84	TeX-fold-env	44
TeX-arg-define-label	83	TeX-fold-macro	44
TeX-arg-define-length	83	TeX-fold-math	44
TeX-arg-define-macro	83	TeX-fold-mode	43
TeX-arg-define-savebox	84	TeX-fold-paragraph	44
TeX-arg-document	84	TeX-fold-region	44
TeX-arg-environment	83	TeX-fold-section	44
TeX-arg-eval	82	TeX-font	16
TeX-arg-file	83	TeX-home-buffer	65
TeX-arg-file-name	83	TeX-insert-braces	14, 106
TeX-arg-file-name-sans-extension	83	TeX-insert-dollar	13
TeX-arg-free	82	TeX-insert-macro	27
TeX-arg-hook	85	TeX-insert-quote	12
TeX-arg-index	82	TeX-interactive-mode	55
TeX-arg-index-tag	82	TeX-ispell-skip-setcar	54
TeX-arg-input-file	83	TeX-ispell-skip-setcdr	54
TeX-arg-key-val	85	TeX-ispell-tex-arg-end	54
TeX-arg-label	82	TeX-kill-job	65, 107
TeX-arg-length	82	TeX-master-file-ask	69
TeX-arg-literal	82	TeX-narrow-to-group	49
TeX-arg-lr	84	TeX-next-error	62
TeX-arg-macro	83	TeX-normal-mode	71
TeX-arg-pagestyle	84	TeX-PDF-mode	55
TeX-arg-pair	85	TeX-pin-region	51
TeX-arg-ref	82	TeX-previous-error	63
TeX-arg-savebox	83	TeX-read-hook	85
TeX-arg-set-exit-mark	85	TeX-read-key-val	85
TeX-arg-size	85	TeX-recenter-output-buffer	65
TeX-arg-space	85	TeX-revert-document-buffer	68
TeX-arg-tb	84	TeX-save-document	70
TeX-arg-verb	84	TeX-source-correlate-mode	55, 60
TeX-arg-verb-delim-or-brace	84	TeX-toggle-debug-bad-boxes	63
TeX-arg-version	83	TeX-toggle-debug-warnings	63
TeX-auto-add-regexp	90	TeX-toggle-suppress-ignored-warnings	64
TeX-auto-generate	79	TeX-view	59, 61
TeX-clean	66	TeX-view-mouse	61
TeX-command-buffer	50	Texinfo-environment	106
TeX-command-master	50	Texinfo-insert-node	106
TeX-command-region	50	Texinfo-mark-environment	29
TeX-command-run-all	51	Texinfo-mark-node	29
TeX-comment-or-uncomment-paragraph	29	Texinfo-mark-section	29
		Texinfo-mode	68
		turn-on-auto-fill	33

Variable Index

A

AmSTeX-clean-intermediate-suffixes	66
AmSTeX-clean-output-suffixes	66
AmSTeX-mode-hook	68

C

ConTeXt-clean-intermediate-suffixes	66
ConTeXt-clean-output-suffixes	66
ConTeXt-engine	57
ConTeXt-Mark-version	58
ConTeXt-mode-hook	68
ConTeXt-Omega-engine	57

D

docTeX-clean-intermediate-suffixes	66
docTeX-clean-output-suffixes	66
docTeX-indent-across-comments	33
docTeX-mode-hook	68

F

fill-column	33
font-latex-deactivated-keyword-classes ...	40
font-latex-fontify-script	41
font-latex-fontify-script-max-level	41
font-latex-fontify-sectioning	38
font-latex-match-bold-command-keywords ...	39
font-latex-match-bold-declaration-keywords	39
font-latex-match-function-keywords	38
font-latex-match-italic-command-keywords .	39
font-latex-match-italic-declaration-keywords	39
font-latex-match-math-command-keywords .	39, 41
font-latex-match-reference-keywords	38
font-latex-match-sectioning-0-keywords ...	38
font-latex-match-sectioning-1-keywords ...	38
font-latex-match-sectioning-2-keywords ...	38
font-latex-match-sectioning-3-keywords ...	38
font-latex-match-sectioning-4-keywords ...	38
font-latex-match-sectioning-5-keywords ...	38
font-latex-match-slide-title-keywords	38
font-latex-match-textual-keywords	38
font-latex-match-type-command-keywords ...	39
font-latex-match-type-declaration-keywords	39
font-latex-match-underline-command-keywords	39
font-latex-match-variable-keywords	38
font-latex-match-warning-keywords	38
font-latex-math-environments	41
font-latex-script-char-face	42
font-latex-script-display	42
font-latex-sectioning-0-face	38
font-latex-sectioning-1-face	38

font-latex-sectioning-2-face	38
font-latex-sectioning-3-face	38
font-latex-sectioning-4-face	38
font-latex-sectioning-5-face	38
font-latex-slide-title-face	38
font-latex-subscript-face	41
font-latex-superscript-face	41
font-latex-user-keyword-classes	40

J

japanese-LaTeX-default-style	76
japanese-TeX-engine-default	76
japanese-TeX-mode	75
japanese-TeX-use-kanji-opt-flag	76

L

LaTeX-amsmath-label	20
LaTeX-auto-class-regexp-list	72
LaTeX-auto-counter-regexp-list	72
LaTeX-auto-index-regexp-list	72
LaTeX-auto-label-regexp-list	71
LaTeX-auto-length-regexp-list	72
LaTeX-auto-minimal-regexp-list	71
LaTeX-auto-pagestyle-regexp-list	72
LaTeX-auto-regexp-list	72
LaTeX-auto-savebox-regexp-list	72
LaTeX-babel-hyphen	75
LaTeX-babel-hyphen-after-hyphen	75
LaTeX-babel-hyphen-language-alist	75
LaTeX-begin-regexp	31
LaTeX-biblatex-use-Biber	52
LaTeX-classname-class-options	89
LaTeX-clean-intermediate-suffixes	66
LaTeX-clean-output-suffixes	66
LaTeX-command	57
LaTeX-csquotes-close-quote	12
LaTeX-csquotes-open-quote	12
LaTeX-csquotes-quote-after-quote	12
LaTeX-default-author	85
LaTeX-default-document-environment	19
LaTeX-default-environment	19
LaTeX-default-format	22
LaTeX-default-options	84
LaTeX-default-position	22
LaTeX-default-style	84
LaTeX-default-width	22
LaTeX-done-mark	17
LaTeX-electric-left-right-brace	15
LaTeX-enable-toolbar	50
LaTeX-end-regexp	31
LaTeX-eqnarray-label	20
LaTeX-equation-label	20
LaTeX-figure-label	21
LaTeX-fill-break-at-separators	34
LaTeX-fill-break-before-code-comments	34
LaTeX-fill-excluded-macros	35

LaTeX-float	21
LaTeX-flymake-chktex-options	65
LaTeX-fold-env-spec-list	46
LaTeX-fold-macro-spec-list	46
LaTeX-fold-math-spec-list	46
LaTeX-font-list	16
LaTeX-indent-environment-check	30
LaTeX-indent-environment-list	30, 31
LaTeX-indent-level	30, 31
LaTeX-insert-into-comments	30
LaTeX-item-indent	30, 31
LaTeX-item-regexp	30
LaTeX-label-alist	19
LaTeX-label-annotation-max-length	26
LaTeX-level	17
LaTeX-listing-label	21
LaTeX-math-abbrev-prefix	23
LaTeX-math-default	23
LaTeX-math-list	23
LaTeX-math-menu-unicode	23
LaTeX-mode-hook	68
LaTeX-name	17
LaTeX-Omega-command	57
LaTeX-packagename-package-options	89
LaTeX-paragraph-commands	33
LaTeX-provided-class-options	88
LaTeX-provided-package-options	88
LaTeX-section-hook	17
LaTeX-section-label	17, 18
LaTeX-short-caption-prompt-length	21
LaTeX-style-list	84
LaTeX-syntactic-comments	31
LaTeX-table-label	21
LaTeX-title	17
LaTeX-toc	17
LaTeX-top-caption-list	21
LaTeX-verbatim-environments	42
LaTeX-verbatim-macros-with-braces	42
LaTeX-verbatim-macros-with-delims	42

O

outline-regexp	29
----------------------	----

P

plain-TeX-auto-regexp-list	72
plain-TeX-clean-intermediate-suffixes	66
plain-TeX-clean-output-suffixes	66
plain-TeX-enable-toolbar	50
plain-TeX-mode-hook	68

T

TeX-after-compilation-finished-functions ..	68
TeX-arg-cite-note-p	83
TeX-arg-input-file-search	83, 84
TeX-arg-item-label-p	22
TeX-arg-right-insert-p	14
TeX-auto-cleanup-hook	91
TeX-auto-empty-regexp-list	71
TeX-auto-full-regexp-list	72
TeX-auto-global	78
TeX-auto-local	79
TeX-auto-parse-length	71
TeX-auto-prepare-hook	91
TeX-auto-private	79
TeX-auto-regexp-list	71, 91
TeX-auto-save	70
TeX-auto-save-aggregate	79
TeX-auto-untabify	71
TeX-bar-Latex-button-alist	50
TeX-bar-Latex-buttons	50
TeX-bar-TeX-all-button-alists	50
TeX-bar-TeX-buttons	50
TeX-brace-indent-level	31
TeX-check-engine	58
TeX-check-path	53
TeX-check-TeX	57
TeX-check-TeX-command-not-found	57
TeX-clean-confirm	66
TeX-close-quote	12
TeX-command	57
TeX-command-default	52
TeX-command-extra-options	58
TeX-command-list	50, 52
TeX-complete-expert-commands	19, 28
TeX-complete-list	26
TeX-date-format	83
TeX-debug-bad-boxes	63
TeX-debug-warnings	63
TeX-default-macro	27
TeX-default-mode	75, 76
TeX-dialect	81
TeX-display-help	63
TeX-DVI-via-PDFTeX	55
TeX-electric-escape	27
TeX-electric-math	13
TeX-electric-sub-and-superscript	24
TeX-engine	56, 75
TeX-engine-alist	57, 75
TeX-engine-alist-builtin	57
TeX-error-overview-frame-parameters	64
TeX-error-overview-open-after-TeX-run	64
TeX-error-overview-setup	64
TeX-expand-list	52
TeX-file-line-error	58
TeX-file-recurse	78
TeX-fold-alert-color	47
TeX-fold-auto	44
TeX-fold-auto-reveal	47

TeX-fold-auto-reveal-commands	47	TeX-language-sk-hook	73
TeX-fold-begin-end-spec-list	47	TeX-language-sv-hook	73
TeX-fold-bib-file	48	TeX-LaTeX-sentinel-banner-regexp	57
TeX-fold-close-quote	48	TeX-macro-global	7, 78
TeX-fold-command-prefix	45	TeX-macro-private	78
TeX-fold-env-spec-list	46	TeX-master	50, 69
TeX-fold-force-fontify	43	TeX-math-input-method-off-regexp	24
TeX-fold-help-echo-max-length	46	TeX-mode-hook	68
TeX-fold-macro-spec-list	45	TeX-modes	5
TeX-fold-math-spec-list	46	TeX-newline-function	30, 31
TeX-fold-open-quote	48	TeX-Omega-command	57
TeX-fold-preserve-comments	44	TeX-one-master	69
TeX-fold-quotes-on-insert	48	TeX-open-quote	12
TeX-fold-region-functions	44	TeX-outline-extra	48
TeX-fold-type-list	43	TeX-output-dir	65
TeX-fold-unfold-around-mark	44	TeX-parse-all-errors	63
TeX-fold-unspec-env-display-string	46	TeX-parse-self	70, 75
TeX-fold-unspec-macro-display-string	46	TeX-PDF-from-DVI	56
TeX-fold-unspec-use-name	46	TeX-PDF-mode	55
TeX-font-list	16	TeX-quote-after-quote	12
TeX-header-end	50, 51	TeX-quote-language-alist	74
TeX-ignore-file	78	TeX-raise-frame-function	62
TeX-ignore-warnings	63	TeX-refuse-unmatched-dollar	13
TeX-indent-close-delimiters	32	TeX-region	50, 51
TeX-indent-open-delimiters	32	TeX-save-query	70
TeX-insert-braces	27	TeX-show-compilation	58
TeX-insert-braces-alist	27	TeX-source-correlate-map	61
TeX-insert-macro-default-style	27	TeX-source-correlate-method	55, 60
TeX-install-font-lock	36	TeX-source-correlate-mode	55
TeX-interactive-mode	55	TeX-source-correlate-start-server	61
TeX-ispell-extend-skip-list	53	TeX-style-global	78
TeX-ispell-verb-delimiters	55	TeX-style-local	79
TeX-japanese-process-input-coding-system ..	77	TeX-style-path	77
TeX-japanese-process-output-coding-system ..	77	TeX-style-private	79
TeX-kill-process-without-query	51	TeX-suppress-ignored-warnings	64
TeX-language-bg-hook	73	TeX-trailer-start	50, 51
TeX-language-cz-hook	73	TeX-view-evince-keep-focus	60
TeX-language-de-hook	73	TeX-view-predicate-list	59
TeX-language-dk-hook	73	TeX-view-program-list	59
TeX-language-en-hook	73	TeX-view-program-selection	59
TeX-language-is-hook	73	Texinfo-clean-intermediate-suffixes	66
TeX-language-it-hook	73	Texinfo-clean-output-suffixes	66
TeX-language-nl-hook	73	Texinfo-mode-hook	68
TeX-language-pl-hook	73	texinfo-section-list	29
TeX-language-pt-br-hook	73	texmathp-tex-commands	41
TeX-language-pt-hook	73	texmathp-tex-commands-default	41

Concept Index

<code>\</code>	
<code>'\begin'</code>	18
<code>\chapter</code>	8, 16
<code>\cite</code> , completion of	28
<code>\emph</code>	9, 15
<code>'\end'</code>	18
<code>\include</code>	68
<code>\input</code>	68
<code>\item</code>	22
<code>\label</code>	8, 16
<code>\label</code> , completion	28
<code>\mathgt</code>	77
<code>\mathmc</code>	77
<code>\ref</code> , completion	28
<code>\section</code>	8, 16
<code>\subsection</code>	8, 16
<code>\textbf</code>	9, 15
<code>\textgt</code>	77
<code>\textit</code>	9, 15
<code>\textmc</code>	77
<code>\textmd</code>	15
<code>\textnormal</code>	16
<code>\textrm</code>	9, 16
<code>\textsc</code>	9, 16
<code>\textsf</code>	9, 16
<code>\textsl</code>	9, 15
<code>\textsw</code>	16
<code>\texttt</code>	9, 16
<code>\textulc</code>	16

A

Abbreviations	23
Adding a style hook	79
Adding bibliographies	88
Adding environments	85
Adding labels	88
Adding macros	81
Adding other information	88
Adding support for completion of package/class options	88
<code>align.el</code>	30
<code>amsmath</code>	20, 22
Arguments to \TeX macros	25
ASCII \TeX	72, 75
<code>auto</code> directories	77
<code>auto-fill-mode</code>	30
Auto-Reveal	43
Automatic	77
Automatic Customization	77
Automatic Parsing	70
Automatic updating style hooks	79

B

Bad boxes	62
Biber	52
<code>biblatex</code>	52
Bibliographies, adding	88
Bibliography	50
bibliography, completion	28
<code>BibTeX</code>	50
<code>BibTeX</code> , completion	28
<code>book.el</code>	79
Braces	12
Brackets	12
Brazilian Portuguese	73
Bulgarian	73

C

Changing font	15
Changing the parser	90
Chapters	8, 16
Checking	64
<code>ChinaTeX</code>	72
<code>chktex</code>	64
citations, completion of	28
<code>cite</code> , completion of	28
CJK language	72
<code>CJK-LAT\TeX</code>	72
Cleaning	66
Commands	50
Completion	25
Controlling the output	65
Copying	2
Copyright	2
<code>CT\TeX</code>	72
Current file	65
Customization	7
Customization, personal	7
Customization, site	7
Czech	73

D

Danish	73
Debugging	62
Default command	50
Defining bibliographies in style hooks	88
Defining environments in style hooks	85
Defining labels in style hooks	88
Defining macros in style hooks	81
Defining other information in style hooks	88
Deleting fonts	9, 16
Descriptions	22
Display math mode	12
Display math, converting	24
Distribution	2
Documentation	66
Documents	68
Documents with multiple files	68

Dollar signs, color bleed with	42
Dollars	12
Double quotes	12
Dutch	73

E

English	73
Enumerates	22
Environments	18
Environments, adding	85
Eqnarray	20
Equation	20
Equations	20
Errors	62
Europe	72
European Characters	72
Examining package/class options	88
Example of a style file	79
Expansion	25
External Commands	50
Extracting T _E X symbols	77

F

Faces	42
FDL, GNU Free Documentation License	92
Figure environment	21
Figures	21
Filling	33
Finding errors	64
Finding the current file	65
Finding the master file	65
Floats	21
Flymake	64
Folding	43, 48
Font Locking	36
Font macros	15
font-latex	36
Fonts	15
Formatting	30, 33, 50
Forward search	60
Free	2
Free software	2

G

General Public License	2
Generating symbols	77
German	73
Global directories	78
Global macro directory	78
Global style hook directory	78
Global T _E X macro directory	78
GPL	2

H

Header	50
Headers	48
Hide Macros	43
H _A T _E X	72

I

I/O correlation	55, 60
Including	68
Indentation	30
Indenting	30
Indexing	50
Initialization	7
Inline math, converting	24
input method	24
Inputting	68
Internationalization	72
Inverse search	60
iso-cvt.el	73
ISO 8859 Latin 1	72
ISO Character set	72
ispell	53, 73
Italian	73
Itemize	22
Items	22

J

Japan	75
Japanese	75
j _L T _E X	75
jT _E X	72, 75

K

Killing a process	65
kT _E X	72

L

Label prefix	18, 21
Labels	18, 21
Labels, adding	88
labels, completion of	28
lacheck	64
Language Support	72
L _A T _E X	50
Latin 1	72
License	2
Literature	50
Local style directory	79
Local style hooks	79

M

Macro arguments	25
Macro completion	25
Macro expansion	25
<code>macro.el</code>	90
<code>macro.tex</code>	90
Macros, adding	81
<code>makeindex</code>	50
Making a bibliography	50
Making an index	50
Many Files	68
Master file	65, 68
Matching dollar signs	12
Math environment, converting	24
Math mode delimiters	12
Math, fontification of	41
Math, fontification problems with	42
Mathematics	23
Multifile Documents	68
Multiple Files	68

N

Next error	62
Nippon	75
NTT jT _E X	72, 75

O

Other information, adding	88
Outlining	43, 48
Output	65
Overfull boxes	62
Overview	48

P

package/class options, Examining	88
Parsing errors	62
Parsing L ^A T _E X errors	62
Parsing new macros	90
Parsing T _E X	70, 77
Parsing T _E X output	62
PDF mode	55
PDFSync	55, 60
Personal customization	7
Personal information	78
Personal macro directory	78
Personal T _E X macro directory	78
pL ^A T _E X	75
Polish	73
Portuguese	73
Prefix for labels	18, 21
Previewing	58
Printing	50
Private directories	78
Private macro directory	78
Private style hook directory	78

Private T _E X macro directory	78
Problems	64
Processes	65
pT _E X	72, 75

Q

Quotes	12
Quotes, fontification of	40

R

Redisplay output	65
Refilling	33
Reformatting	30, 33
Region	50
Region file	50
Reindenting	30
Reveal	43
Right	2
Running BibT _E X	50
Running <code>chktex</code>	64
Running commands	50
Running Flymake	64
Running <code>lacheck</code>	64
Running L ^A T _E X	50
Running <code>makeindex</code>	50
Running T _E X	50

S

Sample style file	79
Sectioning	8, 16
Sectioning commands, fontification of	38
Sections	8, 16, 48
Setting the default command	50
Setting the header	50
Setting the trailer	50
Site customization	7
Site information	78
Site initialization	7
Site macro directory	78
Site T _E X macro directory	78
Slovak	73
Source specials	55, 60
Specifying a font	15
Starting a previewer	58
Stopping a process	65
Style	64
<code>style</code>	79
Style file	79
Style files	79
Style hook	79
Style hooks	79
subscript	23
Subscript, fontification of	41
superscript	23
Superscript, fontification of	41

support for completion of package/class options,	
Adding	88
Swedish	73
Symbols	23
SyncTeX	55, 60
Syntax Highlighting	36

T

Tabify	70
Table environment	21
Tables	21
Tabs	70
<code>tex-jp.el</code>	75
<code>tex-site.el</code>	7
TeX	50
TeX parsing	77
tool bar, toolbar	50
Trailer	50

U

Underfull boxes	62
Untabify	70
Updating style hooks	79
up ^L TeX	75
upTeX	72, 75

V

Verbatim, fontification of	42
Viewer predicates	88
Viewing	58

W

Warranty	2
Writing to a printer	50

X

X-Symbol	73
----------------	----