AUCTeX

A much enhanced TeX/LaTeX mode for Emacs.
Version 11.54

by Kresten Krab Thorup
updates from 6.1 to 11.13 by Per Abrahamsen
updates from 11.14 by David Kastrup

Short Contents

Copying 1
1 Introduction to AUCTEX 2
2 Installing AUCTEX 4
3 Quick Start 12
4 Inserting Frequently Used Commands 16
5 Advanced Editing Features 23
6 Controlling the Display of Source Code 30
7 Starting Processors, Viewers and Other Programs 38
8 Multifile Documents 44
9 Automatic Parsing of TEX files 46
10 Internationalization 48
11 Automatic Customization 50
12 Writing Your own Style Support 53
A Changes and New Features 60
B Future Development 65
C Frequently Asked Questions 69
Key Index 70
Function Index 71
Variable Index 72
Concept Index 74

Table of Contents

Copying **1**

1 Introduction to AUCTEX **2**
- 1.1 Availability 2
- 1.2 Contacts 3

2 Installing AUCTEX **4**
- 2.1 Prerequisites 4
- 2.2 Configure 4
- 2.3 Build/install 5
- 2.4 Loading the package 6
- 2.5 Providing AUCTEX as a package 6
- 2.6 Installation for non-privileged users 7
- 2.7 Installation under MS Windows 8
- 2.8 Customizing 10
- 2.9 Contributed files 11

3 Quick Start **12**
- 3.1 Functions for editing TeX files 12
  - 3.1.1 Making your TEX code more readable 12
  - 3.1.2 Entering sectioning commands 13
  - 3.1.3 Inserting environments 13
  - 3.1.4 Inserting macros 13
  - 3.1.5 Changing the font 13
  - 3.1.6 Other useful features 14
- 3.2 Creating and viewing output, debugging 14
  - 3.2.1 One Command for LATEX, helpers, viewers, and printing 14
  - 3.2.2 Choosing an output format 14
  - 3.2.3 Debugging LATEX 15
  - 3.2.4 Running LATEX on parts of your document 15

4 Inserting Frequently Used Commands **16**
- 4.1 Insertion of Quotes, Dollars, and Braces 16
- 4.2 Inserting Font Specifiers 17
- 4.3 Inserting chapters, sections, etc. 18
- 4.4 Inserting Environment Templates 20
  - 4.4.1 Equations 20
  - 4.4.2 Floats 21
  - 4.4.3 Itemize-like 21
  - 4.4.4 Tabular-like 21
  - 4.4.5 Customizing environments 22

5 Advanced Editing Features 23
5.1 Entering Mathematics 23
5.2 Completion 23
5.3 Commenting 25
5.4 Indenting 25
5.5 Filling 27

6 Controlling the Display of Source Code 30
6.1 Font Locking 30
6.2 Folding Macros and Environments 35
6.3 Outlining the Document 37

7 Starting Processors, Viewers and Other Programs 38
7.1 Executing Commands 38
7.2 Viewing the formatted output 40
7.2.1 Forward and inverse search 41
7.3 Catching the errors 41
7.4 Checking for problems 42
7.5 Controlling the output 42

8 Multifile Documents 44

9 Automatic Parsing of TeX files 46

10 Internationalization 48
10.1 Using AUCTeX with European Languages 48
10.2 Japanese TeX 49

11 Automatic Customization 50
11.1 Automatic Customization for the Site 50
11.2 Automatic Customization for a User 51
11.3 Automatic Customization for a Directory 51

12 Writing Your own Style Support 53
12.1 A Simple Style File 53
12.2 Adding Support for Macros 53
12.3 Adding Support for Environments 56
12.4 Adding Other Information 57
12.5 Automatic Extraction of New Things 57

Appendix A Changes and New Features 60

Appendix B Future Development 65
B.1 Mid-term Goals 65
B.2 Wishlist 65
B.3 Bugs 68

Appendix C Frequently Asked Questions 69

Key Index 70

Function Index 71

Variable Index 72

Concept Index 74

Copying

(This text stolen from the TEXinfo 2.16 distribution).

The programs currently being distributed that relate to AUCTEX include lisp files for Emacs (and XEmacs). These programs are *free*; this means that everyone is free to use them and free to redistribute them on a free basis. The AUCTEX related programs 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 programs that relate to AUCTEX, that you receive source code or else can get it if you want it, that you can change these programs 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 the AUCTEX related programs, 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 the programs that relate to AUCTEX. If these programs 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 programs currently being distributed that relate to AUCTEX are found in the General Public Licenses that accompany them.

1 Introduction to AUCTeX

This section of the AUCTeX manual gives a brief overview of what AUCTeX is, and the section is also available as a 'README' file. It is **not** an attempt to document AUCTeX. Real documentation for AUCTeX is available in the rest of the manual, which you can find in the 'doc' directory.

Read the 'INSTALL' or 'INSTALL.windows' file respectively for information about how to install AUCTeX. The files comprise the same information as the Installation chapter in the AUCTeX manual.

If you are upgrading from the previous version of AUCTeX, the latest changes can be found in the 'CHANGES' file. If you are upgrading from an older version, read the History chapter in the AUCTeX manual.

AUCTeX is a comprehensive customizable integrated environment for writing input files for TeX/LaTeX/ConTeXt using GNU Emacs.

AUCTeX lets you run TeX/LaTeX/ConTeXt and LaTeX/ConTeXt-related tools, such as a output filters or post processor from inside Emacs. Especially 'running LaTeX' is interesting, as AUCTeX lets you browse through the errors TeX 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.

AUCTeX automatically indents your 'LaTeX-source', not only as you write it — you can also let it indent and format an entire document. It has a special outline feature, which can greatly help you 'getting an overview' of a document.

Apart from these special features, AUCTeX provides a large range of handy Emacs macros, which in several different ways can help you write your LaTeX/ConTeXt documents fast and painlessly.

All features of AUCTeX are documented using the GNU Emacs online documentation system. That is, documentation for any command is just a key click away!

AUCTeX is written entirely in Emacs-Lisp, and hence you can easily add new features for your own needs. It was not made as part of any particular employment or project (apart from the AUCTeX project itself). AUCTeX is distributed under the 'GNU Emacs General Public License' and may therefore almost freely be copied and redistributed.

The next sections are a short introduction to some 'actual' features. For further information, refer to the built-in online documentation of AUCTeX.

1.1 Availability

The most recent version is always available at

http://ftp.gnu.org/pub/gnu/auctex/

WWW users may want to check out the AUCTeX page at

http://www.gnu.org/software/auctex/

1.2 Contacts

There has been established a mailing list for help, bug reports, feature requests and general discussion about AUCTEX. You're very welcome to join. Traffic average at an article by day, but they come in bursts. If you are only interested in information on updates, you could refer to the newsgroups 'comp.text.tex' and 'gnu.emacs.sources'.

If you want to contact the AUCTEX mailing list, send mail to auc-tex-subscribe@sunsite.dk in order to join. Articles should be sent to auc-tex@sunsite.dk.

To contact the current maintainers of AUCTEX directly, email auc-tex_mgr@sunsite.dk.

2 Installing AUCTEX

Installing AUCTEX should be simple: merely `./configure`, `make`, and `make install`. This does not yet activate the package, but merely makes it available. See Section 2.4 [Loading the package], page 6 for the activation. Please read through this document fully before installing anything. The installation procedure has changed as compared to earlier versions. In particular, note that there is some additional information for MS Windows installations in See Section 2.7 [Installation under MS Windows], page 8.

2.1 Prerequisites

- A recent version of Emacs 21, alternatively XEmacs 21.

  Support for Emacs 20 has been dropped in favor of getting more important work done. For XEmacs, you need at least version 1.84 of the `xemacs-base` package (released on 01/27/2004) or a sumo tarball dated 02/02/2004 or newer for compiling AUCTEX because of non-trivial changes in '`easy-mmode.el`': please use the XEmacs package system for upgrading if necessary. The current developers don't have the resources for providing backward compatibility to earlier versions.
- A working LATEX installation

  This is not really needed to *install* the package, but will be required for useful operation of it. The elisp of AUCTEX will probably run without LATEX, but you will find relatively little use for it.
- The `texinfo` package

  This is needed for building the documentation. If you don't have this, or you have a too old version of it (try building and you'll find out), you may download a separate tar file with the prebuilt documentation from Savannah and install it over the main unpacked tar archive.

2.2 Configure

The first step is to configure the source code, telling it where various files will be. To do so, run

```
./configure options
```

(Note: if you have fetched AUCTEX from CVS rather than a regular release, you will have to first generate `./configure` by running `autogen.sh` in the '`auctex`' directory.)

On many machines, you will not need to specify any options, but if `configure` cannot determine something on its own, you'll need to help it out with one of these options:

`--with-emacs[=/path/to/emacs]`
: If you are using a pretest which isn't in your `$PATH`, or `configure` is not finding the right Emacs executable, you can specify it with this option.

`--with-xemacs[=/path/to/xemacs]`
: Configure for generation under XEmacs (Emacs is the default). Again, the name of the right XEmacs executable can be specified, complete with path if necessary.

`--with-lispdir=/dir`
: This tells where to install Emacs Lisp files. Normally, this option is unnecessary, but may be used if you don't like the directory that configure is suggesting.

`--with-packagedir=/dir`
: This tells where to install the XEmacs Package. Again, this option is normally unnecessary, but may be used if you don't like the directory that configure is suggesting, and you know that XEmacs regards the directory you specify as a package directory.

  If you are installing AUCTeX for a single user, and you have installed no XEmacs packages as that user before, then `configure` may try to install AUCTeX in the systemwide package directory (that it cannot write to), causing installation to fail. In that case, a good value for this option is '`~/.xemacs/xemacs-packages`', as XEmacs looks there for per-user packages by default.

`--with-tex-input-dirs="/dir-1/;/dir-2/;...;/dir-N/"`
: This option allows to specify the directories to be used for file searches, e.g. for completion of LaTeX or BibTeX style file names. They will be used in the variable `TeX-macro-global`.

  Normally only the subdirectories '`tex/`' and '`bibtex/bst/`' below the used texmf trees should be relevant. Putting the roots of texmf trees here is possible but may slow down file searching.

  The directories should be separated by semicolons and each has to end with a directory separator, i.e. a slash or a backslash respectively. If more than one directory is specified or spaces are used in the name of a directory the whole string has to be quoted.

`--with-auto-dir=/dir`
: You can use this option to specify the directory containing automatically generated information. It is not necessary for most TeX installs, but may be used if you don't like the directory that configure is suggesting.

`--help`
: This is not an option specific to AUCTeX. A number of standard options to `configure` exist, and we do not have the room to describe them here; a short description of each is available, using `--help`.

2.3 Build/install

Once `configure` has been run, simply enter

```
make
```

at the prompt to byte-compile the lisp files, and build the documentation files. To install the files into the locations chosen earlier, type

```
make install
```

You may need special privileges to install, e.g., if you are installing into system directories.

2.4 Loading the package

First you should make sure that AUCTEX gets loaded. You then need to place a few lines in your personal '.emacs' file (or a site-wide configuration file).

For XEmacs, if you specified a valid package directory during installation, or none at all, then XEmacs installation should do everything necessary in order to install AUCTEX as a package and activate it. Restarting XEmacs should then make the package visible, and *C-c C-c* should give you a command prompt.

If you used `--with-packagedir`, you have to make sure that the directory 'lisp/auctex' under the directory you specified is in XEmacs' `load-path` variable.

For GNU Emacs, the recommended way to activate AUCTEX is to add the following line to your '.emacs' file:

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

If you used `--with-lispdir`, you have to make sure that the directory specified is in Emacs' `load-path` variable, so that you would instead use, e.g.,

```
(setq load-path (cons "~/elisp" load-path))
(require 'tex-site)
```

For site-wide activation in GNU Emacs, see See Section 2.5 [Advice for package providers], page 6.

That is all. There are other ways of achieving the equivalent thing, but we don't mention them here any more since they are not better, and people got confused into trying everything at once.

2.5 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. The use of packages should in general not impact performance negatively if a user chooses not to employ it, but should be as convenient as possible. The policy with regard to AUCTEX has been to *refrain* from activating it automatically when it is installed as a package. This is reasonable because

- Emacs comes with a simpler default TEX mode with different keybindings. Some users might prefer that.
- AUCTEX is activated via `(require 'tex-site)`. Once this has happened, it is not possible to get back the original TEX mode. A site-wide default would for this reason be hard to override.

If, however, you are certain that the users all prefer AUCTEX, you may place the following line in 'default.el':

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

XEmacs uses a package system. The default AUCTEX installation should cater for everything necessary in that case.

2.6 Installation for non-privileged users

Often people without system administration privileges want to install software for their private use. In that case you need to specify more options top the `configure` script. For XEmacs users, this is fairly easy, because the XEmacs package system has been designed to make this sort of thing practical: but GNU Emacs users (and XEmacs users for whom the package system is for some reason misbehaving) may need to do a little more work.

GNU Emacs users can solve this problem by using the '`--prefix`' option to the `configure` script, and let it point to the personal home directory. In that way, resulting binaries will be installed under the '`bin`' subdirectory of your home directory, manual pages under '`man`' and so on. That way, it is reasonably easy to maintain a bunch of additional packages, since the prefix argument is supported by most `configure` scripts.

You'll have to add something like '`/home/myself/share/emacs/site-lisp`' to your `load-path` variable, if it isn't there already.

XEmacs users can achieve the same end by pointing `configure` at an appropriate package directory (normally '`--with-packagedir=~/.xemacs/xemacs-packages`' will serve). This should only need to be done once, and should be needed fairly rarely; if you have installed any personal XEmacs packages before, `configure` should detect that, and automatically install AUCTeX there too; equally, if you have installed AUCTeX somewhere searched by XEmacs, AUCTeX should be automatically reinstalled over that copy.

(`configure` may guess wrong if the site administrator has installed AUCTeX somewhere else: if so, just use the '`--with-packagedir`' option to override `configure`'s choice.)

But there is another problem: perhaps you want to make it easy for other users to share parts of your personal Emacs configuration. In general, you can do this by writing '`~myself/`' anywhere where you specify paths to something installed in your personal subdirectories, not merely '`~/`', since the latter, when used by other users, will point to non-existent files.

For yourself, it will do to manipulate environment variables in your '`.profile`' resp. '`.login`' files. But if people will be copying just Elisp files, their copies will not work. While it would in general be preferable if the added components where available from a shell level, too (like when you call the standalone info reader, or try using '`preview.sty`' for functionality besides of Emacs previews), it will be a big help already if things work from inside of Emacs.

Here is how to do the various parts:

Making the Elisp available

In XEmacs, you should ask the other users to add symbolic links in their '`~/.xemacs/xemacs-packages/lisp`', '`~/.xemacs/xemacs-packages/info`' and '`~/.xemacs/xemacs-packages/etc`' directories. (Alas, there is presently no easy programmatic way to do this, except to have a script do the symlinking for them.)

In GNU Emacs, you'll want the invocation lines described in See Section 2.4 [Loading the package], page 6. In addition, you'll want a line such as

```
(add-to-list 'load-path "~myself/share/emacs/site-lisp/preview")
```

Making the Info files available

While for yourself, you'll probably want to manipulate the 'INFOPATH' variable; for access inside of Elisp something like the following might be convenient:

```
(eval-after-load 'info
   '(add-to-list 'Info-directory-list "~myself/info"))
```

In XEmacs, as long as XEmacs can see the package, there should be no need to do anything at all; the info files should be immediately visible. However, you might want to set 'INFOPATH' anyway, for the sake of standalone readers outside of XEmacs. (The info files in XEmacs are normally in '~/.xemacs/xemacs-packages/info'.)

2.7 Installation under MS Windows

Installation of AUCTEX is a bit more complicated, but we are working to resolve the issues involved. Please report success/failure to us at 'auc-tex@sunsite.dk'. Here are the steps to perform:

1. The installation of AUCTEX will require the MSYS tool set from http://www.mingw.org. If you have the Cygwin tool set from http://cygwin.com installed, that should do just fine as well, but it is quite larger and slower.

   If you are installing AUCTEX with one of those sets for an Emacs compiled in a different one, you should try to avoid tool-specific path names like '/cygwin/c'. Instead, use the 'c:' syntax. It might also help to use forward slashes instead of the backward slashes more typical for MS Windows: while backward slashes are supposed to work if properly escaped in the shell, this is one area easily overlooked by the developers. The same holds for file or directory names with spaces in them. Of course, we want to hear about any problems in that area.

   Compiling Emacs is outside of the scope of this manual. AUCTEX itself does not require a C compiler for installation.

2. Install GNU Emacs from http://ftp.gnu.org/pub/gnu/windows/emacs/ or XEmacs from http://www.xemacs.org.
3. You need a working TEX installation. One popular installation under Windows is MikTEX (http://www.miktex.org). Another much more extensive system is TEX live (http://www.tug.org/texlive) which is rather close to its Unix cousins.
4. Now the fun stuff starts. Unpack the AUCTEX distribution into some installation directory. **Do not** unpack it right into your Emacs' own directories: the installation will copy the material that needs to be placed there. Keep the installation directory separate: you can remove its contents after installation completes. Since you are reading this, you probably have already unpacked AUCTEX, but it should still be easy to move it elsewhere now.
5. Ready for takeoff. Start some shell (typically `bash`) capable of running `configure`, change into the installation directory and call `./configure` with appropriate options.

   Typical options you'll want to specify will be

   `--prefix=`*`drive:/path/to/emacs-directory`*

   which makes sure that (1) the AUCTeX manual will be installed in the 'info/' directory of your Emacs installation and (2) the automatically

generated global style hooks will be installed in the 'var/' directory of your Emacs installation. If you are collecting stuff like that in a central directory hierarchy (not untypical with Cygwin), you might want to specify that here instead. You stand a good chance that this will be the only option you need to supply, as long as your TEX-related executables are in your system path, which they better be for AUCTEX's operation, anyway.

`--with-emacs`
: if you are installing for a version of Emacs. You can use '`--with-emacs=`*`/path/to/emacs`*' to specify the name of the installed Emacs executable, complete with its path if necessary (if Emacs is not within a directory specified in your *PATH* environment setting).

`--with-xemacs`
: if you are installing for a version of XEmacs. Again, you can use '`--with-xemacs=`*`/path/to/xemacs`*' to specify the name of the installed XEmacs executable complete with its path if necessary. It may also be necessary to specify this option if a copy of Emacs is found in your *PATH* environment setting, but you still would like to install a copy of AUCTEX for XEmacs.

`--with-lispdir=`*`/dir`*
: This may be needed for GNU Emacs installation, but hopefully `configure` should figure this out by itself. Don't use this for XEmacs, rather use

`--with-packagedir=`*`/dir`*
: which gives the location of the package directory for XEmacs where stuff should be installed. Again, hopefully this is not necessary to specify.

`--with-tex-input-dirs="`*`/dir-1/;/dir-2/;...;/dir-N/`*`"`
: This option allows to specify the directories to be used for file searches, e.g. for completion of LaTeX or BibTeX style file names. They will be used in the variable `TeX-macro-global`.

  `configure` will usually figure out the right value for this option but will only succeed if it finds the necessary programs of the TEX system in the system path. Which should be there for running AUCTEX, anyway.

`--with-auto-dir=`*`/dir`*
: Directory containing automatically generated information. You should not normally need to set this, as '`--prefix`' should take care of this.

Some additional information about the above options may be found in Section 2.2 [Configure], page 4.

6. If you need to use the prebuilt documentation (see above), now is the time to unpack it over the rest of the installation directory.
7. Run `make` in the installation directory (we have had one report that Emacs did not manage to byte compile the Elisp files, and that had to be done by hand. No idea about what might have gone wrong there).
8. Run `make install` in the installation directory.

9. For GNU Emacs, the recommended way to activate AUCTEX is to add the following line to your '`.emacs`' file:

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

   The configuration for Windows systems is probably not quite fitting. Instead of loading '`tex-site.el`' in that manner, you might want to load '`tex-mik.el`' (for MikTEX) or '`tex-fptex`' (for fpTEX) instead. Those will lead to somewhat more appropriate values for your system. You can always use

   M-x customize-group ⟨RET⟩ *AUCTeX* ⟨RET⟩

   in order to customize more stuff, or use the '`Customize`' menu.

10. Load a '`.tex`' file Emacs or XEmacs and see if you get the '`Command`' menu. Try using that to LATEX the file.

Well, that about is all. Have fun!

2.8 Customizing

Most of the site-specific customization should already have happened during configuration 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 variables with a special significance. They are accessible directly by typing *M-x customize-variable RET <variable> RET*.

`TeX-lisp-directory` [User Option]

  The directory where you installed the AUCTEX lisp files.

This variable is set automatically during configuration. If you don't issue a `make install`, for example if you don't want to install AUCTEX in a different place, you will have to set this variable manually to the location of the compiled files. It is generally advisable to do a full installation including `make install` because program and documentation files will be copied to their proper places.

`TeX-macro-global` [User Option]

  Directories containing the site's TEX 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.

2.9 Contributed files

There are several files that are not part of AUCTEX proper, but included in the distribution in case they are useful.

'`bib-cite.el`'
: Better support for bibliographies and much more.

'`tex-jp.el`'
: Support for Japanese.

They can be installed together with AUCTEX by executing `make contrib` and `make install-contrib`. Read the comments in the start of each file for more information about how to use the files, what they do, and who wrote and maintains them.

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 LATEX 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 file '`INSTALL`' in the base directory of the unpacked distribution tarball. These installation instructions are available in this manual as well, Chapter 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.

If AUCTEX is installed, you might still need to activate it, by inserting

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

in your user init file.[1] In order to get support for many of the LATEX 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 multi-file 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, LATEX, and some other formats. Then we describe the functions that AUCTEX provides for processing the input files with LATEX, BibTEX, etc., and for viewing and debugging.

3.1 Functions for editing TeX files

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. You can enable it locally by typing *M-x font-lock-mode RET*. If you want to have font locking activated generally, enable `global-font-lock-mode`, e.g. with *M-x customize-variable RET global-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] This usually is a file in your home directory called '`.emacs`' if you are utilizing GNU Emacs or '`.xemacs/init.el`' if you are using XEmacs.

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.

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 LaTeX 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.

3.1.4 Inserting macros

`C-c C-m`, or simply *`C-c RET`* will give you a prompt that asks you for a LaTeX 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 5.2 [Completion], page 23.

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`*.

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 a 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.

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.

3.2 Creating and viewing output, debugging

3.2.1 One Command for LaTeX, helpers, viewers, and printing

If you have typed some text and want to run LaTeX (or TeX, 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.

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.

3.2.2 Choosing an output format

From a LaTeX 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 source specials support

C-c C-t C-o
: toggles usage of Omega/lambda.

3.2.3 Debugging LaTeX

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 occured 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 ocurred. 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-boxes`) you can toggle whether AUCTeX should notify you of overfull and underfull boxes in addition to regular errors.

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.

3.2.4 Running LaTeX 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.

4 Inserting Frequently Used Commands

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

4.1 Insertion of Quotes, Dollars, and Braces

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.

`TeX-close-quote` [User Option]

String inserted by typing `"` to close a quotation.

`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.

If you include the style files '`german`' or '`ngerman`' `TeX-open-quote` and `TeX-close-quote` will be set to the values of `LaTeX-german-open-quote` and `LaTeX-german-close-quote` respectively. `TeX-quote-after-quote` will be set to the value of `LaTeX-german-quote-after-quote`. If you write texts which facilitate the style files mentioned before, you should customize those special variables instead of the standard variables.

Similarly you should customize `LaTeX-csquotes-open-quote`, `LaTeX-csquotes-close-quote` and `LaTeX-csquotes-quote-after-quote` if you use the '`csquotes`' package. 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 '`(n)german`'-related ones.

In AUCTeX, dollar signs should match like they do in TeX. This has been partially implemented, we assume dollar signs always match within a paragraph. 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. If you enter a dollar sign that matches a double dollar sign '`$$`' AUCTeX will automatically insert two dollar signs. If you enter a second dollar sign that matches a single dollar sign, the single dollar sign will automatically be converted to a double dollar sign.

`TeX-insert-dollar` *arg* [Command]

($) Insert dollar sign.

Show matching dollar sign if this dollar sign end the TEX math mode. Ensure double dollar signs match up correctly by inserting extra dollar signs when needed.

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

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.

4.2 Inserting Font Specifiers

Perhaps the most used keyboard commands of AUCTEX 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 TEX 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-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.

`TeX-font` *arg* [Command]

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

If *replace* is not 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 not nil.

4.3 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

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

name Name of the sectioning command, derived from *level*.

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

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

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 *level* and *name*.

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

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

`LaTeX-section-section`
Insert LaTeX section command according to *name*, *title*, and *toc*. If *toc* is nil, no toc entry is inserted. If *toc* or *title* are empty strings, *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 '`.emacs`' file.

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.

4.4 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.

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 not-nil (i.e. you have given a prefix argument), the current environment is modified and no new environment is inserted.

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.

If the document is empty, or the cursor is placed at the top of the document, AUCTeX will default to insert a 'document' environment.

Most of these are described further in the following sections, and you may easily specify more. See Section 4.4.5 [Customizing environments], page 22.

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.

4.4.1 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`', '`aligned`', '`flalign`' and '`gather`'.

4.4.2 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 LATEX this is a sequence of the letters '`htbp`' as described in the LATEX manual. The value will default to the value of `LaTeX-float`.

caption This is the caption of the float.

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.

4.4.3 Itemize-like

In an itemize-like environment, nodes (i.e., '`\item`'s) may be inserted using *C-c* ⟨LFD⟩.

`LaTeX-insert-item` [Command]
(*C-c* ⟨LFD⟩) 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.

4.4.4 Tabular-like

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-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.

4.4.5 Customizing environments

See Section 12.3 [Adding Environments], page 56, for how to customize the list of known environments.

5 Advanced Editing Features

The previous chapter described how to write the main body of the text easily and with a minimum of errors. In this chapter we will describe some features for entering more specialized sorts of text, for formatting the source by indenting and filling and for navigating through the document.

5.1 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-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 '`.emacs`' file:

```
(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 variable `LaTeX-math-list` holds the actual mapping.

`LaTeX-math-list` [User Option]

A list containing key command mappings to use in `LaTeX-math-mode`. The car of each element is the key and the cdr is the macro name.

`LaTeX-math-menu-unicode` [User Option]

Whether the LaTeX 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.

5.2 Completion

Emacs lisp programmers probably know the `lisp-complete-symbol` command, usually bound to `M-⟨TAB⟩`. 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, usually bound to `M-⟨TAB⟩`. Using `LaTeX-complete-symbol` makes it easier to type and remember the names of long LaTeX 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`".

`TeX-complete-symbol` [Command]

(*M-*⟨TAB⟩) Complete TeX symbol before point.

A more direct way to insert a macro is with `TeX-insert-macro`, bound to *C-c C-m*. 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 AUCTeX knows the macro, prompt for each argument.

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

`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. If set to `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. `LaTeX-includegraphics-options-alist`.

`TeX-default-macro` [User Option]

Default macro to insert when invoking `TeX-insert-macro` first time.

A faster alternative is to bind the function `TeX-electric-macro` to '\'. This can be done by setting the variable `TeX-electric-escape`

`TeX-electric-escape` [User Option]

If this is non-nil when AUCTeX is loaded, the TeX escape character '\' will be bound to `TeX-electric-macro`

The difference between `TeX-insert-macro` and `TeX-electric-macro` is that space 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 AUCTeX knows the macro, prompt for each argument. Space will complete and exit.

By default AUCTeX will put an empty set braces '`{}`' after a macro with no arguments to stop it from eating the next whitespace. This can be stopped by entering `LaTeX-math-mode`, see Section 5.1 [Mathematics], page 23, or 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.

Completions work because AUCTEX can analyze TEX files, and store symbols in emacs lisp files for later retrieval. See Chapter 11 [Automatic], page 50, for more information.

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 BibTEX entries when you enter a '\cite' macro. For this kind of completion to work, parsing must be enabled as described in see Chapter 9 [Parsing Files], page 46. For '\cite' you must also make sure that the BibTEX files have been saved at least once after you enabled automatic parsing on save, and that the basename of the BibTEX file does not conflict with the basename of one of TEX files.

5.3 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.

5.4 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. If more environments are nested, they are indented 'accumulated' just like most programming languages usually are seen indented in nested constructs.

You can explicitely 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 accidently 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 5.5 [Filling], page 27, 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`.

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`' environments 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 LaTeX environments and TeX 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⟩
: `LaTeX-indent-line` will indent the current line.

⟨LFD⟩
: `newline-and-indent` inserts a new line (much like ⟨RET⟩) and moves the cursor to an appropriate position by the left margin.

  Most keyboards nowadays don't have a linefeed key and *C-j* is tedious to type. Therefore you can customize AUCTeX to perform indentation (or to make coffee) upon typing ⟨RET⟩ as well. The respective option is called `TeX-newline-function`.

C-j
: Alias for ⟨LFD⟩

`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.

The filling code currently cannot handle tabular-like environments which will be completely messed-up if you try to format them. This is why most of these environments are included in this customization option without a special indentation function. This will prevent that they get filled.

`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 LaTeX 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 `newline` 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 `newline-and-indent` or `reindent-then-newline-and-indent`. 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.

5.5 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. For all text modes with `text-mode-hook`, for all AUCTeX modes with `TeX-mode-hook` or for specific modes with `plain-TeX-mode-hook`, `LaTeX-mode-hook`, `ConTeXt-mode-hook` or `docTeX-mode-hook`. 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 explicitely 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 explicitely 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 LATEX 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.

M-g
: Alias for *C-c C-q C-r*

`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 LaTeX source in case a package for displaying formatted TeX 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.

6 Controlling the Display of Source Code

It is often desirable to get visual help of what markup code in a text actually does whithout having to decipher it explicitely. For this purpose Emacs and AUCTeX 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. AUCTeX 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, AUCTeX 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.

6.1 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 AUCTeX, the font locking patterns and functionality of font-latex are activated by default. You can switch to a different font locking scheme or disable font locking in AUCTeX by customizing the variable `TeX-install-font-lock`.

Besides font-latex AUCTeX ships with a scheme which is derived from Emacs' default LaTeX 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.

Macros

Highlighting of macros can be customized by adapting keyword lists which can be found in the customization group `font-latex-keywords`. The lists contain names of macros without the leading backslash.

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.

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-reference-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-title-fontity`.

`font-latex-title-fontity` [User Option]

Per default sectioning commands will be shown in a larger, proportional font, which corresponds to the value '`height`' for this variable. The font size varies with the sectioning level, e.g. '`\part`' will have a larger font than '`\subsection`'. font-latex provides four different levels. The respective keyword lists are described in the following table. If you rather like to use the base font and a different color, set the variable to '`color`'. In this case the face `font-lock-type-face` will be used to fontify the argument of the command.

`font-latex-match-title-1-keywords`
: Keywords for level 1 sectioning commands.
Face: `font-latex-title-1-face`

`font-latex-match-title-2-keywords`
: Keywords for level 2 sectioning commands.
Face: `font-latex-title-2-face`

`font-latex-match-title-3-keywords`
: Keywords for level 3 sectioning commands.
Face: `font-latex-title-3-face`

`font-latex-match-title-4-keywords`
: Keywords for level 4 sectioning commands.
Face: `font-latex-title-4-face`

Commands for changing fonts

LaTeX 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 TeX 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 LaTeX 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 LaTeX's built-in macros and macros of packages known by AUCTeX 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-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`

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 keywords are names of commands you want to match omitting the leading backslash.

The face argument can either be an existing face or font specifications made by you. (The latter option is not available on XEmacs.)

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.

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.

`font-latex-quotes` [User Option]

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.

Subscript and superscript in math

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.

Note that this feature is not available on XEmacs, for whicht it is disabled per default. In GNU Emacs raising and lowering is not enabled for versions 21.3 and before due to it working not properly.

`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*.

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 and consequently provides three keyword lists for customization.

`font-latex-verb-like-commands` [User Option]

List of commands with the form '`\foo|...|`' to be fontified as verbatim without the leading backslash. (The delimiters for the command's argument can be any character except an asterisk.)

`font-latex-verbatim-macros` [User Option]

List of macros with the form '`\foo{...}`' to be fontified as verbatim without the leading backslash. In contrast to `font-latex-verb-like-commands` the macro's argument is enclosed in a TeX group and not two arbitrary characters.

`font-latex-verbatim-environments` [User Option]

List of environments the content of which should be fontified as verbatim.

Multi-line fontification

Font locking in LaTeX source code often involves constructs spanning more than one line of text. For these constructs to be handled correctly GNU Emacs as well as font-latex provide mechanisms for multi-line fontification which can be controlled by the variable `font-latex-do-multi-line`.

`font-latex-do-multi-line` [User Option]

Control multi-line fontification.

Setting the variable to t will enable font-latex's mechanism, setting it to nil will disable it. Setting it to 'try-font-lock' (the default) will use font-lock's mechanism if available and font-latex's method if not.

Setting this variable will only have effect after resetting buffers controlled by font-latex or restarting Emacs.

Faces

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*.

6.2 Folding Macros and Environments

There can be macros and environments which have content that is not part of the text body you are writing, like footnotes and citations. Those enclose text which you often only want to see while actually editing it and which otherwise distract your view of the document body. Similarly there are macros where you are not interested in viewing the macro besides its content but rather want to see the content only, like font specifiers where the content might already be fontified in a special way by font locking.

With AUCTeX's folding functionality you can collapse those items and replace them by either a fixed string or the content of one of their arguments instead. If you want to make the original text visible again temporarily 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 latter is currently only supported on GNU Emacs.) 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 and environments:

`TeX-fold-buffer` [Command]

(*C-c C-o C-b*) Hide all macros specified in the variables `TeX-fold-macro-spec-list` and `TeX-fold-env-spec-list`. This command can also be used to refresh the whole buffer and hide any new macros and environments which were inserted after the last invocation of the command.

`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-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 sqare 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-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-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.

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

`TeX-fold-macro-spec-list` [User Option]
: List of display strings or argument numbers and macros to fold. If you specify a number, the content of the first mandatory argument of a LaTeX macro will be used as the placeholder.

  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-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.

6.3 Outlining the Document

AUCTEX supports the standard outline minor mode using LATEX/ConTEXt 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` [Variable]
: List of extra TEX 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)))
```

You may want to check out the unbundled '`out-xtra`' package for even better outline support. It is available from your favorite emacs lisp archive.

7 Starting Processors, Viewers and Other Programs

The most powerful features of AUCTeX may be those allowing you to run (La)TeX/ConTeXt and other external commands like BibTeX and `makeindex` from within Emacs, viewing and printing the results, and moreover allowing you to *debug* your documents.

7.1 Executing Commands

Formatting the document with TeX, LaTeX or ConTeXt, viewing with a previewer, printing the document, running BibTeX, making an index, or checking the document with `lacheck` or `chktex` all require running an external command.

There are two ways to run an external command, you can either run it on all of the current documents with `TeX-command-master`, or on the current region with `TeX-command-region`. A special case of running TeX 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`.

See Chapter 2 [Installation], page 4, for a discussion about `TeX-command-list` and Chapter 8 [Multifile], page 44 for a discussion about `TeX-master`.

`TeX-command-region` [Command]

(*C-c C-r*) Query the user for a command, and run it on the "region file". Some commands (typically those invoking TeX or LaTeX) will write the current region into the region file, after extracting the header and tailer from the master file. If mark is inactive (which can happen with `transient-mark-mode`), use the old 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-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`.

`TeX-command-buffer` [Command]

(*C-c C-b*) Query the user for a command, and run it on the "region file". Some commands (typically those invoking TeX or LaTeX) will write the current buffer into the region file, after extracting the header and tailer from the master file. See above for details.

AUCTeX will allow one process for each document, plus one process for the region file to be active at the same time. Thus, if you are editing *n* different documents, you can have *n* plus one processes running at the same time. If the last process you started was on the region, the commands described in Section 7.3 [Debugging], page 41 and Section 7.5 [Control], page 42 will work on that process, otherwise they will work on the process associated with the current document.

`TeX-region` [User Option]
: The name of the file for temporarily storing the text when formatting the current region.

`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 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 TeX mode.

AUCTeX will try to guess what command you want to invoke, but by default it will assume that you want to run TeX in TeX mode and LaTeX in LaTeX mode. You can overwrite this by setting the variable `TeX-command-default`.

`TeX-command-default` [User Option]
: The default command to run in this buffer. Must be an entry in `TeX-command-list`.

If you want to overwrite the values of `TeX-header-end`, `TeX-trailer-start`, or `TeX-command-default`, you can do that for all files by setting them in either `TeX-mode-hook`, `plain-TeX-mode-hook`, or `LaTeX-mode-hook`. To overwrite them for a single file, define them as file variables (see section "File Variables" in *The Emacs Editor*). You do this by putting special formatted text near the end of the file.

```
%%% Local Variables:
%%% TeX-header-end: "% End-Of-Header"
%%% TeX-trailer-start: "% Start-Of-Trailer"
%%% TeX-command-default: "SliTeX"
%%% End:
```

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.

`TeX-PDF-mode` [Command]
: (*C-c C-t C-p*) This command toggles the PDF mode of AUCTeX, a buffer-local minor mode. You can customize `TeX-PDF-mode` to give it a different default. The default

is used when AUCTEX does not have additional clue about what a document might want. This option usually results in calling either PDFTEX or ordinary TEX.

`TeX-DVI-via-PDFTeX` [User Option]

If this is set, DVI will also be produced by calling PDFTEX, setting `\pdfoutput=0`. This makes it possible to use packages like '`pdfcprot`' even when producing DVI files. Some modern TEX distributions, e.g. teTEX 3.0, do this anyway, so that you need not enable it within AUCTEX.

`TeX-interactive-mode` [Command]

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

`TeX-source-specials-mode` [Command]

(*C-c C-t C-s*) toggles Source Special support. Source Specials will move the DVI viewer to the location corresponding to point (forward search), and it will use '`emacsclient`' or '`gnuclient`' to have the previewer move Emacs to a location corresponding to a control-click in the previewer window. See Section 7.2 [Viewing], page 40.

You can permanently activate `TeX-source-specials-mode` with

```
(TeX-source-specials-mode 1)
```

or by customizing the variable `TeX-source-specials-mode`. There is a bunch of customization options, use `customize-group` on the group '`TeX-source-specials`' to find out more.

It has to be stressed *very* strongly however, that Source Specials can cause differences in page breaks, in 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.

`TeX-Omega-mode` [Command]

(*C-c C-t C-o*) This command toggles the use of the Omega (Ω) mode of AUCTEX, a buffer-local minor mode. If it is switched on, `omega` will be used instead of `tex`, and `lambda` instead of `latex`.

7.2 Viewing the formatted output

AUCTEX allows you to start external programs for previewing your document. These are normally invoked by pressing *C-c C-c* once the document is formatted or via the respective entry in the Command menu.

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 information about which file types and style options are associated with which viewers and options for them is stored in the variables `TeX-output-view-style` and `TeX-view-style`.

`TeX-view` [Command]

The command `TeX-view`, bound to *C-c C-v*, starts 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 DVI viewer when using `TeX-source-specials-mode`.

`TeX-output-view-style` [User Option]

List of output file extensions, style options and view options.

`TeX-view-style` [User Option]

List of style options and view options. This is the predecessor of `TeX-output-view-style` which does not allow the specification of output file extensions. It is used as a fallback in case none of the alternatives specified in `TeX-output-view-style` match. In case none of the entries in `TeX-view-style` match either, no suggestion for a viewer will be made.

7.2.1 Forward and inverse search

You can make use of forward and inverse searching if this is supported by your DVI viewer and you enabled `TeX-source-specials-mode` as described in Section 7.1 [Commands], page 38. AUCTeX will automatically pass the necessary command line options to the viewer in order to display the page containing the content you are currently editing (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-specials-view-start-server` to inhibit the question and always or never start the server respectively. Once the server and the viewer are running you can use a mouse click in the viewer to jump to the corresponding part of your document in Emacs (inverse search). Refer to the documentation of your viewer to find out what you have to do exactly. In xdvi you usually have to use *C-down-mouse-1*.

`TeX-source-specials-view-start-server` [User Option]

If `TeX-source-specials-mode` is active and a DVI 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 be started always. Set it to `nil` if the server should never be started. Inverse search will not be available in the latter case.

7.3 Catching the errors

Once you've formatted your document you may 'debug' it, i.e. browse through the errors (La)TeX reported.

`TeX-next-error` [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.

Normally AUCTeX will only report real errors, but you may as well ask it to report 'bad boxes' as well.

`TeX-toggle-debug-bad-boxes` [Command]

(*C-c C-w*) Toggle whether AUCTeX should stop at bad boxes (i.e. over/under full boxes) as well as at normal errors.

As default, AUCTeX will display that special '`*help*`' buffer containing the error reported by TeX along with the documentation. There is however an 'expert' option, which allows you to display the real TeX output.

`TeX-display-help` [User Option]

When non-nil AUCTeX will automatically display a help text whenever an error is encountered using `TeX-next-error` (*C-c `*).

7.4 Checking for problems

Running TeX or LaTeX will only find regular errors in the document, not examples of bad style. Furthermore, description of the errors may often be confusing. The utility `lacheck` can be used to find style errors, such as forgetting to escape the space after an abbreviation or using '`...`' instead of '`\ldots`' and many other problems like that. You start `lacheck` with *C-c C-c Check* ⟨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.

Another newer program which can be used to find errors is `chktex`. It is much more configurable than `lacheck`, but doesn't find all the problems `lacheck` does, at least in its default configuration. You must install the programs before using them, and for `chktex` you must also modify `TeX-command-list`. You can get `lacheck` from '`<URL:ftp://ftp.ctan.org/tex-archive/support/lacheck/>`' or alternatively `chktex` from '`<URL:ftp://ftp.ctan.org/tex-archive/support/chktex/>`'. Search for 'chktex' in '`tex.el`' to see how to switch between them.

7.5 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.

8 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 6.3 [Outline], page 37)). This can be done by having a "master" file in which you include the various files with the TeX macro '`\input`' or the LaTeX 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 '`.emacs`' file AUCTeX will use a more advanced algorithm.

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

If AUCTeX finds the line indicating the end of the header in a master file (`TeX-header-end`), it can figure out for itself that this is a master file. Otherwise, it will ask for the name of the master file associated with the buffer. To avoid asking you again, AUCTeX 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 '`.emacs`' file.

```
(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 AUCTeX 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`, AUCTeX will query you for the name.

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

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

`TeX-one-master` [User Option]

Regular expression matching ordinary TeX 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 AUCTeX 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 AUCTeX automatically modifying your files, you can set this variable to '`"<none>"`'. By default, AUCTeX will modify any file with an extension of '`.tex`'.

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.

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

AUCTeX keeps track of macros, environments, labels, and style files that are used in a given document. For this to work with multifile documents, AUCTeX 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 11.3 [Automatic Local], page 51.

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`.

9 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 '.emacs' file.

```
(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 11.3 [Automatic Local], page 51. 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 Chapter 8 [Multifile], page 44, 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 done 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 AUCTEX saves your buffer, it can optionally convert all tabs in your buffer into spaces. Tabs confuse AUCTEX's error message parsing and so should generally be avoided. However, tabs are significant in some environments, and so by default AUCTEX does not remove them. To convert tabs to spaces when saving a buffer, insert the following in your '.emacs' file:

```
(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 '.emacs' file.

```
;; Only parse \documentstyle information.
(setq-default TeX-auto-regexp-list 'LaTeX-auto-minimal-regexp-list)
;; The documentstyle command is usually near the beginning.
(setq-default TeX-auto-parse-length 2000)
```

This example will speed the parsing up significantly, but AUCTEX 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 TEX file that will be parsed.

The pre-specified lists of regexps are defined below. You can use these before loading AUCTEX by quoting them, as in the example above.

`TeX-auto-empty-regexp-list` [Constant]
: Parse nothing

`LaTeX-auto-minimal-regexp-list` [Constant]
: Only parse documentstyle.

`LaTeX-auto-label-regexp-list` [Constant]
: Only parse LATEX labels.

`LaTeX-auto-regexp-list` [Constant]
: Parse common LATEX commands.

`plain-TeX-auto-regexp-list` [Constant]
: Parse common plain TEX commands.

`TeX-auto-full-regexp-list` [Constant]
: Parse all TEX and LATEX commands that AUCTEX can use.

10 Internationalization

There are several problems associated with editing non-English TeX with GNU Emacs. Modern versions of GNU Emacs and TeX are usable for European (Latin, Cyrillic, Greek) based languages, but special versions of TeX and Emacs are needed for Korean, Japanese, and Chinese.

10.1 Using AUCTeX with European Languages

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.

With LaTeX2e, just add '`\usepackage[latin1]{inputenc}`'. Other languages than Western European ones will probably have other encoding needs.

To be able to display non-ASCII characters you will need an appropriate font and a version of GNU Emacs capable of displaying 8-bit characters (e.g. Emacs 21). The manner in which this is supported differs between Emacsen, so you need to take a look at your respective documentation.

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-compose.el`'
: Similar package bundled with new versions of XEmacs.

'`X-Symbol`'
: a much more complete package for both Emacs and XEmacs that can also handle a lot of mathematical characters and input methods.

AUCTeX supports style files for several languages. Each style file may modify some AUCTeX 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 the '`dk.sty`' file. This requires parsing to be enabled, see Chapter 9 [Parsing Files], page 46.

```
(add-hook 'TeX-language-dk-hook
  (function (lambda () (ispell-change-dictionary "danish"))))
```

The following style files are recognized.

'`dk`'
: Runs style hook `TeX-language-dk-hook`.

'`dutch`'
: Runs style hook `TeX-language-nl-hook`.

'`german`'
: Runs style hook `TeX-language-de-hook`. Gives '`"`' word syntax, makes the `"` key insert a literal '`"`', and pressing it twice will give you opening or closing german quotes ('`"‘`' or '`"’`'), if you have configured `TeX-open-quote` and `TeX-close-quote` accordingly. See Section 4.1 [Quotes], page 16.

'italian' Runs style hook TeX-language-it-hook. Pressing ⟨"⟩ will insert '"<' and '">' depending on context.

'plfonts'
'plhb' Runs style hook TeX-language-pl-hook. Gives '"' word syntax and makes the ⟨"⟩ key insert a literal '"'. Pressing ⟨"⟩ twice will insert '"<' or '">' depending on context.

10.2 Japanese TeX

To write Japanese text with AUCTeX you need to have versions of TeX and Emacs that support Japanese. There exist at least two variants of TeX for Japanese text (jTeX and pTeX), and AUCTeX can be used with MULE supported Emacsen.

To install Japanese support for AUCTeX, copy 'tex-jp.el' to AUCTeX installed directory. Next two commands will automatically install contributed files.

```
make contrib
make install-contrib
```

See 'INSTALL' and 'Makefile' for more information.

To use the Japanese TeX variants, simply enter japanese-tex-mode, japanese-latex-mode, or japanese-slitex-mode, and everything should work. If not, send mail to Shinji Kobayashi '<koba@flab.fujitsu.co.jp>', who kindly donated the code for supporting Japanese in AUCTeX. None of the primary AUCTeX maintainers understand Japanese, so they can not help you.

If you usually use AUCTeX in Japanese, setting following variables is useful.

TeX-default-mode [User Option]

Mode to enter for a new file when it can't be determined whether the file is plain TeX or LaTeX or what.

To use Japanese TeX always, set japanese command for example:

```
(setq TeX-default-mode 'japanese-latex-mode)
```

japanese-TeX-command-default [User Option]

The default command for TeX-command in japanese TeX mode.

The default value is 'jTeX'.

japanese-LaTeX-command-default [User Option]

The default command for TeX-command in japanese LaTeX mode.

The default value is 'jLaTeX'.

japanese-LaTeX-default-style [User Option]

The default style/class when creating new japanese LaTeX document.

The default value is 'j-article'.

See 'tex-jp.el' for more information.

11 Automatic Customization

Since AUCTEX is so highly customizable, it makes sense that it is able to customize itself. The automatic customization consists of scanning TEX 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 TEX 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 TEX documents.

If compared with the environment variable `TEXINPUTS`, the global level corresponds to the directories built into TEX. The private level corresponds to the directories you add yourself, except for '`.`', which is the local level.

By default AUCTEX 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 AUCTEX style files. Each must end with a slash.

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 name '`.`', '`..`', '`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.

11.1 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. Must end with a slash.

  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.

11.2 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`* to extract the information.

`TeX-macro-private` [User Option]
: Directories where you store your personal TeX macros. Each must end with a slash.

  This defaults to the directories listed in the '`TEXINPUTS`' and '`BIBINPUTS`' environment variables.

`TeX-auto-private` [User Option]
: List of directories containing automatically generated information. Must end with a slash.

  These correspond to the personal TeX macros.

`TeX-auto-generate` *TEX AUTO* [Command]
: (*`M-x TeX-auto-generate`*) 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 information. Must end with a slash.

  These correspond to the personal TeX macros.

11.3 Automatic Customization for a Directory

AUCTeX can update the style information about a file each time you save it, and it will do this if the directory `TeX-auto-local` exist. `TeX-auto-local` is by default set to '`"auto/"`', so simply creating an '`auto`' directory will enable automatic saving of style information.

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 TeX information. Must end with a slash.

  These correspond to TeX macros found in the current directory.

`TeX-auto-local` [User Option]

Directory containing automatically generated TeX information. Must end with a slash.

These correspond to TeX macros found in the current directory.

12 Writing Your own Style Support

See Chapter 11 [Automatic], page 50, 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 TeX style file, please send it to us.

12.1 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"
 (function (lambda () (setq LaTeX-largest-level
    (LaTeX-section-level ("chapter"))))))
```

This file specifies that the largest kind of section in a LaTeX document using the book document style is chapter. 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 is executed. The actual files will only be evaluated once, but the hooks will be called for each buffer using the style file.

`TeX-add-style-hook` *style hook* [Function]

Add *hook* to the list of functions to run when we use the TeX file *style*.

12.2 Adding Support for Macros

The most common thing to define in a style hook is new symbols (TeX 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"
 (function
  (lambda ()
     (TeX-add-symbols
     '("arabic" TeX-arg-counter)
     '("label" TeX-arg-define-label)
     '("ref" TeX-arg-label)
     '("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 TeX macro. What is done depends on the argument type.

If a macro is defined multiple times, AUCTeX will chose 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-label 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.

`string` Use the string as a prompt to prompt for the argument.

`number` Insert that many braces, leave point inside the first.

`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-file`
: Prompt for a tex or sty filename, and use it without the extension. Run the file hooks defined for it.

`TeX-arg-label`
: Prompt for a label completing with known labels.

`TeX-arg-macro`
: Prompt for a TeX macro with completion.

`TeX-arg-environment`
: Prompt for a LaTeX environment with completion.

`TeX-arg-cite`
: Prompt for a BibTeX citation.

`TeX-arg-counter`
: Prompt for a LaTeX counter.

`TeX-arg-savebox`
: Prompt for a LaTeX savebox.

`TeX-arg-file`
: Prompt for a filename in the current directory, and use it without the extension.

`TeX-arg-input-file`
: Prompt for a filename in the current directory, and use it without the extension. Run the style hooks for the file.

`TeX-arg-define-label`
: Prompt for a label completing with known labels. Add label to list of defined labels.

`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-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-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.

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 `exit-mark`. It will point nowhere, until the argument hook sets it.

12.3 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"
 (function
  (lambda ()
    (LaTeX-add-environments
     '("document" LaTeX-env-document)
     '("enumerate" LaTeX-env-item)
     '("itemize" LaTeX-env-item)
     '("list" LaTeX-env-list)))))
```

The only hook that is generally useful is `LaTeX-env-item`, which is used for environments that contain items. 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`. 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"
 (function
  (lambda ()
    (LaTeX-add-environments
     '("loop" "From" "To" "Step")))))
```

If an environment is defined multiple times, AUCTEX will chose the one with the longest definition. Thus, if you have an enumerate style file, and want it to replace the standard LATEX 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"
 (function
  (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*.

12.4 Adding Other Information

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.

12.5 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'.
  (mapcar (function (lambda (list)
    (mapcar (function (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"
 (function
  (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 Changes and New Features

News in 11.54

- The parser (used e.g. for `TeX-auto-generate-global`) was extended to recognize keywords common in LaTeX packages and classes, like "\DeclareRobustCommand" or "\RequirePackage". Additionally a bug was fixed which led to duplicate entries in AUCTeX style files.
- Folding can now be done for paragraphs and regions besides single constructs and the whole buffer. With the new `TeX-fold-dwim` command content can both be hidden and shown with a single key binding. In course of these changes new key bindings for unfolding commands where introduced. The old bindings are still present but will be phased out in future releases.
- Info files of the manual now have a .info extension.
- There is an experimental toolbar support now. It is not activated by default. If you want to use it, add

  ```
  (add-hook 'LaTeX-mode-hook 'LaTeX-install-toolbar)
  ```

  to your init file.
- The manual now contains a new chapter "Quick Start". It explains the main features and how to use them, and should be enough for a new user to start using AUCTeX.
- A new section "Font Locking" was added to the manual which explains syntax highlighting in AUCTeX and its customization. Together with the sections related to folding and outlining, the section is part of the new chapter "Display".
- Keywords for syntax highlighting of LaTeX constructs to be typeset in bold, italic or typewriter fonts may now be customized. Besides the built-in classes, new keyword classes may be added by customizing the variable '`font-latex-user-keyword-classes`'. The customization options can be found in the customization group '`font-latex-keywords`'.
- Verbatim content is now displayed with the '`fixed-pitch`' face. (GNU Emacs only)
- Syntax highlighting should not spill out of verbatim content anymore. (GNU Emacs only)
- Verbatim commands like '`\verb|...|`' will not be broken anymore during filling.
- You can customize the completion for graphic files with `LaTeX-includegraphics-read-file`.
- Support for the LaTeX packages '`url`', '`listings`', '`jurabib`' and '`csquotes`' was added with regard to command completion and syntax highlighting.
- Performance of fontification and filling was improved.
- Insertion of nodes in Texinfo mode now supports completion of existing node names.
- Setting the variable `LaTeX-float` to `nil` now means that you will not be prompted for the float position of figures and tables. You can get the old behaviour of `nil` by setting the variable to `""`, i.e. an empty string. See also Section 4.4.2 [Floats], page 21.
- The XEmacs-specific bug concerning `overlays-at` was fixed.
- Lots of bug fixes.

News in 11.53

- The LaTeX math menu can include Unicode characters if your Emacs built supports it. See the variable `LaTeX-math-menu-unicode`, Section 5.1 [Mathematics], page 23.
- Bug fixes for XEmacs.
- Completion for graphic files in the TeX search path has been added.
- `start` is used for the viewer for MikTeX and fpTeX.
- The variable `TeX-fold-preserve-comments` can now be customized to deactivate folding in comments.

News in 11.52

- Installation and menus under XEmacs work again (maybe for the first time).
- Fontification of subscripts and superscripts is now disabled when the fontification engine is not able to support it properly.
- Bug fixes in the build process.

News in 11.51

- PDFTeX and Source Special support did not work with ConTeXt, this has been fixed. Similar for Source Special support under Windows.
- Omega support has been added.
- Bug fixes in the build process.
- `TeX-fold` now supports folding of environments in Texinfo mode.

News in 11.50

- The use of source specials when processing or viewing the document can now be controlled with the new `TeX-source-specials` minor mode which can be toggled via an entry in the Command menu or the key binding *C-c C-t C-s*. If you have customized the variable `TeX-command-list`, you have to re-initialize it for this to work. This means to open a customization buffer for the variable by typing `M-x customize-variable RET TeX-command-list RET`, selecting "Erase Customization" and do your customization again with the new default.
- The content of the command menu now depends on the mode (plain TeX, LaTeX, ConTeXt etc.). Any former customization of the variable `TeX-command-list` has to be erased. Otherwise the command menu and the customization will not work correctly.
- Support for hiding and auto-revealing macros, e.g. footnotes or citations, and environments in a buffer was added, Section 6.2 [Folding], page 35.
- You can now control if indentation is done upon typing ⟨RET⟩ by customizing the variable `TeX-newline-function`, Section 5.4 [Indenting], page 25.
- Limited support for `doc.sty` and `ltxdoc.cls` ('`dtx`' files) was added. The new docTeX mode provides functionality for editing documentation parts. This includes formatting (indenting and filling), adding and completion of macros and environments while staying in comments as well as syntax highlighting. (Please note that the mode is not finished yet. For example syntax highlighting does not work yet in XEmacs.)

- For macro completion in docTeX mode the AUCTeX style files '`doc.el`', '`ltxdoc.el`' and '`ltx-base.el`' were included. The latter provides general support for low-level LaTeX macros and may be used with LaTeX class and style files as well. It is currently not loaded automatically for those files.
- Support for ConTeXt with a separate ConTeXt mode is now included. Macro definitions for completion are available in Dutch and English.
- The filling and indentation code was overhauled and is now able to format commented parts of the source syntactically correct. Newly available functionality and customization options are explained in the manual.
- Filling and indentation in XEmacs with preview-latex and activated previews lead to the insertion of whitespace before multi-line previews. AUCTeX now contains facilities to prevent this problem.
- If `TeX-master` is set to `t`, AUCTeX will now query for a master file only when a new file is opened. Existing files will be left alone. The new function `TeX-master-file-ask` (bound to *C-c _* is provided for adding the variable manually.
- Sectioning commands are now shown in a larger font on display devices which support such fontification. The variable `font-latex-title-fontity` can be customized to restore the old appearance, i.e. the usage of a different color instead of a change in size.
- Support for `alphanum.sty`, `beamer.cls`, `booktabs.sty`, `captcont.sty`, `emp.sty`, `paralist.sty`, `subfigure.sty` and `units.sty/nicefrac.sty` was added. Credits go to the authors mentioned in the respective AUCTeX style files.
- Inserting graphics with *C-c RET \includegraphics RET* was improved. See the variable `LaTeX-includegraphics-options-alist`.
- If `LaTeX-default-position` is `nil`, don't prompt for position arguments in Tabular-like environments, see Section 4.4.4 [Tabular-like], page 21.
- Completion for available packages when using *C-c RET \usepackage RET* was improved on systems using the kpathsea library.
- The commenting functionality was fixed. The separate functions for commenting and uncommenting were unified in one function for paragraphs and regions respectively which do both.
- Syntax highlighting can be customized to fontify quotes delimited by either >>German<< or <<French>> quotation marks by changing the variable `font-latex-quotes`.
- Certain TeX/LaTeX keywords for functions, references, variables and warnings will now be fontified specially. You may add your own keywords by customizing the variables `font-latex-match-function-keywords`, `font-latex-match-reference-keywords`, `font-latex-match-variable-keywords` and `font-latex-match-warning-keywords`.
- If you include the style files '`german`' or '`ngerman`' in a document (directly or via the '`babel`' package), you should now customize `LaTeX-german-open-quote`, `LaTeX-german-close-quote` and `LaTeX-german-quote-after-quote` instead of `TeX-open-quote`, `TeX-close-quote` and `TeX-quote-after-quote` if you want to influence the type of quote insertion.
- Upon viewing an output file, the right viewer and command line options for it are now determined automatically by looking at the extension of the output file and certain

options used in the source file. The behavior can be adapted or extended respectively by customizing the variable `TeX-output-view-style`.

- You can control whether `TeX-insert-macro` (*C-c RET*) ask for all optional arguments by customizing the variable `TeX-insert-macro-default-style`, Section 5.2 [Completion], page 23.
- `TeX-run-discard` is now able to completely detach a process that it started.
- The build process was enhanced and is now based on `autoconf` making installing AUCTeX a mostly automatic process. See Chapter 2 [Installation], page 4 and Section 2.7 [Installation under MS Windows], page 8 for details.

News in 11.14

- Many more LaTeX and LaTeX2e commands are supported. Done by Masayuki Ataka <ataka@milk.freemail.ne.jp>

News in 11.12

- Support for the KOMA-Script classes. Contributed by Mark Trettin <Mark.Trettin@gmx.de>.

News in 11.11

- Support for '`prosper.sty`', see `http://prosper.sourceforge.net/`. Contributed by Phillip Lord <p.lord@russet.org.uk>.

News in 11.10

- `comment-region` now inserts `%%` by default. Suggested by "Davide G. M. Salvetti" <salve@debian.org>.

News in 11.06

- You can now switch between using the '`font-latex`' (all emacsen), the '`tex-font`' (Emacs 21 only) or no special package for font locking. Customize `TeX-install-font-lock` for this.

News in 11.04

- Now use `-t landscape` by default when landscape option appears. Suggested by Erik Frisk <frisk@isy.liu.se>.

News in 11.03

- Use '`tex-fptex.el`' for fpTeX support. Contributed by Fabrice Popineau <Fabrice.Popineau@supelec.fr>.

News in 11.02

- New user option `LaTeX-top-caption-list` specifies environments where the caption should go at top. Contributed by ataka@milk.freemail.ne.jp (Masayuki Ataka).

- Allow explicit dimensions in '`graphicx.sty`'. Contributed by ataka@milk.freemail.ne.jp (Masayuki Ataka).
- Limited support for '`verbatim.sty`'. Contributed by ataka@milk.freemail.ne.jp (Masayuki Ataka).
- Better support for asmmath items. Patch by ataka@milk.freemail.ne.jp (Masayuki Ataka).
- More accurate error parsing. Added by David Kastrup <David.Kastrup@t-online.de>.

News in 11.01

- Bug fixes.

Older versions

See the file '`history.texi`' for older changes.

Appendix B Future Development

The following sections describe future development of AUCTeX. 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.

B.1 Mid-term Goals

- Integration of preview-latex into AUCTeX

  AUCTeX users will thereby get the functionality provided by preview-latex without the hassle of an additional installation.

  Looking at the backend, the integration involves folding of error parsing and task management of both packages which will ease development efforts and avoid redundant work.

- More flexible option and command handling

  The current state of command handling with `TeX-command-list` is not very flexible because there is no distinction between executables and command line options to be passed to them.

  Customization of `TeX-command-list` by the user will interfere with updates of AUCTeX.

- Error help catalogs

  Currently, the help for errors is more or less hardwired into '`tex.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 the were identified.

- Combining '`docTeX`' with RefTeX

  Macro cross references should also be usable for document navigation using RefTeX.

B.2 Wishlist

- Spell checking of macros

  A special ispell dictionary for macros could be nice to have.

- Quick error overviews

  An error overview window (extract from the log file with just the error lines, clickable like a "grep" buffer) and/or 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 LaTeX 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 TeX support For starters, `LaTeX-math-mode` is not very LaTeX-specific in the first place, and similar holds for indentation and formatting.
- Poor man's Source Specials In particular in PDF mode (and where Source Specials cause problems), alternatives would be desirable. One could implement inverse search by something like Heiko Oberdiek's '`vpe.sty`', and forward search by using the '`.aux`' file info to correlate labels in the text (possibly in cooperation with RefTeX) with previewer pages.
- 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 Tinnefeld <tinnefeld@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 behaviour. 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 controled with mapping based on the extension.
- Multiple argument completion for '`\bibliography`'. In general, I ought to make `,` special for these kind of completions.
- Suggest '`makindex`' when appropriate.
- AUCTeX should be able to parse LaTeX2e '`.cls`' files. Here are the regexps by '`<thiemann@informatik.uni-tuebingen.de>`' Peter Thiemann.

```
        ("\\\\DeclareRobustCommand{?\\\\\\([a-zA-Z]+\\)}?\\[\\([0-9]+\\)\\]\
    \\[\\([^\]\\\\\n\r]+\\)\\]"
         (1 2 3) LaTeX-auto-optional)
        ("\\\\DeclareRobustCommand{?\\\\\\([a-zA-Z]+\\)}?\\[\\([0-9]+\\)\\]"
         (1 2) LaTeX-auto-arguments)
        ("\\\\DeclareRobustCommand{?\\\\\\([a-zA-Z]+\\)}?" 1 TeX-auto-symbol)
        ("\\\\DeclareFixedFont{?\\\\\\([a-zA-Z]+\\)}?"
         1 TeX-auto-symbol)
        ("\\\\Declare\\(Text\\|Old\\)FontCommand{?\\\\\\([a-zA-Z]+\\)}?"
         2 TeX-auto-symbol)
```

```
("\\\\DeclareMath\\(Symbol\\|Delimiter\\|Accent\\|Radical\\){?\\\\\\([a-zA-Z]
 2 TeX-auto-symbol)
;;; it is also valid to declare just a single symbol, e.g. <,
;;; with \DeclareMathSymbol but it is not necessary to register that here
("\\\\DeclareText\\(Command\\|Symbol\\|Accent\\|Composite\\){?\\\\\\([a-zA-Z]
 2 TeX-auto-symbol)
```

- 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.
- The parser should catch warnings about undefined crossreferences. Suggested by Richard Hirsch '`i3080501@ws.rz.tu-bs.de`'.
- 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>`'.
- 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 Bhumbla '`<rbhumbla@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>`'.
- Make features.

  A new command `TeX-update` (*`C-c C-u`*) could be used to create an up-to-date dvi file by repeatedly running BibTeX, MakeIndex and (La)TeX, until an error occurs or we are done.

  An alternative is to have an '`Update`' command that ensures the '`dvi`' file is up to date. This could be called before printing and previewing.
- 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
  ```
- Completion for counters and sboxes.

- 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.

- Make `TeX-next-error` parse ahead and store the results in a list, using markers to remember buffer positions in order to be more robust with regard to line numbers and changed files. This is what `next-error` does. (Or did, until Emacs 19).
- 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`’.

B.3 Bugs

- The parsed files and style hooks for ‘`example.dtx`’, ‘`example.sty`’, ‘`example.drv`’ and ‘`example.bib`’ all clash. Bad.
- *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.
- AUCTeX 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 tabular, tabbing, table, math, and array environments.
- Syntax highlighting of LaTeX constructs spanning more than one line sometimes stops in the middle of the construct. Highlighting can get lost during typing. (XEmacs only)
- No syntactic font locking of verbatim macros and environments. (XEmacs only)
- Font locking inside of verbatim macros and environments is not inhibited. This may result in syntax highlighting of unbalanced dollar signs and the like spilling out of the verbatim content. (XEmacs only)
- Folding of LaTeX constructs spanning more than one line may result in overfull lines. (XEmacs only)

Appendix C Frequently Asked Questions

1. What versions of Emacs and XEmacs are supported?

   AUCTeX was tested with Emacs 21 and XEmacs 21.4.15. Older versions may work but are unsupported. Older versions of XEmacs might possibly made to work by updating the 'xemacs-base' package through the XEmacs package system.

2. Why doesn't the completion or multifile stuff work?

   It must be enabled first, insert this in your emacs:

   ```
   (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.

3. Why doesn't `TeX-save-document` work?

   `TeX-check-path` has to contain "./" somewhere.

4. 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'.

5. 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.

Key Index

"

" 16

$

$ 17

C

C-c % 25
C-c ; 25
C-c] 20
C-c ^ 43
C-c _ 45
C-c ‘ 42
C-c { 17
C-c ~ 23
C-c C-b 38
C-c C-c 38
C-c C-d 45
C-c C-e 20
C-c C-f 17
C-c C-f C-b 13, 17
C-c C-f C-c 14, 17
C-c C-f C-e 13, 17
C-c C-f C-f 14, 17
C-c C-f C-i 13, 17
C-c C-f C-r 13, 17
C-c C-f C-s 13, 17
C-c C-f C-t 14, 17
C-c C-k 42
C-c C-l 42
C-c C-m 24
C-c C-n 46
C-c C-o b 36
C-c C-o C-b 35
C-c C-o C-e 36
C-c C-o C-f 35
C-c C-o C-m 36
C-c C-o C-o 36
C-c C-o C-p 36
C-c C-o C-r 36
C-c C-o i 36
C-c C-o p 36
C-c C-o r 36
C-c C-q C-e 28
C-c C-q C-p 28
C-c C-q C-r 28
C-c C-q C-s 28
C-c C-r 38
C-c C-s 18
C-c C-t C-i 40
C-c C-t C-o 40
C-c C-t C-p 39
C-c C-t C-r 38
C-c C-t C-s 40
C-c C-v 41
C-c C-w 42
C-c ⟨LFD⟩ 21
C-j 26

L

⟨LFD⟩ 26

M

M-g 28
M-q 28
M-⟨TAB⟩ 24

T

⟨TAB⟩ 26

Function Index

L

LaTeX-add-bibliographies 57
LaTeX-add-environments 57
LaTeX-add-labels 57
LaTeX-close-environment 20
LaTeX-env-item 56
LaTeX-environment 20
LaTeX-fill-environment 28
LaTeX-fill-paragraph 28
LaTeX-fill-region 28
LaTeX-fill-section 28
LaTeX-indent-line 26
LaTeX-insert-environment 57
LaTeX-insert-item 21
LaTeX-math-mode 23
LaTeX-section 18
LaTeX-section-heading 19
LaTeX-section-label 19
LaTeX-section-section 19
LaTeX-section-title 19
LaTeX-section-toc 19

T

TeX-add-style-hook 53
TeX-add-symbols 53
TeX-arg-cite 55
TeX-arg-conditional 54
TeX-arg-coordinate 56
TeX-arg-corner 55
TeX-arg-counter 55
TeX-arg-define-cite 55
TeX-arg-define-counter 55
TeX-arg-define-environment 55
TeX-arg-define-label 55
TeX-arg-define-macro 55
TeX-arg-define-savebox 55
TeX-arg-environment 55
TeX-arg-eval 54
TeX-arg-file 55
TeX-arg-free 54
TeX-arg-input-file 55
TeX-arg-label 55
TeX-arg-literal 54
TeX-arg-lr 55
TeX-arg-macro 55
TeX-arg-pagestyle 56
TeX-arg-pair 56
TeX-arg-savebox 55
TeX-arg-size 56
TeX-arg-tb 55
TeX-arg-verb 56
TeX-auto-generate 51
TeX-command-buffer 38
TeX-command-master 38
TeX-command-region 38
TeX-comment-or-uncomment-paragraph 25
TeX-comment-or-uncomment-region 25
TeX-complete-symbol 24
TeX-electric-macro 24
TeX-fold-buffer 35
TeX-fold-clearout-buffer 36
TeX-fold-clearout-item 36
TeX-fold-clearout-paragraph 36
TeX-fold-clearout-region 36
TeX-fold-dwim 36
TeX-fold-env 36
TeX-fold-macro 36
TeX-fold-mode 35
TeX-fold-paragraph 36
TeX-fold-region 36
TeX-font 17
TeX-header-end 44
TeX-home-buffer 43
TeX-insert-braces 17
TeX-insert-dollar 17
TeX-insert-macro 24
TeX-insert-quote 16
TeX-interactive-mode 40
TeX-kill-job 42
TeX-master-file-ask 45
TeX-next-error 42
TeX-normal-mode 46
TeX-Omega-mode 40
TeX-PDF-mode 39
TeX-pin-region 38
TeX-recenter-output-buffer 42
TeX-save-document 45
TeX-source-specials-mode 40
TeX-toggle-debug-bad-boxes 42
TeX-view 41

Variable Index

F

font-latex-do-multi-line 34
font-latex-fontify-script 34
font-latex-match-bold-command-keywords ... 32
font-latex-match-bold-declaration-keywords 32
font-latex-match-function-keywords 31
font-latex-match-italic-command-keywords 32
font-latex-match-italic-declaration-keywords 32
font-latex-match-math-command-keywords ... 32
font-latex-match-reference-keywords 31
font-latex-match-textual-keywords 31
font-latex-match-title-1-keywords 32
font-latex-match-title-2-keywords 32
font-latex-match-title-3-keywords 32
font-latex-match-title-4-keywords 32
font-latex-match-type-command-keywords ... 32
font-latex-match-type-declaration-keywords 32
font-latex-match-variable-keywords 31
font-latex-match-warning-keywords 31
font-latex-quotes 33
font-latex-script-display 34
font-latex-title-fontity 32
font-latex-user-keyword-classes 33
font-latex-verb-like-commands 34
font-latex-verbatim-environments 34
font-latex-verbatim-macros 34

J

japanese-LaTeX-command-default 49
japanese-LaTeX-default-style 49
japanese-TeX-command-default 49

L

LaTeX-amsmath-label 21
LaTeX-auto-label-regexp-list 47
LaTeX-auto-minimal-regexp-list 47
LaTeX-auto-regexp-list 47
LaTeX-csquotes-close-quote 16
LaTeX-csquotes-open-quote 16
LaTeX-csquotes-quote-after-quote 16
LaTeX-default-environment 20
LaTeX-default-format 21
LaTeX-default-position 21
LaTeX-eqnarray-label 20
LaTeX-equation-label 20
LaTeX-figure-label 21
LaTeX-fill-break-at-separators 28
LaTeX-fill-break-before-code-comments 29
LaTeX-float 21
LaTeX-german-close-quote 16
LaTeX-german-open-quote 16
LaTeX-german-quote-after-quote 16
LaTeX-indent-environment-check 26
LaTeX-indent-environment-list 26
LaTeX-indent-level 25, 26
LaTeX-item-indent 25, 27
LaTeX-math-abbrev-prefix 23
LaTeX-math-list 23
LaTeX-math-menu-unicode 23
LaTeX-paragraph-commands 28
LaTeX-section-hook 18, 19
LaTeX-section-label 18, 19
LaTeX-syntactic-comments 26, 27
LaTeX-table-label 21

P

plain-TeX-auto-regexp-list 47

T

TeX-auto-cleanup-hook 59
TeX-auto-empty-regexp-list 47
TeX-auto-full-regexp-list 47
TeX-auto-global 51
TeX-auto-local 52
TeX-auto-parse-length 47
TeX-auto-prepare-hook 59
TeX-auto-private 51
TeX-auto-regexp-list 47, 59
TeX-auto-save 46
TeX-auto-untabify 46
TeX-brace-indent-level 27
TeX-check-path 39
TeX-close-quote 16
TeX-command-default 39
TeX-command-list 38
TeX-default-macro 24
TeX-default-mode 49
TeX-display-help 42
TeX-DVI-via-PDFTeX 40
TeX-electric-escape 24
TeX-file-recurse 50
TeX-fold-env-spec-list 37
TeX-fold-macro-spec-list 36
TeX-fold-unspec-env-display-string 37
TeX-fold-unspec-macro-display-string 37
TeX-fold-unspec-use-name 37
TeX-font-list 18
TeX-header-end 38, 39
TeX-ignore-file 50
TeX-insert-braces 24
TeX-insert-macro-default-style 24

TeX-install-font-lock 30
TeX-interactive-mode 40
TeX-lisp-directory 10
TeX-macro-global 10, 50
TeX-macro-private 51
TeX-master 38, 44
TeX-newline-function 25, 27
TeX-Omega-mode 40
TeX-one-master 44
TeX-open-quote 16
TeX-outline-extra 37
TeX-output-view-style 41
TeX-parse-self 46
TeX-PDF-mode 39
TeX-quote-after-quote 16
TeX-region 38, 39
TeX-save-query 45
TeX-source-specials-mode 40
TeX-source-specials-view-start-server 41
TeX-style-global 51
TeX-style-local 51
TeX-style-path 50
TeX-style-private 51
TeX-trailer-start 38, 39
TeX-view-style 41

Concept Index

.

'.emacs' 6

\

'\begin' 20
\chapter 13, 18
\cite, completion of 25
\emph 13, 17
'\end' 20
\include 44
\input 44
\item 21
\label 13, 18
\label, completion 25
\ref, completion 25
\section 13, 18
\subsection 13, 18
\textbf 13, 17
\textit 13, 17
\textrm 13, 17
\textsc 14, 17
\textsf 14, 17
\textsl 13, 17
\texttt 14, 17

A

Abbreviations 23
Adding a style hook 53
Adding bibliographies 57
Adding environments 56
Adding labels 57
Adding macros 53
Adding other information 57
Advanced features 23
amsmath 20
ANSI 48
Arguments to TEX macros 23
'auto' directories 50
Auto-Reveal 35
Automatic 50
Automatic Customization 50
Automatic Parsing 46
Automatic updating style hooks 51

B

Bad boxes 41
Bibliographies, adding 57
Bibliography 38
bibliography, completion 25
BibTEX 38
BibTEX, completion 25
'book.el' 53
Braces 16
Brackets 16

C

Changing font 17
Changing the parser 57
Chapters 13, 18
Character set 48
Checking 42
chktex 42
citations, completion of 25
cite, completion of 25
Commands 38
Completion 23
Controlling the output 42
Copying 1
Copyright 1
Current file 42
Customization 10
Customization, personal 10
Customization, site 10

D

Danish 48
Debugging 41
Default command 38
Defining bibliographies in style hooks 57
Defining environments in style hooks 56
Defining labels in style hooks 57
Defining macros in style hooks 53
Defining other information in style hooks 57
Deleting fonts 14, 17
Denmark 48
Descriptions 21
Display math mode 16
Distribution 1
Documents 44
Documents with multiple files 44
Dollars 16
Double quotes 16
Dutch 48

E

Enumerates 21
Environments 20
Environments, adding 56
Eqnarray 20
Equation 20
Equations 20
Errors 41

Europe . . . 48
European Characters . . . 48
Example of a style file . . . 53
Expansion . . . 23
External Commands . . . 38
Extracting TeX symbols . . . 50

F

Faces . . . 35
Figure environment . . . 21
Figures . . . 21
File Variables . . . 39
Filling . . . 27
Finding errors . . . 42
Finding the current file . . . 42
Finding the master file . . . 42
Floats . . . 21
Folding . . . 35, 37
Font Locking . . . 30
Font macros . . . 17
font-latex . . . 30
Fonts . . . 17
Formatting . . . 25, 27, 38
Forward search . . . 41
Free . . . 1
Free software . . . 1

G

General Public License . . . 1
Generating symbols . . . 50
Germany . . . 48
Global directories . . . 50
Global macro directory . . . 50
Global style hook directory . . . 50
Global TeX macro directory . . . 50
GPL . . . 1

H

Header . . . 38
Headers . . . 37
Hide Macros . . . 35
Holland . . . 48

I

Including . . . 44
Indentation . . . 25
Indenting . . . 25
Indexing . . . 38
Initialization . . . 10
Inputing . . . 44
Installation . . . 5
Internationalization . . . 48
Inverse search . . . 41
ISO 8859 Latin 1 . . . 48
ISO 8859 Latin 2 . . . 48
'iso-cvt.el' . . . 48
Itemize . . . 21
Items . . . 21

J

Japan . . . 49
Japanese . . . 49
jLaTeX . . . 49
jTeX . . . 49

K

Killing a process . . . 42

L

Label prefix . . . 20, 21
Labels . . . 20, 21
Labels, adding . . . 57
labels, completion of . . . 25
lacheck . . . 42
LaTeX . . . 38
Latin 1 . . . 48
Latin 2 . . . 48
License . . . 1
Literature . . . 38
Local style directory . . . 51
Local style hooks . . . 51
Local Variables . . . 39

M

Macro arguments . . . 23
Macro completion . . . 23
Macro expansion . . . 23
'macro.el' . . . 57
'macro.tex' . . . 57
Macros, adding . . . 53
Make . . . 5
makeindex . . . 38
Making a bibliography . . . 38
Making an index . . . 38
Many Files . . . 44
Master file . . . 42, 44
Matching dollar signs . . . 16
Math mode delimiters . . . 16
Mathematics . . . 23
MULE . . . 49
Multifile Documents . . . 44
Multiple Files . . . 44

N

National letters . . . 48
Next error . . . 41
Nippon . . . 49

O

Omega 40
Other information, adding 57
Outlining 35, 37
Output 42
Overfull boxes 41
Overview 37

P

Parsing errors 41
Parsing LaTeX errors 41
Parsing new macros 57
Parsing TeX 46, 50
Parsing TeX output 41
Personal customization 10
Personal information 51
Personal macro directory 51
Personal TeX macro directory 51
pLaTeX 49
Poland 48
Prefix for labels 20, 21
Previewing 40
Printing 38
Private directories 51
Private macro directory 51
Private style hook directory 51
Private TeX macro directory 51
Problems 42
Processes 42
pTeX 49

Q

Quotes 16
Quotes, fontification of 33

R

Redisplay output 42
Refilling 27
Reformatting 25, 27
Region 38
Region file 38
Reindenting 25
Reveal 35
Right 1
Running BibTeX 38
Running `chktex` 42
Running commands 38
Running `lacheck` 42
Running LaTeX 38
Running `makeindex` 38
Running TeX 38

S

Sample style file 53
Sectioning 13, 18
Sectioning commands, fontification of 31
Sections 13, 18, 37
Setting the default command 38
Setting the header 38
Setting the trailer 38
Site customization 10
Site information 50
Site initialization 10
Site macro directory 50
Site TeX macro directory 50
Source specials 41
Specifying a font 17
Starting a previewer 40
Stopping a process 42
'`style`' 53
Style 42
Style file 53
Style files 53
Style hook 53
Style hooks 53
Subscript, fontification of 34
Superscript, fontification of 34
Symbols 23
Syntax Highlighting 30

T

Tabify 46
Table environment 21
Tables 21
Tabs 46
TeX 38
TeX parsing 50
'`tex-site.el`' 10
Trailer 38

U

Untabify 46
Updating style hooks 51

V

Variables 39
Verbatim, fontification of 34
Viewing 40

W

Warranty 1
Wonderful boxes 41
Writing to a printer 38

X

'`x-compose.el`' 48
X-Symbol 48