GNU TEXMACS USER MANUAL

Table of contents

Chapter 1

Getting started

1.1. Conventions for this manual

Throughout the TEXMACS manual, menu entries will be typeset using a *sans serif* font, like in Document, File → Load or Text → Font shape → Italic. Keyboard input will be typeset in a *typewriter* font inside boxes, like in `C-s`. At the righthand side of menu entries, you see keystroke equivalents, when these are available. The following abbreviations are used for such keystrokes:

- `S-` For shift key combinations.
- `C-` For control key combinations.
- `A-` For alternate key combinations.
- `M-` For meta key combinations.
- `H-` For hyper key combinations.

For instance, `A-C-b` stands for `alt-ctrl-b`. Spaces inside keyboard shortcuts indicate multiple key-presses. For instance, `M-t N b` stands for `meta-t` `N` `b`.

The `alt`, `meta` and `hyper` keys are not available on all keyboards. On recent PC's, the `meta` key is often replaced by the `windows` key. In the case when one or several modifier keys are missing on your keyboard, you may use `escape` instead of `M-`, `escape escape` instead of `A-` and `F5`, `escape escape escape` or `A-C-` instead of `H-`. For instance, `escape w` is equivalent to `A-w`. You may also configure the keyboard modifiers in order to take full advantage out of the powerful set of keyboard shortcuts which is provided by TEXMACS.

Notice that the TEXMACS menus and keyboard behavior are *contextual*, i.e. they depend on the current mode (i.e. text mode or "math mode"), the current language and the position of the cursor inside your document. For instance, inside math mode, you have special keyboard shortcuts which are handy for typing mathematical formulas, but which are useless in text mode.

1.2. Configuring TEXMACS

When starting TEXMACS for the first time, the program automatically configures itself in a way which it thinks to be most suitable for you. For instance, TEXMACS will attempt to determine your systems settings for the language and the paper type of your printer. However, the automatic configuration may sometimes fail or you may want to use an alternative configuration. In that case, you should go to the Edit → Preferences menu and specify your preferences.

In particular, we recommend you to configure the desired "look and feel" of TEXMACS. By default, we use the EMACS look and feel, which ensures a limited compatibility of the TEXMACS keyboard shortcuts with those of EMACS. Also, TEXMACS comes with a powerful keyboard shortcut system, which attempts to optimize the use of the modifier keys like shift and control on your keyboard. However, on many X Window systems these modifier keys are not well configured, so that you may wish to redo this yourself. More details can be found in the section about the configuration of TEXMACS.

1.3. CREATING, SAVING AND LOADING DOCUMENTS

When launching TEXMACS without any command line options, the editor automatically creates a new document for you. You may also create a new document yourself using File → New. Newly created documents do not yet carry a name. In order to give them a name, you should click on File → Save as.

We recommend you to give documents a name immediately after their creation; this will avoid you to loose documents. It is also recommended to specify the global settings for your document when necessary. First of all, you may specify a document style like article, book or seminar using Document → Style. If you write documents in several languages, then you may want to specify the language of your document using Document → Language. Similarly, you may specify a paper type using Document → Page → Size.

After modifying your document, you may save it using File → Save. Old documents can be retrieved using File → Load. Notice that you can edit several documents in the same window using TEXMACS; you can switch between different *buffers* using Go.

1.4. PRINTING DOCUMENTS

You can print the current file using File → Print → Print all. By default, TEXMACS assumes that you have a 600dpi printer for a4 paper. These default settings can be changed in Preferences → Printer. You can also print to a postscript file using File → Print → Print all to file (in which case the default printer settings are used for creating the output) or File → Export → Postscript (in which case the printer settings are ignored).

When adequately configuring TEXMACS, the editor is guaranteed to be *wysiwyg*: the result after printing out is exactly what you see on your screen. In order to obtain full wysiwyg-ness, you should in particular select Document → Page → Type → Paper and Document → Page → Screen layout → Margins as on paper. You should also make sure that the characters on your screen use the same number of dots per inch as your printer. This rendering precision of the characters may be changed using Document → Font → Dpi. Currently, minor typesetting changes may occur when changing the dpi, which may globally affect the document through line and page breaking. In a future release this drawback should be removed.

Chapter 2

Writing simple documents

2.1. Generalities for typing text

As soon as you have performed the preparatory actions as explained above, you can start typing. The usual English characters and punctuation symbols can easily be obtained on most keyboards. Accented characters from foreign languages can systematically be obtained using the escape key. For instance, "é" is obtained by typing A-' e. Similarly, we obtain "à" via A-` a and so on. Long words at borders of successive lines are automatically hyphenated. In order to hyphenate foreign languages correctly, you should specify the language of the document in the menu Document → Language.

At the left hand side of the footer, you see the document style, the text properties at the current cursor position. Initially, it displays "text roman 12", which means that you type in text mode using a 12 point roman font and no document style. You can change the text properties (font, font size, color, language) in the Text menu. You can also change the text properties of the text you have already typed by selecting a region and then change the text properties in the Text menu. Some text properties can also be changed for all the document in the Document → Font and Document → Language menus.

At the right hand side of the footer, the character or object (like a change in the text properties) just before the cursor is displayed. We also display all environments which are active at the cursor position. This information should help you to orient yourself in the document.

2.2. Typing structured text

Usually, long documents have a structure: they are organized in chapters, sections and subsections, they contain different types of text, such as regular text, citations, footnotes, theorems, etc. After selecting a *document style* in Document → Style, TEXMACS takes care of specific layout issues, such as numbering of sections, pages, theorems, typesetting citations, footnotes and theorems in a nice way and so on.

Currently, four document standard styles have been implemented: letter, article, book and seminar. The seminar style is used for making transparencies. As soon as you have selected such a style, you can organize your text into sections (see Text → Section) and use specific *environments*. Examples of environments are theorem, proposition, remark and so on (see Text → Environment). Other examples are lists of items (see Text → Itemize) or numbered lists (see Text → Enumerate).

When you get more acquainted with TEXMACS, it is possible to add your own new environments in your own style file. Assume for instance that you often make citations and that you want those to appear in italic, with left and right margins of 1cm. Instead of manually changing the text and paragraph properties each time you make a citation, it is better to create a citation environment. Not only it will be faster to create a new citation when doing so, but it is also possible to systematically change the layout of your citations throughout the document just by changing the definition of the citation environment. The latter situation occurs for instance if you discover *a posteriori* that you prefer the citations to appear in a smaller font.

2.3. Content-based tags

The simplest examples of structure in a text are content-based tags. In Text → content tags you see a list of them. Content based tags indicate that a given portion of text is of a particular kind or that it serves a specific purpose. For instance, important text should be marked using the `strong` tag. Its default rendering uses a bold type face, like in this **strong text**. However, strong text might be rendered in a different way according to the document style. For instance, strong text may be rendered in a different color on transparencies for presentations. Here follows a short list of the most common content-based tags and their purpose:

Tag	Example	Purpose
`strong`	this is **important**	Indicate an important region of text
`em`	the *real* thing	Emphasize a region of text
`dfn`	A *gnu* is a horny beast	Definition of some concept
`samp`	the `ae` ligature æ	A sequence of literal characters
`name`	the LINUX system	The name of a particular thing
`person`	I am JORIS	The name of a person
`cite*`	Melville's *Moby Dick*	A bibliographic citation
`abbr`	I work at the C.N.R.S.	An abbreviation
`acronym`	the HTML format	An acronym
`verbatim`	the program said `hello`	Verbatim text like computer program output
`kbd`	Please type `return`	Text which should be entered on a keyboard
`code*`	`cout << 1+1;` yields 2	Code of a computer program
`var`	`cp src-file dest-file`	Variables in a computer program

Table 2.1. Some of the most common content-based tags.

2.4. Lists

Using Text → Itemize you may start an unnumbered list. You may either select a particular tag like • (bullets), – (dashes) or → (arrows) to indicate entries in the list or the default tag. Lists may be *nested* inside other tags, like in the following list:

- First item.
- Now comes the sublist:
  - A subitem.

- Another one.

- A final item.

The default tag is rendered in a different way depending on the level of nesting. At the outermost level, we used the • tag, at the second level ∘, and so on. When you are inside a list, notice that pressing `return` automatically starts a new item. If you need items which are several paragraphs long, then you may always use `S-return` in order to start a new paragraph.

Enumerate environments, which are started using Text → Enumerate, behave in a similar way as itemize, except that the items are numbered. Here follows an example of an enumeration which was started using Text → Enumerate → Roman:

I. A first item.

II. A second one.

III. And a last one.

The last type of lists are descriptive lists. They are started using Text → Description and allow you to describe a list of concepts:

Gnu. A hairy but gentle beast.

Gnat. Only lives in a zoo.

2.5. Environments

In a similar way as content-based tags, environments are used to mark portions of text with a special meaning. However, while content-based tags usually enclose small portions of text, environments often enclose portions that are several paragraphs long. Frequently used environments in mathematics are `theorem` and `proof`, like in the example below:

Theorem 2.1. *There exist no positive integers a, b, c and n with $n \geqslant 3$, such that $a^n + b^n = c^n$.*

Proof. I do not have room here to write the proof down. □

You may enter environments using Text → Environment. Other environments with a similar rendering as theorems are `proposition`, `lemma`, `corollary`, `axiom`, `definition`. You may use the `dueto` macro (entered using `\ d u e t o return`) in order to specify the person(s) to which the theorem is due, like in

Theorem 2.2. (Pythagoras) *Under nice circumstances, we have $a^2 + b^2 = c^2$.*

Other frequently used environments with a similar rendering as theorems, but which do not emphasize the enclosed text, are `remark`, `note`, `example`, `warning`, `exercise` and `problem`. The remaining environments `verbatim`, `code`, `quote`, `quotation` and `verse` can be used in order to enter multiparagraph text or code, quotations or poetry.

2.6. LAYOUT ISSUES

As a general rule, TEXMACS takes care of the layout of your text. Therefore, although we did not want to forbid this possibility, we do not encourage you to typeset your document visually. For instance, you should not insert spaces or blank lines as substitutes for horizontal and vertical spaces between words and lines; instead, additional space should be inserted explicitly using Insert → Space. This will make your text more robust in the sense that you will not have to reconsider the layout when performing some minor changes, which affect line or page breaking, or major changes, such as changing the document style.

Several types of explicit spacing commands have been implemented. First of all, you can insert rigid spaces of given widths and heights. Horizontal spaces do not have a height and are either stretchable or not. The length of a stretchable spaces depends on the way a paragraph is hyphenated. Furthermore, it is possible to insert tabular spaces. Vertical spaces may be inserted either at the start or the end of a paragraph: the additional vertical space between two paragraphs is the maximum of the vertical space after the first one and the vertical space before the second one (contrary to TEX, this prevents from superfluous space between two consecutive theorems).

As to the paragraph layout, the user may specify the paragraph style (justified, left ragged, centered or right ragged), the paragraph margins and the left (resp. right) indentation of the first (resp. last) line of a paragraph. The user also controls the spaces between paragraphs and successive lines in paragraphs.

You can specify the page layout in the Document → Page menu. First of all, you can specify the way pages are displayed on the screen: when selecting "paper" as page type in Document → Page → Type, you explicitly see the page breaks. By default, the page type is "papyrus", which avoids page breaking during the preparation of your document. The "automatic" page type assumes that your paper size is exactly the size of your window. The page margins and text width are specified in Document → Page → Layout. Often, it is convenient to reduce the page margins for usage on the screen; this can be done in Document → Page → Screen layout.

2.7. THE FONT SELECTION SYSTEM

In TEXMACS, fonts have five main characteristics:

- Its name (roman, pandora, concrete, etc.).
- Its family (roman, typewriter or sans serif).
- Its size (a base size (in points) and a relative size (normal, small, etc.).
- Its series (bold, medium or light).
- Its shape (right, italic, small caps, etc.).

Notice that in the font selection system of LATEX 2ε, the font name and family are only one (namely, the family). Notice also that the base font size is specified for the entire document in Document → Font → Size.

2.8. Mastering the keyboard

2.8.1. General prefix rules

Since there are many keyboard shortcuts, it is important to have some ways of classifying them in several categories, in order to make it easier to memorize them. As a general rule, keyboard shortcuts which fall in the same category are indentified by a common prefix. The main such common prefixes are:

`C-`x Control key based shortcuts are used for frequently used editing commands. They depend very much on the "look and feel" in Edit → Preferences. For instance, if you use an Emacs-compatible look and feel, then the shortcuts of the form `C-`x correspond to Emacs commands, like `C-y` for pasting text.

`A-`x The alternate key is used for commands which depend on the mode that you are in. For instance, `A-s` produces **strong** text in text mode and a square root $\sqrt{}$ in math mode. Notice that `escape escape` is equivalent to `A-`.

`M-`x The meta key is used for general purpose TeXmacs commands, which can be used in all modes. For instance, `M-!` produces a label. It is also used for additional editing commands, like `A-w` for copying text if you use the Emacs look and feel. Notice that `escape` is equivalent to `M-`.

`H-`x The user keyboard modifier key is used for producing special symbols like Greek characters in math mode. You may configure your keyboard so as to let caps-lock play the rôle of the hyper key. The `F5` is equivalent to `M-`.

We recall that the particular modifier keys which are used in order to obtain the `M-` and `H-` prefixes can be configured in Edit → Preferences.

2.8.2. Some fundamental keyboard shortcuts

Some standard keyboard actions which are valid in all modes are:

`S-return` always starts a new paragraph.

`C-backspace` remove an object or environment.

`A-space` insert a small space.

`A-S-space` insert a small negative space.

`A-A-home` manually set start of the selection.

`A-A-end` manually set end of the selection.

`M-<` go to the start of the document.

`M->` go to the end of the document.

2.8.3. Keyboard shortcuts for text mode

The main keyboard actions available in text mode are `$` in order to enter math mode and the following shorthands to create accented characters:

Shortcut		Example		Shortcut		Example	
`A-'`	Acute ´	`A-' e`	é	`A-‘`	Grave	`A-‘ e`	è
`A-^`	Hat ˆ	`A-^ e`	ê	`A-^`	Umlaut ¨	`A-" e`	ë
`A-~`	Tilde ˜	`A-~ a`	ã	`A-C`	Cedilla ¸	`A-C c`	ç
`A-U`	Breve ˘	`A-U g`	ğ	`A-V`	Check ˇ	`A-V s`	š
`A-O`	Above ring ˚	`A-O a`	å	`A-.`	Above dot ˙	`A-. z`	ż
`A-H`	Hungarian ˝	`A-H o`	ő				

Table 2.2. Typing accented characters with TEXMACS.

The special characters ß, SS, œ, Œ, æ and Æ are obtained by typing `S-F5 s`, `S-F5 S`, `S-F5 o e`, `S-F5 A E`, `S-F5 a` resp. `S-F5 A`.

In French, you may also use the special keybindings `< <` and `> >` in order to obtain the French *guillemets*. In Spanish, the opening exclamation and question marks are obtained by typing `! *` or `! ‘` resp. `? *` or `? ‘`.

2.8.4. Hybrid commands and LaTeX simulation

TEXMACS allows you to enter LaTeX commands directly from the keyboard as follows. You first hit the `\`-key in order to enter the hybrid LaTeX/TEXMACS command mode. Next you type the command you wish to execute. As soon as you finished typing your command, the left footer displays something like

```
<return>: action to be undertaken
```

When you hit the `return`-key at this stage, your command will be executed. For instance, in math-mode, you may create a fraction by typing `\ f r a c return`.

If the command you have typed is not a (recognized) LaTeX command, then we first look whether the command is an existing TEXMACS macro, function or environment (provided by the style file). If so, the corresponding macro expansion, function application or environment application is created (with the right number of arguments). Otherwise, it is assumed that your command corresponds to an environment variable and we ask for its value. The `\`-key is always equivalent to one of the commands `M-i l`, `M-i e`, `M-i a`, `M-i #` or `M-i v`.

2.8.5. Dynamic objects

Certain more complex objects can have several *states* during the editing process. Examples of such *dynamic objects* are labels and references, because the appearance of the reference depends on a dynamically determined number. Many other examples of dynamic markup can be found in the documentation about writing style files.

When entering a dynamic object like a label using `M-!`, the default state is *inactive*. This inactive state enables you to type the information which is relevant to the dynamic object, such as the name of the label in our case. Certain dynamic objects take an arbitrary number of parameters, and new ones can be inserted using `tab`.

When you finished typing the relevant information for your dynamic object, you may type `return` in order to *activate* the object. An active dynamic object may be deactivated by placing your cursor just behind the object and hitting `backspace`.

2.8.6. Customization of the keyboard

It is possible for the user to modify the keyboard behaviour. In order to do so, we suggest first to look at the files in the directory `$TEXMACS_PATH/progs/keyboard`, where the standard keyboard behaviour is defined. Then you may redefine the keyboard behaviour in your private initialization file.

Chapter 3

Mathematical formulas

In order to type mathematical formulas, you should first enter "math mode" by pressing the `$`-key or by inserting an equation (using Insert → Mathematics → Equation). In math mode, you have specific commands and key-combinations to type mathematical symbols and formulas. For instance, the `H-` prefix can be used in order to enter Greek symbols (recall that `H-` is equivalent to `F5`, `escape escape escape` or `A-C-`).

The editor favors typing mathematics with a certain meaning. This feature, which will be developed more in future releases, is useful when communicating with a computer algebra package. At this moment, you should for instance explicitly type the multiplication `*` between symbols a and b. By default, typing `a b` will yield ab and not $a\,b$.

3.1. Main mathematical constructs

The main mathematical objects are created using the `A-` prefix as follows:

Shortcut	Purpose	Example
`A-$`	Text	$L = \{x \mid x \text{ is sufficiently large}\}$
`A-f`	Fractions	$\frac{a}{b+c}$
`A-s`	Square roots	$\sqrt{x+y}$
`A-S`	n-th Roots	$\sqrt[3]{x^3+y^3}$
`A-n`	Negations	$\frac{a/}{b \not= c}$

Table 3.1. Creation of major mathematical markup.

Primes, subscripts and superscripts are created as follows:

Shortcut	Purpose	Example
`'`	Primes	f' or $(g+h)'''$
`‘`	Back-primes	$\backprime f$
`_`	Subscripts	x_n or x_{i_3}
`^`	Superscripts	x^2, x_n^2 or $\mathrm{e}^{\mathrm{e}^x}$
`A-l _`	Left subscripts	${}_2x$
`A-l ^`	Left superscripts	${}^{\pi}x$ or ${}^{*}_{*}\mathrm{He}^{*}_{*}$

Table 3.2. Creation of primes, subscripts and superscripts

3.2. Typing mathematical symbols

The Greek characters are obtained in TEXMACS by combining the hyper modifier key `H-` with a letter. For instance, `H-a` yields α and `H-G` yields Γ. Recall that the `F5` key is equivalent to `H-`, so that ρ can also be obtained by typing `F5 r`. Similarly, `F6`, `F7`, `F8` and `S-F6` can be used in order to type bold, calligraphic, fraktur and blackboard bold characters. For instance, `F8 m` yields $\mathfrak{m}$, `S-F6 R` yields $\mathbb{R}$ and `F6 F7 Z` yields $\boldsymbol{\mathcal{Z}}$.

Greek characters can also be obtained as "variants" of Latin characters using the `tab`-key. For instance, `p tab` yields π. The `tab`-key is also used for obtaining variants of the Greek letters themselves. For instance, both `H-p tab` and `p tab tab` yield ϖ.

Many other mathematical symbols are obtained by "natural" key-combinations. For instance, `- >` yields $\to$, `- - >` yields $\longrightarrow$ and `> =` yields $\geqslant$. Similarly, `| -` yields $\vdash$, `| - >` yields $\mapsto$ and `- > < -` yields $\rightleftarrows$. Some general rules hold in order to obtain variants of symbols:

- `tab` is the main key for obtaining variants. For instance, `> =` yields $\geqslant$, but `> = tab` yields $\geq$. Similarly, `< tab` yields $\prec$, `< tab =` yields $\preccurlyeq$ and `< tab = tab` yields $\preceq$. Also, `P tab` yields $\wp$ and `e tab` yields the constant $\mathrm{e}=\exp(1)$. You may "cycle back" using `S-tab`.

- `@` is used for putting symbols into circles or boxes. For instance, `@ +` yields $\oplus$ and `@ x` yields $\otimes$. Similarly, `@ tab +` yields $\boxplus$.

- `/` is used for negations. For instance, `= /` yields $\neq$ and `< = /` yields $\nleqslant$. Notice that `< = tab tab /` yields $\nleqslant$, while `< = tab tab / tab` yields $\lneqq$.

- `!` is used after arrows in order to force scripts to be placed above or below the arrow. For instance, `- - > ^ x` yields $\longrightarrow^{x}$, but `- - > ! ^ x` yields $\xrightarrow{x}$.

Several other symbols which cannot be entered naturally in the above way are obtained using the `S-F5` prefix. Here follows a short table of such symbols:

Shortcut	Symbol	Shortcut	Symbol
`S-F5 a`	$\amalg$		
`S-F5 n`	$\cap$	`S-F5 u`	$\cup$
`S-F5 v`	$\vee$	`S-F5 w`	$\wedge$

Table 3.3. Some symbols which cannot be obtained using general rules in a natural way.

3.3. Typing big operators

The following key-combinations are used in order to create big symbols:

Shortcut	Result	Shortcut	Result
S-F5 I	$\int$	S-F5 O	$\oint$
S-F5 P	$\prod$	S-F5 A	$\coprod$
S-F5 S	$\sum$	S-F5 @ +	$\bigoplus$
S-F5 @ x	$\bigotimes$	S-F5 @ .	$\bigodot$
S-F5 U	$\bigcup$	S-F5 N	$\bigcap$
S-F5 V	$\bigvee$	S-F5 W	$\bigwedge$

Table 3.4. Big mathematical operators.

The big integral signs admit two variants, depending on where you want to place subscripts and superscripts. By default, the scripts are placed as follows:

$$\int_0^\infty \frac{\mathrm{d}x}{1+x^2}.$$

The alternative rendering "with limits"

$$\int\limits_0^\infty \frac{\mathrm{d}x}{1+x^2}.$$

is obtained using S-F5 L I. Similarly, you may type S-F5 L O in order to obtain $\oint$ with limits.

3.4. TYPING LARGE DELIMITERS

Large delimiters are created as follows:

Shortcut	Result	Shortcut	Result
A-(	$($	A-)	$)$
A-[	$[$	A-]	$]$
A-{	$\{$	A-}	$\}$
A-<	$\langle$	A->	$\rangle$
A-/	$/$	A-\	$\backslash$

Table 3.5. Keyboard shortcuts for large delimiters.

In TEXMACS, large delimiters may either be "left delimiters", "right delimiters" or "middle delimiters". By default, (, [, { and ⟨ are left delimiters,),], } and ⟩ are right delimiters and |, / and \ are middle delimiters. But there status can be changed using the A-l, A-r and A-m key combinations. For instance, A-l) produces), considered as a large left delimiter.

In TEX and LATEX, "middle delimiters", or "separators" do not exist; they are used for producing the vertical bars in formulas like

$$\left\langle \frac{a}{b+c} \middle| \frac{p}{q+r} \middle| \frac{a}{b+c} \right\rangle.$$

There may be as many middle delimiters between a left and a right delimiter as one wishes.

3.5. Wide mathematical accents

The table below how to type mathematical accents above symbols or entire formulas. Indeed, some of these accents automatically become as wide as the formulas below them.

Shortcut	Example	Wide variant	Shortcut	Result
A-~	$\tilde{x}$	$\widetilde{x+y}$	A-'	$\acute{x}$
A-^	$\hat{x}$	$\widehat{x+y}$	A-`	$\grave{x}$
A-B	$\bar{x}$	$\overline{x+y}$	A-.	$\dot{x}$
A-V	$\vec{x}$	$\overrightarrow{\mathrm{AB}}$	A-"	$\ddot{x}$
A-C	$\check{x}$	$\widecheck{x+y}$		
A-U	$\breve{x}$	$\underbrace{x+y}$		

Table 3.6. Keyboard shortcuts for wide mathemarical accents.

Chapter 4

Tabular material

4.1. Creating tables

In order to create a table, you may either use Insert → Table or one of the following keyboard shorthands:

`M-t N t` Create a regular table.

`M-t N T` Create a regular table whose cells are centered.

`M-t N b` Create a regular "block", whose cells are separated by lines.

`M-t N B` Create a block whose cells are centered.

In math mode, a few other table-like structures are provided:

`M-t N m` Create a matrix.

`M-t N d` Create a determinant.

`M-t N c` Create a choice list.

The `\eqnarray*` environment is also a special kind of table-like structure, which extends over the entire line. You may start a list of equations using Insert → Mathematics → Equations.

When starting a new table, its size is minimal (usually 1×1) and its cells are empty. New rows and columns are inserted using the `A-left`, `A-right`, `A-up` and `A-down` shorthands. For instance, `A-right` creates a new column at the right of the current cursor position. You may also start a new row below the current cursor position by hitting `return`.

4.2. The formatting mode

In TEXMACS, arbitrary blocks of cells in the table may be formatted in particular ways. For instance, you may give individual cells a background color, but you may also decide an entire column to be horizontally centered. By default, formatting commands operate on individual cells, but this may be changed via Table → Cell operation mode. The following operation modes are available:

`M-t m c` Operate on individual cells.

`M-t m h` Operate on rows.

`M-t m v` Operate on columns.

`M-t m t` Operate on the entire table.

It is also possible to select a block of cells using the mouse and perform a single operation on that rectangle.

4.3. Specifying the cell and table alignment

The most frequent formatting operation is the horizontal or vertical alignment of a block of cells. You may use the `M-←`, `M-→`, `M-↑` and `M-↓` keystrokes to quickly align more to the left, right, top or bottom.

A specific alignment can also be selected in the Table → Horizontal cell alignment and Table → Vertical cell alignment menus. Alternatively, you may use keyboard shorthands of the types `M-t h` x and `M-t v` x for horizontal resp. vertical alignment.

Similarly, you may specify how the table itself should be aligned with respect to the surrounding text. This is either done via the Table → Horizontal table alignment and Table → Vertical table alignment submenus, or using keyboard shorthands of the form `M-t H` x or `M-t V` x. Here x represents `l` for "left", `c` for "centered", `r` for "right", `b` for "bottom" and `t` for "top".

4.4. Specifying the cell and table size

Using Table → Cell width → Set width resp. Table → Cell height → Set height you may specify the width or height of a cell. In fact, the specified width (or height) may be taken into account in three different ways:

Minimum mode. The actual width of the cell will be the minimum of the specified width and the width of the box inside the cell.

Exact mode. The width of the cell will be precisely the specified one.

Maximum mode. The actual width of the cell will be the maximum of the specified width and the width of the box inside the cell.

The border width and the cell padding (to be explained below) are taken into account in the size of the box inside the cell.

You may also specify the width and the height of the entire table in Table → Special table properties. In particular, you may specify the table to run over the entire width of a paragraph. When specifying a width (or height) for the entire table, you may specify how the unused space is distributed over the cells using Table → Special cell properties → Distribute unused space. By default, the unused space is equally distributed.

4.5. Borders, padding and background color

You may specify the border widths and padding spaces of a cell in all possible four directions: on the left, on the right, at the bottom and at the top (see Table → Cell border). You have keyboard shorthands of the forms `M-t b` x and `M-t p` x in order to specify border widths and cell padding.

The default border width for cells in the block environment is `1ln`, i.e. the standard line width in the current font (like the width of a fraction bar). This width occurs at the right and the bottom of each cell (except when the cell is on the first row or column). The default horizontal cell padding is `1spc`: the width of a white space in the current font. The default vertical cell padding is `1sep`: the standard minimal separation between two close boxes.

Cells may be given a background color via Table → Cell background color.

The entire table may also be given a border and a table padding in Table → Special table properties → Border. In this case, the padding occurs outside the border.

4.6. ADVANCED TABLE FEATURES

In the menus, you also find some other more special features for tables. Very briefly, these include the following:

- Change the "span" of a cell and let it run over its neighbouring cells on its right and below.
- Creation of entire subtables inside cells.
- Correction of the depth and height of text, in order to let the baselines match.
- Horizontal hyphenation of cell contents and vertical hyphenation of the entire table.
- Gluing several rows and/or columns together, so that the glued cells become "part of the borders" of the remaining cells.
- Disactivation of the table, in order to see its "source code".
- Setting the "extension center" of a table. From now on, the formatting properties of this cell will be used for new cells created around this center.
- Specification of the minimal and maximum size of a table, which will be respected during further editing. (this is mainly useful when creating table macros).

Currently, all tables come inside an environment like `tabular`, `block`, `matrix`, etc. When creating your own table macros, you may use Table → Special table properties → Extract format to extract the format from a given table.

Chapter 5

Links and automatically generated content

5.1. Creating labels, links and references

You may create a new inactive label using M-! or Insert → Link → Label and a reference to this label using M-? or Insert → Link → Reference. Be careful to put the label at a point where its number will be correct. When labeling sections, the recommended place is just after the section name. When labeling equations, the recommended place is at the start inside the equation.

It is possible to create hyperlinks to other documents using M-i > or Insert → Link → Hyperlink. The first field of the hyperlink is the associated text, which is displayed in blue when activated. The second field contains the name of a document, which may be on the web. As is usual for hyperlinks, a link of the form *#label* points to a label in the same document and a link of the form *url#label* points to a label in the document located at *url*.

In a similar fashion, an action may be associated to a piece of text or graphics using M-i * or Insert → Link → Action. The second field now contains a Guile/Scheme script command, which is executed whenever you double click on the text, after its activation. For security reasons, such scripts are not always accepted. By default, you are prompted for acceptation; this default behaviour may be changed in Options → Security. Notice that the Guile/Scheme command

```
(system "shell-command")
```

evaluates `shell-command` as a shell command.

Finally, you may directly include other documents inside a given document using M-i i or Insert → Link → Include. This allows you for instance to include the listing of a program in your text in such a way that your modifications in your program are automatically refelcted in your text.

5.2. Inserting images

You can include images in the text using the menu Insert → Image. Currently, TEXMACS recognizes the `ps`, `eps`, `tif`, `pdf`, `pdm`, `gif`, `ppm`, `xpm` and `fig` file formats. Here, `gs` (i.e. ghostscript) is used to render postscript images. If ghostscript has not yet been installed on your system, you can download this package from

```
www.cs.wisc.edu/~ghost/index.html
```

Currently, the other file formats are converted into postscript files using the scripts `tiff2ps`, `pdf2ps`, `pnmtops`, `giftopnm`, `ppmtogif`, `xpmtoppm`. If these scripts are not available on your system, please contact your system administrator.

By default, images are displayed at their design size. The following operations are supported on images:

- Clipping the images following a rectangle. The lower left corner of the default image is taken as the origin for specifying a rectangle for clipping.
- Resizing an image. When specifying a new width, but no height at the prompt (or vice versa), the image is resized so as to preserve the aspect ration.
- Magnifying the image. An alternative way to resize an image, by multiplying the width and the height by a constant.

We also included a script to convert pictures, with optional LaTeX formulas in it, into encapsulated postscript. In order to include a LaTeX formula in an `xfig` picture, we recall you should enter the formula as text, while selecting a LaTeX font and setting the special flag in the text flags.

5.3. Generating a table of contents

It is very easy to generate a table of contents for your document. Just put your cursor at the place where you want your table of contents and click on Insert → Automatic → Table of contents.

In order to generate the table of contents, you should be in a mode where page breaks are visible (select paper in Document → Page → Type), so that the appropriate references to page numbers can be computed. Next, use Document → Update → Table of contents or Document → Update → All to generate the table of contents. You may have to do this several times, until the document does not change anymore. Indeed, the page numbers may change as a result of modifications in the table of contents!

5.4. Compiling a bibliography

At the moment, TeXmacs uses `bibtex` to compile bibliographies. The mechanism to automatically compile a bibliography is the following:

- Write a `.bib` file with all your bibliographic references. This file should have the format of a standard bibliography file for LaTeX.
- Use Insert → Link → Citation and Insert → Link → Invisible citation to insert citations, which correspond to entries in your `.bib` file.
- At the place where your bibliography should be compiled, click on Insert → Automatic → Bibliography. At the prompt, you should enter a `bibtex` style (such as `plain`, `alpha`, `abbrv`, etc.) and your `.bib` file.
- Use Document → Update → Bibliography in order to compile your bibliography.

5.5. Generating an index

For the generation of an index, you first have to put index entries in your document using Insert → Link → Index entry. At a second stage, you must put your cursor at the place where you want your index to be generated and click on Insert → Automatic → Index. The index is than generated in a similar way as the table of contents.

In the Insert → Link → Index entry menu, you find several types of index entries. The simplest are "main", "sub", "subsub", which are macros with one, two and three arguments respectively. Entries of the form "sub" and "subsub" may be used to subordinate index entries with respect to other ones.

A complex index entry takes four arguments. The first one is a key how the entry has to be sorted and it must be a "tuple" (created using `M-i <`) whose first component is the main category, the second a subcategory, etc. The second argument of a complex index entry is either blank or "strong", in which case the page number of your entry will appear in a bold typeface. The third argument is usually blank, but if you create two index entries with the same non-blank third argument, then this will create a "range" of page numbers. The fourth argument, which is again a tuple, is the entry itself.

It is also possible to create an index line without a page number using "interject" in Insert → Link → Index entry. The first argument of this macro is a key for how to sort the index line. The second argument contains the actual text. This construct may be useful for creating different sections "A", "B", etc. in your index.

5.6. Compiling a glossary

Glossaries are compiled in a similar way as indexes, but the entries are not sorted. A "regular" glossary entry just contains some text and a page number will be generated for it. An "explained" glossary entry contains a second argument, which explains the notation. A "duplicate" entry may be used to create a page number for the second occurence of an entry. A glossary line creates an entry without a page number.

5.7. Books and multifile documents

When a document gets really large, you may want to subdivide it into smaller pieces. This both makes the individual pieces more easily reusable in other works and it improves the editor's responsiveness. An entire file can be inserted into another one using Insert → Link → Include. In order to speed up the treatment of included documents, they are being buffered. In order to update all included documents, you should use Tools → Update → Inclusions.

When writing a book, one usually puts the individual chapters in files `c1.tm`, `c2.tm` until `cn.tm`. One next creates one file `book.tm` for the whole book, in which the files `c1.tm`, `c2.tm` until `cn.tm` are included using the above mechanism. The table of contents, bibliography, etc. are usually put into `book.tm`.

In order to see cross references to other chapters when editing a particular chapter `ci.tm`, one may specify `book.tm` as a "master file" for the files `c1.tm` to `cn.tm` using Document → Master → Attach. Currently, the chapter numbers themselves are not dealt with by this mechanism. You may want to manually assign the environment variable `chapternr` at the start of each chapter file in order to get the numbering right when editing.

Chapter 6

Editing tools

6.1. Cut and paste

You can select text and formulas by maintaining the left mouse button. In order to delete the selected region, use Edit → Cut. In order to copy the selected region, first click on Edit → Copy. Next, paste it as many times as you want to the location of your cursor, using Edit → Paste. Alternatively, you may copy a selected region using the middle mouse button.

It is also possible to the change text properties of a selected region. For instance, in order to transform some black text in red, you select it using the left mouse button and click on Text → Color → Red. Similarly, if you select a formula and you click on Insert → Mathematics → Fraction, then the formula becomes the numerator of some fraction.

When using the copy and paste mechanism to communicate with other applications, text is copied and pasted using the $\TeX_{\text{MACS}}$ data format. You may specify other import and export formats using Edit → Import resp. Edit → Export. By default, copying and pasting uses the primary text buffer. Using Edit → Copy to and Edit → Paste from, you may specify as many other buffers as you like.

6.2. Search and replace

You can start searching text by pressing C-s or Edit → Search. During a search, the "search string" is displayed at the left hand side of the footer. Each character you type is appended to this search string and the next occurrence of it is surrounded by a red box. When pressing C-s a second time during a search, the next occurrence is being searched. A beep indicates that no more occurrences were found in the document; pressing C-s will continue the search at the beginning of your document. You may press backspace in order to undo key presses during a search.

Usually, text is being searched for in a forward manner, starting from the current cursor position. You may also search backwards, using C-r. During a search, only text in the same mode and the same language will be found, as those which are active at the position where you started your search. In other words, when searching an x in math-mode, you will not find any x's in the ordinary text. As a current limitation, the search string can only contain ordinary text and no math-symbols or more complicated structured text.

A query replace is started by pressing C-= or Edit → Replace. You are prompted for a string which is to be replaced and the string by which to replace. At each occurrence of the string to be replaced you are prompted and you have to choose between replacing the string (y), not replacing it (n) and replace this and all further occurrences (a). Like in the case of searching, the query-replace command is mode and language sensitive.

6.3. Spell checking

If the program `ispell` has been installed on your system, then you may use it to check your text for misspelled words by pressing `M-$` or Edit → Spell. Notice that you might have to verify that the dictionaries corresponding to the languages in which your texts have been written have been installed on your system; this is usually the case for English.

When you launch the spell checker (either on the whole text or a selected region), you will be prompted at each misspelled word and the footer displays the available options:

a) Accepts the misspelled word and all its future occurrences in the text.

r) Replace the misspelled word by a correction you have to enter.

i) Indicate that the "misspelled" word is actually correct and that it has to be inserted in your personal dictionary.

1-9) Several suggested corrections for your misspelled word.

Notice that `ispell` just checks for misspelled words. No grammatical faults will be detected.

When starting the spell checker, it will use the dictionary of the language which is active at the current cursor position (or the start of a selection). Only text in that language will be checked for. If your document contains text in several languages, then you will have to launch the spell checker once for each language being used.

6.4. Undo and redo

It is possible to gradually undo the changes you made in a document from the moment that you launched TeXmacs. This can be done via Edit → Undo or using the keystrokes `M-[` or `C-/`. Undone changes can be "redone" using Edit → Redo or `M-]`.

In order to save memory, the number of successive actions which can be undone is limited to 100 (by default). It is possible to increase this number by adding a command like

```
(set-maximal-undo-depth 1000)
```

in our personal initialization file (see Help → Scheme). When specifying a negative number as your maximal undo depth, any number of actions can be undone.

Chapter 7

Advanced layout features

7.1. Flows

Complex documents often contain footnotes or floating objects, which appear differently on pages as the main text. In fact, the content of such complex documents use several *flows*, one for the main text, one for the footnotes, one for floats, and still another one for two column text. The different flows are broken across pages in a quite independent way.

In order to insert a footnote, you may use Insert → Page insertion → Footnote. The number of columns of the text may be changed in Paragraph → Number of columns.

7.2. Floating objects

Floating objects are allowed to move on the page independently from the main text. Usually they contain figures or tables which are too large to nicely fit into the main text. A floating object may be inserted using Insert → Page insertion → Floating object.

You may also create a floating object and directly insert a figure or table inside it using Insert → Page insertion → Floating figure resp. Insert → Page insertion → Floating table. However, sometimes you might want to insert several smaller figures or tables inside one floating object. You may do this using Insert → Image → Small figure resp. Insert → Table → Small table.

After creating a floating object, you may control its position using Insert → Position float (when inside the float). You may specify whether you allow the floating object to appear at the top of the page, at the bottom, directly in the text, or on the next page. By default, the float may appear everywhere. However, a floating object will never appear inside the main text at less than three lines from the bottom or the top of a page.

7.3. Page breaking

The page breaking may be controlled very precisely by the user inside Document → Page → Breaking. In the submenu Algorithm, you may specify the algorithm being used. Professional page breaking is best in print, but may slow down the editing when being used interactively in paper mode. Sloppy page breaking is fastest and medium is professional except for multicolumn material, for which the professional algorithm is significantly slower.

You may also allow the page breaking algorithm to enlarge or reduce the length of pages in exceptional cases in the submenu Limits. The stretchability of vertical space between paragraphs and so may be specified in Flexibility. The factor 1 is default; a smaller factor enforces a more rigid spacing, but the quality of the breaks may decrease.

Chapter 8

Using GNU TEXMACS as an interface

An important feature of TEXMACS is it's ability to communicate with extern systems in shell-like sessions. Typically, it is possible to evaluate commands of an extern computer algebra system inside such a session and display the results in a nice, graphical way. It is also possible to evaluate shell commands and Scheme programs inside such sessions.

8.1. Sessions

8.1.1. Basic usage

A session can be started from the Insert → Session menu. A session consists of a sequence of input and output environments and possible text between them. When pressing `return` inside an input environment of a session, the text inside the environment is evaluated and the result is displayed in an output environment.

When entering a command in a session, the application attempts to execute it. Several commands may be launched concurrently in the same document, but the output will only be active in the session where the cursor is and at the place of the cursor. Therefore, we recommend to use different buffers for parallel executions. Executions may be interrupted from the iconbar. it is also possible to disconnect (close) the application; in that case no further commands can be executed in the corresponding session.

In the second iconbar you also have a few buttons for selecting mathematical input and interrupting execution. When implemented for the given system, mathematical input allows you to type the input in a graphical, two dimensional form. The other two buttons allow you to interrupt execution of a particular command (although this does not work well for certain systems) or to disconnect the extern system. When pressing return in the input of a non-connected system, the system will be restarted automatically.

8.1.2. Customizing the session styles

Each session environment takes two arguments: the programming language and a name for the session. All input to be evaluated is redirected to the package which implements the programming languages and the session name is passed as an extra argument.

It is possible to redefine the screen layout of sessions as follows. For each session in programming language 'p', you then have to modify the environments 'input-p' and 'output-p' which respectively correspond to the layout of the input and the output. The first argument to 'input-p' is the prompt for the input.

It is possible to give sessions a name (the default name being "default"): by clicking on Insert → Session → Other, one both has to enter a session type and a session name. Different sessions which have the same type and the same name share correspond to one instance of the application being executed. Consequently, such sessions share a common environment. By using different session names, one may concurrently launch several instances of the same application.

8.2. SUPPORTED SYSTEMS

When taking a look at the Insert → Session menu, only those systems which are actually installed on your system will show up. The only exceptions are shell sessions and scheme sessions, which are always available.

Below, you find a short list of free computer algebra systems which have been interfaced with TEXMACS. There also exist interfaces with several proprietary interfaces, but you should look at the documentation of those systems for more information.

8.2.1. Shell sessions and scheme sessions

In a "shell session" it is possible to evaluate shell commands. All input and output is verbatim. No particular command-line utilities (such as completion mechanisms) have been implemented yet. The output of the shell command is displayed gradually as the program executes.

In a "SCHEME session" you can evaluate GUILE/SCHEME programs. The input should be verbatim text. The input is evaluated and the result is displayed. No gradual output mechanism has been implemented yet for SCHEME session.

8.2.2. Giac

GIAC Is A Computer algebra system, which can be downloaded from

`http://www-fourier.ujf-grenoble.fr/~parisse/english.html`

8.2.3. GTybalt

GTYBALT is a free computer algebra system which is built on top of GINAC, CLN and a program to interpret C and C++ commands. For more information, see

`http://www.fis.unipr.it/~stefanw/gtybalt.html`

8.2.4. Macaulay 2

MACAULAY 2 is a new software system devoted to supporting research in algebraic geometry and commutative algebra. The software is available now in source code for porting, and in compiled form for LINUX, SUN OS, SOLARIS, WINDOWS, and a few other unix machines. You can get it from

`http://www.math.uiuc.edu/Macaulay2`

8.2.5. Maxima

Maxima is not alone one of the oldest and best computer algebra systems around, it is also one of the only general purpose systems for which there is a free implementation. You can get it from

`http://www.ma.utexas.edu/users/wfs/maxima.html`

The supported version is GCL-based Maxima 5.6. For CLisp-based Maxima 5.6, edit your `tm_maxima` and replace `-load` by `-i`. For Maxima 5.9-pre, replace `-load` by `-p`. Known problems:

- If you press `return` when a statement is not complete (typically, terminated by `;` or `$`), the interface will hang.
- If you cause the Lisp break prompt to appear, the interface will hang.
- The command `info` is not supported (it is defined in the underlying Lisp, and difficult to support portably).
- Some commands in the debugger work, but some (including `:c`) don't work, nobody knows why.
- The command `load` sometimes behaves mysteriously.

8.2.6. Pari

Pari is a software package for computer-aided number theory. It consists of a C library, `libpari` (with optional assembler cores for some popular architectures), and of the programmable interactive `gp` calculator. You can download Pari from

`ftp://megrez.math.u-bordeaux.fr/pub/pari`

You will need a version newer than Pari-2.1.0 for use from inside TeXmacs (for an already installed Pari-system, type `gp --version`).

8.2.7. Qcl

QCL is a high level, architecture independent programming language for quantum computers, with a syntax derived from classical procedural languages like C or Pascal. This allows for the complete implementation and simulation of quantum algorithms (including classical components) in one consistent formalism. The TeXmacs interface is mainly useful for displaying quantum states in a readable way. For more information, see

`http://tph.tuwien.ac.at/~oemer/qcl.html`

Starting from 1.0.0.8, TeXmacs supports QCL 0.4.3 or newer. Users of older versions should upgrade.

8.2.8. Yacas

Yacas is, as it's name suggest, yet another computer algebra system. Things implemented include: arbitrary precision, rational numeric, vector, complex, and matrix computations (including inverses and determinants and solving matrix equations), derivatives, solving, Taylor series, numerical solving (Newtons method), and a lot more non-mathematical algorithms. The language natively supports variables and user-defined functions. There is basic support for univariate polynomials, integrating functions and tensor calculations. You can get Yacas at

`http://www.xs4all.nl/~apinkus/yacas.html`

Chapter 9

TEXMACS style files

9.1. TEXMACS style files

One of the fundamental strengths of TEXMACS is the possibility to write your own style files and packages. The purpose of style files is multiple:

- They allow the abstraction of repetitive elements in texts, like sections, theorems, enumerations, etc.
- They form a mechanism which allow you to structure your text. For instance, you may indicate that a given portion of your text is an abbreviation, a quotation or "important".
- Standard document styles enable you to write professionally looking documents, because the corresponding style files have been written with a lot of care by people who know a lot about typography and aesthetics.

To a document, it is possible to associate one or several document styles, which are either standard or user defined. The main document style of a document is selected in the Document → Style menu. Extra styles can be added using Document → Use package.

From the editor point of view, each style corresponds to a `.ts` file. The files corresponding to each style are processed in as if they were usual documents, but at the end, the editor only keeps the final environment as the initial environment for the main document. More precisely, the style files are processed in order as well as there own styles, in a recursive manner.

9.2. The standard TEXMACS styles and packages

Currently, the following standard document styles have been implemented:

- Book;
- Article;
- Letter;
- Seminar (for transparencies).

Each of these styles export a certain number of standard functions and environments listed below. All future standard document styles are expected to support at least the above commands and environments and we suggest users to write style files which do so too.

- Sectioning commands.

- Itemize and enumerate environments.
- Equation like environments.
- Theorem like environments.
- Programming environments.

We notice that the theorem like environments are not standard in LaTeX, which is a standard source of non compatibility. New "theorems" can be added with the `newtheorem` command. It is also possible to add new "remarks" with the `newremark` command; "remarks" are different from "theorems" in the sense that their body is usually not typeset in an emphasized font.

Of course, programming environments are not supported by LaTeX either. Such environments are currently under development.

9.3. Designing your own style files

Whenever the standard TeXmacs style files are inadequate for a given purpose, it is possible to write your own style files. However, designing your own style file from scratch may be a complex task. For this reason, we recommend the reuse or customization of the standard TeXmacs style files and packages whenever possible. Consequently, it may be wise to read more about the customization of the TeXmacs style files and packages first.

9.3.1. Look at an example

Before writing your own style file, it may be useful to take a look at some standard style files. For instance, you may load `book.ts` using File → Load (no path is necessary here, since the style directory is included in the default file path).

After loading `book.ts`, you will see many function and environment declarations (these declarations are visible, since style files are written in "preamble mode" (see Options → Mode)). Some more declarations are contained in the files `basic.ts`, `list.ts`, `theorem.ts` and `program.ts` on which `book.ts` is based. These files respectively contain basic, itemize-like, theorem-like and programming environments.

9.3.2. Keyboard shortcuts for writing style files

9.3.2.1. Macros, functions and environment variables

The main key-combinations that you should know to write style files are the following:

`M-=` creates a new assignment. The first argument is a new command name and the second argument an expression.

`M-w` permits to locally change one or more environment variables. With statements are of the form $\langle x_1|a_1|\cdots|x_n|a_n|b\rangle$, where the x_i are the names of the variables, the a_i their local values, and b the text on which the local environment applies.

`M-m` creates a macro. Arguments to the macro can be inserted using the `tab`-key.

`M-f` creates a function. Arguments to the macro can be inserted using the `tab`-key.

`M-i #` get the value of a macro argument.

`M-i v` get the value of an environment variable.

`M-i e` expands the macro with zero or more arguments.

`M-i a` applies a function to zero or more arguments.

More precisely, when evaluating a macro expansion $\{a|x_1|\cdots|x_n\}$ created by `M-i e`, the following action is undertaken:

- If a is not a string nor a macro, then a is evaluated once. This results either in a macro name or a macro expression f.
- If we obtain a macro name, then we replace f by the value of the environment variable f. If, after this, f is still not a macro expression, then we return f.
- Let y_1, ..., y_n be the arguments of f and b it's body (superfluous arguments are discarded; missing arguments take the empty string as their default value). Then we substitute x_i for each y_i in b and return the evaluated result.

Functions are similar to macros, except that the arguments of function appliciations are evaluated and they can not be edited in a direct way (you first need to deactivate the function application, edit the arguments, and reactivate). Also, y_1, ..., y_n are now rather considered as local environment variables, which are given $x_1, ..., x_n$ as their values. These local variables are not remembered when a function returns a function which involves these variables.

9.3.2.2. Computational markup

The following commands can be used for performing dynamic computations:

`M-e |` sequential or of two conditions.

`M-e ^` exclusive or of two conditions.

`M-e &` sequential and of two conditions.

`M-e !` negation of a condition.

`M-e +` add two numbers or lengths.

`M-e -` subtract two numbers or lengths.

`M-e *` multiply two numbers.

`M-e /` divide two numbers.

`M-e ;` concatenate two strings.

`M-e #` display a number in Arabic, roman, Roman, alpha or Alpha (used for instance in enumerations).

`M-e >` translate a word from a source language into a destination language (see the dictionaries in `$TEXMACS_PATH/data/dic`).

`M-e =` test equality.

`M-e E-e` test inequality.

`M-e ?` insert an if statement with an optional else part.

9.3.3. Important TeXMACS paths

This should be elsewhere.

Before writing your own style file, it is useful to know the following important TeXMACS paths:

- `$TEXMACS_PATH` is the main path for TeXMACS.
- `$TEXMACS_HOME_PATH` is the main user path for TeXMACS files (documents, styles or programs). By default, this path is set to `~/.TeXmacs`.
- `$TEXMACS_STYLE_ROOT` the root directories for style files. By default, this path contains `$TEXMACS_PATH/styles` and `$TEXMACS_HOME_PATH/styles`.
- `$TEXMACS_PACKAGE_ROOT` the root directories for style packages. By default, this path contains `$TEXMACS_PATH/packages` and `$TEXMACS_HOME_PATH/packages`.
- `$TEXMACS_STYLE_PATH` contains the path for including style files. By default, this path contains `.` and all subdirectories in `$TEXMACS_STYLE_ROOT` and `$TEXMACS_PACKAGE_ROOT`.
- `$TEXMACS_FILE_PATH` contains the path for searching text files. By default, this path contains `$TEXMACS_STYLE_PATH`, `$TEXMACS_PATH/texts` and `$TEXMACS_HOME_PATH/texts`.

9.4. Customizing the standard TeXMACS styles and packages

Whenever the standard TeXMACS style files are inadequate for a given purpose, it is possible to write your own style files. Designing your own style file from scratch may be a complex task. For this reason, the TeXMACS style files have been subdivided in smaller packages in order to facilitate the reuse of certain parts. The design policy also allows you to redefine many macros *a posteriori*, which allows you to customize the existing style files in an easy way.

Chapter 10

Summary of the principal TeXmacs tags

10.1. The common base for most styles

The common-base d.t.d. contains the markup which is common to virtually all styles. It is subdivided into the following parts:

10.1.1. Standard markup

Various standard markup is defined in std-markup. The following textual content tags all take one argument. Most can be found in the Text → Content tag menu.

strong Indicates an **important** region of text. You can enter this tag via Text → Content tag → Strong.

em Emphasizes a region of text like in "the *real* thing". This tag corresponds to the menu entry Text → Content tag → Emphasize.

dfn For definitions like "a *gnu* is a horny beast". This tag corresponds to Text → Content tag → Definition.

samp A sequence of literal characters like the ae ligature æ. You can get this tag via Text → Content tag → Sample.

name The name of a particular thing or concept like the Linux system. This tag is obtained using Text → Content tag → Name.

person The name of a person like Joris. This tag corresponds to Text → Content tag → Person.

cite* A bibliographic citation like a book or magazine. Example: Melville's *Moby Dick*. This tag, which is obtained using Text → Content tag → Cite, should not be confused with cite. The latter tag is also used for citations, but where the argument refers to an entry in a database with bibliographic references.

abbr An abbreviation. Example: I work at the C.N.R.S. An abbreviation is created using Text → Content tag → Abbreviation or the A-a keyboard shortcut.

acronym An acronym is an abbreviation formed from the first letter of each word in a name or a phrase, such as HTML or IBM. In particular, the letters are not separated by dots. You may enter an acronym using Text → Content tag → Acronym.

`verbatim` Verbatim text like output from a computer program. Example: the program said `hello`. You may enter verbatim text via Text → Content tag → Verbatim. The tag may also be used as an environment for multi-paragraph text.

`kbd` Text which should be entered on a keyboard. Example: please type `return`. This tag corresponds to the menu entry Text → Content tag → Keyboard.

`code*` Code of a computer program like in "`cout << 1+1;` yields `2`". This is entered using Text → Content tag → Code. For longer pieces of code, you should use the `code` environment.

`var` Variables in a computer program like in `cp` *`src-file dest-file`*. This tag corresponds to the menu entry Text → Content tag → Variable.

`math` This is a tag which will be used in the future for mathematics inside regular text. Example: the formula $\sin^2 x + \cos^2 x = 1$ is well-known.

`op` This is a tag which can be used inside mathematics for specifying that an operator should be considered on itself, without any arguments. Example: the operation $+$ is a function from $\mathbb{R}^2$ to $\mathbb{R}$. This tag may become depreciated.

`tt` This is a physical tag for typewriter phase. It is used for compatability with HTML, but we do not recommend its use.

The following are standard environments:

`verbatim` Described above.

`code` Similar to `code*`, but for pieces of code of several lines.

`quote` Environment for short (one paragraph) quotations.

`quotation` Environment for long (multi-paragraph) quotations.

`verse` Environment for poetry.

`center` This is a physical tag for centering one or several lines of text. It is used for compatability with HTML, but we do not recommend its use.

Some standard tabular environments are

`tabular*` Centered tables.

`block` Left aligned tables with a border of standard `1ln` width.

`block*` Centered tables with a border of standard `1ln` width.

The following miscellaneous tags don't take arguments:

`TeXmacs` The TEXMACS logo.

`TeX` The TEX logo.

`LaTeX` The LATEX logo.

`hflush` Used by developers for flushing to the right in the definition of environments.

`hrule` A horizontal rule like the one you see below:

The following miscellaneous tags all take one or more arguments:

`overline` For overlined text, which can be wrapped across several lines.

`underline` For underlined text, which can be wrapped across several lines.

`fold` Macro with two arguments. The first argument is displayed and the second one ignored: the macro corresponds to the folded presentation of a piece of content associated to a short title or abstract. The second argument can be made visible using Insert → Switch → Unfold.

`unfold` Macro with two arguments *x* and *y*, which yields the unfolded presentation of a piece of content *y* associated to a short title or abstract *x*. The second argument can be made invisible using Insert → Switch → Fold.

`switch` Macro with two arguments *x* and *y*, where *y* is a set of possible representations of the switch and *x* the current representation. The function keys F9, F10, F11 and F12 can be used to switch between different representations.

`phantom` Function with one argument *x*. This tag takes as much space as the typesetted argument *x* would take, but *x* is not displayed. For instance, the text "phantom" as an argument of `phantom` yields " ".

`set-header` Function with one argument for permanently changing the header. Notice that certain tags in the style file, like sectional tags, may override such manual changes.

`set-footer` Function with one argument for permanently changing the footer.

10.1.2. Standard symbols

The `std-symbol` d.t.d. defines the special symbols ¢, ¤, ¥, ©, ⓒ, ®, °, ², ³, ¹, µ, ¶, ¼, ½, ¾, € and ™. As soon as the font support will be further improved, this d.t.d. will become obsolete.

10.1.3. Standard mathematical markup

Standard mathematical markup is defined in `std-math`.

`binom` For binomial coefficients $\binom{n}{m}$.

`choose` Alternative name for `binom` (depreciated)

`shrink-inline` A macro which switches to scriptsize text when you are not in display style. This macro is mainly used by developers. For instance, the `binom` macro uses on it.

The following are standard mathematical tabular environments:

`matrix` For matrices $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

`det` For determinants $\Delta = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$.

`choice` For choice lists $|x| = \begin{cases} -x, & \text{if } x \leqslant 0 \\ x, & \text{if } x \geqslant 0 \end{cases}$

10.1.4. Standard lists

The standard TEXMACS lists are defined in `std-list`. The unnumbered lists are:

`itemize` The tag before each item depends on the nesting depth.

`itemize-minus` Uses $-$ for the tag.

`itemize-dot` Uses $\bullet$ for the tag.

`itemize-arrow` Uses $\rightarrow$ for the tag.

Numbered lists correspond to the following environments:

`enumerate` The kind of number before each item depends on the nesting depth.

`enumerate-numeric` Number the items by 1, 2, 3, etc.

`enumerate-roman` Number the items by i, ii, iii, etc.

`enumerate-Roman` Number the items by I, II, III, etc.

`enumerate-alpha` Number the items by a), b), c), etc.

`enumerate-Alpha` Number the items by A, B, C, etc.

The following environments can be used for descriptive lists.

`description` The environment for default descriptive lists (usually `description-compact`).

`description-compact` Align the left hand sides of the items in the list and put their descriptions shortly behind it.

`description-dash` Similar to `description-compact`, but use a — to seperate each item from its description.

`description-align` Align the left hand sides of the descriptions, while aligning the items to the right.

`description-long` Put the items and their descriptions on distinct lines.

New items in a list are indicated through the `item` tag or the unary `item*` tag in the case of descriptions. Developers will also find a few additional, but unstable, macros in `std-list` for defining additional list structures.

10.1.5. Automatic content generation

The std-automatic d.t.d. specifies for the automatic generation of auxiliary content like tables of contents and bibliographies, as well as for the presentation of such auxiliary content. The following tags are used for bibliographies:

cite A function with an arbitrary number of arguments. Each argument is a citation corresponding to an item in a BiB-TEX file. The citations are displayed in the same way as they are referenced in the bibliography and they also provide hyperlinks to the correspoding references. The citations are displayed as question marks if you did not generate the bibliography.

nocite* Similar as cite, but the citations are not displayed in the main text.

bibitem* A function which specifies how to display an item in the bibliography.

The following tags are used for compiling tables of contents:

toc-main-1 A function with one argument for creating primordial entry in the table of contents. This function can for instance be used when a book consists of several parts.

toc-main-2 A function with one argument for creating a main entry in the table of contents. This function is usually used for chapters.

toc-normal-1 A function with one argument for creating a normal entry in the table of contents. This function is often used for sections.

toc-normal-2 Similar as toc-normal-2 for less important entries like subsections.

toc-normal-3 Similar as toc-normal-3 for even less important entries like subsubsections.

toc-small-1 Used for not very important entries such as paragraphs (may be ignored).

toc-small-2 Used for even less important entries such as subparagraphs.

toc-dots The separation between an entry in the table of contents and the corresponding page number. By default, we use horizontal dots.

The following tags are used for indices:

index A function with one argument *x*, which inserts *x* in the index as a principal entry.

subindex A function with two arguments *x* and *y*, which inserts *y* in the index as a subentry of *x*.

subsubindex A function with three arguments *x*, *y* and *z*, which inserts *z* in the index as a subentry of *y*, which is itself a subentry of *x*.

index-complex A function with four arguments *key*, *how*, *range*, *entry*, which is documented in the section about index generation.

index-line This function takes a *key* argument, which tells how to sort the entry, and the actual *entry*. No page number is generated.

`index-1` Macro with an index entry and a page number, which is used for rendering a principal index entry in the index.

`index-1*` Similar to `index-1`, but without the page number.

`index-`n (with n between 1 and 5): macro with an index entry and a page number, which is used for rendering an index entry of level n.

`index-`n`*` Similar to `index-`n, but without the page number.

`index-dots` The macro which produces the dots between an index entry and the corresponding page number(s).

The following tags are used for glossaries:

`glossary` A function which inserts its only argument into the glossary.

`glossary-dup` For creating an additional page number for an entry which was already inserted before.

`glossary-explain` A function for inserting a glossary entry with its explanation.

`glossary-line` Insert a glossary entry without a page number.

`glossary-1` Macro for rendering a glossary entry and its corresponding page number.

`glossary-2` Macro for rendering a glossary entry, its explanation, and its page number.

`glossary-dots` The macro which produces the dots between a glossary entry and the corresponding page number(s).

10.1.6. Special markup for programs and sessions

The `program` d.t.d. mainly provides the following environments for computer algebra sessions:

`session` Macro with three arguments: the computer algebra language, the name of the session and the body of the session itself.

`input` Macro with two arguments: a prompt and a the input itself.

`output` Macro with the body of the output as its argument.

In fact, these environments are based on environments of the form *`lan`*`-session`, *`lan`*`-input` and *`lan`*`-output` for every individual language *`lan`*.

The `program` d.t.d. also provides some markup for the layout of computer programs. However, these tags should be considered as very unstable, since we plan to replace them by a set of more detailed tags:

`algorithm` Macro with two arguments: the name of the algorithm and algorithm itself, together with its possible specification.

`body` The real body of the algorithm.

`indent` For indenting part of an algorithm.

10.2. STANDARD ENVIRONMENTS INSIDE TEXT

The `env` d.t.d. contains the standard environments which are available in most styles. It is subdivided into the following parts:

10.2.1. Defining new environments

The `env-manage` contains high-level markup which can be used by the user to define new environments for theorems, remarks, exercises and figures:

`newtheorem` Defines a theorem-like environment. You should specify a name for the environment (like "experiment") and the corresponding text (like "Experiment").

`newremark` Similar as `newtheorem`, but for remarks.

`newexercise` Similar as `newtheorem`, but for exercises.

`newfigure` Similar as `newtheorem`, but for figures (in big and small pairs).

The d.t.d. also contains low-level markup for the actual definitions of the environments. In fact, the definition of new theorems is done in two stages. At the first stage, the `newtheorem` tag is used in order to specify which theorem-like environments should be defined. At the second stage (just before the user's document is processed) the theorem-like environments are actually defined. This mechanism makes it possible to customize the environments in packages which are processed between the two stages. For instance, the numbering of theorems is customized in this way.

WARNING 10.1. At the moment, you should only use the `newtheorem` and similar tags inside a personal style file or package. If you use `newtheorem` directly inside a document, then the numbering can be incorrect, due to the two-stage scheme explained above. This inconvenience will disappear as soon as it will be possible to specify clean preambles for TEXMACS documents.

10.2.2. Mathematical environments

The `env-math` d.t.d. specifies which mathematical environments can be used inside text-mode. In other words, the environments should be used inside text-mode, but their bodies contain mathematical formulas or tables of mathematical formulas.

`equation` A numbered equation.

`equation*` An unnumbered equation.

`eqnarray` An array of numbered equations (should not be used yet).

`eqnarray*` An array of unnumbered equations.

Inside the `eqnarray*` environment, you can use the `eqnumber` tag in order to number the equation.

Warning 10.2. The numbering of equations inside tables is not yet as it should be. In particular, the `eqnarray` tag is equivalent to `eqnarray*` at the moment. Later on, when the `eqnarray` tag will be implemented correctly, you will also have a `nonumber` tag in order to suppress the number of an equation, and a style package for numbering equations at the left hand side.

Warning 10.3. There is no option for numbering equations at the left hand side available yet. Nevertheless, you may use the manual tag `leqnumber` for this. You also have a tag `nextnumber` which directly display the next number and increases the equation counter.

Warning 10.4. We do not encourage the use of the AMS-TEX environments `align`, `gather` and `split`. Nevertheless, they are available under the names `align`, `gather`, `eqsplit` together with their variants `align*`, `gather*` and `eqsplit*`. In the future, we plan to provide more powerful environments.

10.2.3. Theorem-like environments

The `env-theorem` d.t.d. provides tags for the layout of theorem-like environments. The most important tags are

`theorem*` A macro for displaying a theorem-like environments. The first argument specifies the name of the theorem, like "Theorem 1.2" and the second argument contains the body of the theorem. This environment is used for environments defined by `newtheorem`.

`remark*` Similar to `theorem*`, but for remark-like environments.

`exercise*` Similar to `theorem*`, but for exercise-like environments.

`proof*` Similar to `theorem*`, but for proofs. This environment is mainly used for customizing the name of a proof, like in "End of the proof of theorem 1.2".

`dueto` An environment which can be used to specify the inventors of a theorem.

`corollary*` For unnumbered corollaries. This environment is based on `theorem*`.

`proof` For proofs of theorems. This environment is based on `proof*`.

The following tags can be used for further customization of the environments.

`theoremname` A macro which controls the appearance of the names of theorem-like *and* remark-like environments. Most styles use bold face or small capitals.

`exercisename` Similar to `theoremname`, but for exercises.

`theoremsep` The separator between the name of a theorem-like or remark-like environment and its main body. By default, this is a period followed by a space.

`exercisesep` Similar to `theoremsep`, but for exercises.

10.2.4. Environments for floating objects

The `env-float` d.t.d. provides tags for floating objects. The following tag is the only high-level one:

`footnote` Make a footnote.

The following low-level tags can be used for the definitions of high-level figure and table environments like `big-figure`, `small-figure`, `big-table` and `small-table`:

`small-figure*` A macro for displaying a small figure. The arguments are a short name (like "figure" or "table") for the list of figures, its real name (like "Figure 2.3" or "Table 5"), the figure itself and a caption.

`big-figure*` A variant of `small-figure*` for displaying a big figure.

The following tags can be used for customizing the appearance the text around figures, tables and footnotes:

`figurename` A macro which controls the appearance of the text "Figure". By default, we use bold face.

`figuresep` The separator between the figure and its number and the caption. By default, this is a period followed by a space.

`footnotesep` The separator between the number of the footnote and the text. By default, this is a period followed by a space.

10.2.5. Default environments

The `env-default` d.t.d. contains the default textual environments. They are subdivided into the following groups:

Variants of theorems. The bodies of theorem-like environments are usually emphasized. By default, the following such environments are available via Text → Environment: `theorem`, `proposition`, `lemma`, `corollary`, `axiom`, `definition`, `notation`, `conjecture`.

Variants of remarks. The following ones are available via Text → Environment: `remark`, `example`, `note`, `warning`, `convention`.

Variants of exercises. Two such environments are provided by default and available via Text → Environment: `exercise` and `problem`.

Variants of figures. These environments always come by pairs: big and small ones. By default, we provide `big-figure`, `small-figure`, `big-table` and `small-table`. You can access them through Insert → Image and Insert → Table.

Other useful environments. We provide `keywords` and `AMS-class` (for the A.M.S. subject classification). These environments should be entered at the interior of an `abstract`.

10.3. Headers and footers

10.3.1. Standard headers

The `header` d.t.d. provides tags for customizing the headers and footers. The customization is based on the idea that we may specify a *page text* for every page. This page text can for instance be a running title or the name of the current section. The page text may depend on the parity of a page and appear in a different way for special pages like starts of new chapters. The following tags control the physical layout of different types of pages:

`start-page` This tag, with the page text as its only argument, specifies the layout of the first page of a new chapter or section.

`odd-page-text` Similar to `start-page`, but for the layout of ordinary odd pages.

`even-page-text` Similar to `start-page`, but for the layout of ordinary even pages.

The following tags control the logical header-related actions to be undertaken, when specifying a title, an author, or when starting a new section.

`header-title` A tag with a "title argument" which is used at the specification of the document title.

`header-author` A tag with an "author argument" which is used at the specification of the document author.

`header-primary` A tag with a "section name argument" which is used at the start of each new primary section (i.e. `chapter` for book style, or `section` for article style).

`header-secondary` A tag with a "section name argument" which is used at the start of each new secondary section (i.e. `section` for book style, or `subsection` for article style).

10.3.2. Standard titles

The `header-title` d.t.d. provides tags for title information. The following high-level tags can only be used when encapsulated inside a `make-title` tag:

`title` Specify a title for the document.

`author` Specify one or several authors for the document.

`address` Specify the address of the author.

`address-block` Specify an address of an author (in case of multiple addresses).

`title-email` Specify the email address of the author.

`title-date` Specify the creation date of the article.

The `title` and `author` use the `header-title` and `header-author` tags for specifying the running title and header. You may override these by reusing `header-title` resp. `header-author`. The above tags also depend on the following low-level tags for their physical layout:

`title*` Macro with one argument which specifies the physical layout of titles.

`author*` Macro with one argument which specifies the physical layout of authors.

`address*` Macro with one argument which specifies the physical layout of addresses.

`title-email*` Macro with one argument which specifies the physical layout of email addresses.

`title-date*` Macro with one argument which specifies the physical layout of creation dates.

The `header-title` d.t.d. also defines the `abstract` tag for abstracts of documents.

10.4. LATEX STYLE SECTIONS

The `section-latex` d.t.d. provides the standard tags for sections, which are the same as in LaTeX. Most sectional tags take only one argument: the name of the section. In the future, we plan to provide alternative tags with two arguments, which will allow you to see the body of a section as part of the structure. The following tags usually yield numbered sections, which are referenced in the table of contents:

`chapter` Macro for producing a potentially numbered chapter title.

`section` Macro for producing a potentially numbered section title.

`subsection` Macro for producing a potentially numbered subsection title.

`subsubsection` Macro for producing a potentially numbered subsubsection title.

`paragraph` Macro for producing a potentially numbered paragraph title.

`subparagraph` Macro for producing a potentially numbered subparagraph title.

The tags `chapter*`, `section*`, `subsection*`, `subsubsection*`, `paragraph*` and `subparagraph*` can be used for producing the unnumbered variants of the above tags, which are not referenced in the table of contents. The `section-latex` d.t.d. also provides the following tags:

`chapter**` Macro with two arguments: a special type of chapter (like "Epilogue") and the name of the chapter.

`appendix` A variant of `chapter` or `section` for producing appendices.

`sectionsep` A macro for customizing the separator between the number of a section and its title. By default, we use two spaces.

10.5. Markup for automatic numbering

10.5.1. Numbering environments

The `number-env` d.t.d. provides low-level tags for the numbering of standard environments. One of the most important tags is `init-stdenv` which is used for resetting all environment counters. This is usually done at the start of each chapter or section, or once for the entire document.

The d.t.d. also exports the very low-level tags `newliststdenv`, `newlistfigure` and `newliststdenv-counter`, which control the numbering in collaboration with `env-manage`. The packages `number-us` and `number-europe` are provided for American-style and European-style numbering.

10.5.2. Numbering sections

The `number-section` d.t.d. provides low-level tags for numbering sections. It defines tags `reset-chapter`, `reset-section`, `reset-subsection` and `reset-subsubsection`, for resetting the appropriate counters at each new chapter, section, subsection or subsubsection. It also defines a tag `reset-top` for resetting all top-level counters.

The d.t.d. also defines the tags `thechapter`, `thesection`, `thesubsection` and `thesubsubsection` for printing the name of the current chapter, section, subsection or subsubsection. It finally provides the tag `theprefix` which yields the prefix when numbering environments (equations, theorems, tables, etc.).

Chapter 11

Customizing TeXmacs

One major feature of TeXmacs is that it can be highly customized. First of all, the most important aspects of the program can be configured in Edit → Preferences. Most other parts of TeXmacs can be entirely adapted or reprogrammed using the Guile/Scheme extension language. In the sequel, we give a short overview of how this works in simple cases.

11.1. Introduction to the Guile extension language

Like Emacs, TeXmacs comes with a Lisp-like extension language, namely the Guile Scheme dialect from the Gnome project. For documentation about Guile Scheme, we refer to

```
http://www.gnu.org/software/guile/guile.html
```

Scheme has the advantage that it may be extended with extern C and C++ types and routines. In our case, we have extended Scheme with routines which you can use to create your own menus and key-combinations, and even to write your own extensions to TeXmacs.

If you have downloaded the source files of TeXmacs, then it may be interesting for you to take a look at the files

```
Guile/Glue/build-glue-basic.scm
Guile/Glue/build-glue-editor.scm
Guile/Glue/build-glue-server.scm
```

These three "glue" files contain the C++ routines, which are visible within Scheme. In what follows, we will discuss some of the most important routines. We plan to write a more complete reference guide later. You may also take a look at the scheme `.scm` files in the directory `$TEXMACS_PATH/progs`.

11.2. Writing your own initialization files

When starting up, TeXmacs executes the file `$TEXMACS_PATH/progs/init-texmacs.scm`, unless you have specified your own initialization file in `$TEXMACS_HOME_PATH/progs/my-init-texmacs.scm`. By default, the path `$TEXMACS_HOME_PATH` equals `.TeXmacs`. Usually, you want to add some extra actions to the default initialization file. In this case, you should not forget to include the command

```
(exec-file "$TEXMACS_PATH/progs" "init-texmacs.scm")
```

in your personal initialization file. Similarly, the file `$TEXMACS_PATH/progs/init-buffer.scm` is executed, each time you create a new buffer, unless you provide your own initialization file `$TEXMACS_HOME_PATH/progs/my-init-buffer.scm`.

11.3. Creating your own dynamic menus

In particular, the default initialization file executes

```
$TEXMACS_PATH/progs/menu/main_menu.scm
```

in order to set up the TEXMACS main menu. We suggest you to have a look at this file in order to see how menus are created.

Actually, any menu or part of a menu is represented by a program. The program consists of a list of programs of one of the following forms:

```
(=> "pulldown menu name" menu-definition)
(-> "pullright menu name" menu-definition)
("entry" action)
("entry" "shorthand" action)
---
(if condition menu-definition)
(link variable)
```

The constructors `=>` and `->` are used to create pulldown or pullright menus and `menu-definition` should contain a program which creates the submenu. The constructor `("entry" action)` creates an ordinary entry, where `action` will be compiled and executed when you click on `entry`. The optional `"shorthand"` stands for a keyboard macro with the same action. Items of a menu may be separated using `---`. The constructor `if` is used for inserting menu items only if a certain `condition` is satisfied (for instance, if we are in math mode).

Finally, if we declared a TEXMACS variable to be a menu by

```
(define variable menu-definition)
```

then we may use this menu indirectly using the `link` constructor. This indirect way of declaring submenus has two advantages

- An "indirect" submenu may be linked to as many menus as we like.
- New items may be added to "indirect" submenus *a posteriori* using

  ```
  (set! variable (menu-merge variable menu-declaration))
  ```

  Actually, existing terms may also be overwritten in this way.

Some standard TEXMACS indirect menus are `texmacs-menu`, `file-menu`, `edit-menu`, `insert-menu`, `text-menu`, `paragraph-menu`, `document-menu`, `options-menu` and `help-menu`. The command

```
(menu-main menu-declaration)
```

is used in order to actually set the main menu. For instance, at initialization, we execute

```
(menu-main '(link texmacs-menu))
```

11.4. Creating your own keyboard shortcuts

Keymaps are specified using the command

```
(set-keymap (list of predicates) (list of keymaps))
```

The list of predicates specifies under which circumstances the keymaps are valid. Examples of predicates are `always?`, `in-math?` and `in-french?`, but the user may define his own predicates. Each map is of one of the following forms

```
(key-combination action_1 ... action_n)
(key-combination result)
(key-combination result help-message)
```

In the first case, the `action_i` are SCHEME commands associated to the string `key-combination`. In the second and third case, `result` is a string which is to be inserted in the text when the `key-combination` has been completed. An optional `help-message` may be displayed when the `key-combination` is finished.

Kemaps can be cleared again using the command

```
(remove-keymap (list of predicates) (list of key combinations))
```

where the second arguments is a list of strings, each of which is a `key-combination` as above.

11.5. OTHER INTERESTING FILES

Some other files may also be worth looking at:

- `$TEXMACS_PATH/fonts/enc` contains encodings for different TEX fonts.
- `$TEXMACS_PATH/fonts/virtual` contains definitions of virtual characters.
- `$TEXMACS_PATH/langs/natural/dic` contains the current dictionaries used by TEXMACS.
- `$TEXMACS_PATH/langs/natural/hyphen` contains hyphenation patterns for various languages.
- `$TEXMACS_PATH/progs/fonts` contains SCHEME programs for setting up the fonts.

Chapter 12

Compatibility with other formats

TEXMACS is fully compatible with postscript, which is used as the format in order to print documents. TEXMACS also provides converters from and to LATEX and an input filter for Html.

12.1. Compatibility with LATEX

Although TEXMACS has not been designed to be fully compatible with LATEX, it is possible to convert documents from TEXMACS to LATEX and *vice versa*, although the result will not always be perfect. Also, conversions from TEXMACS to LATEX will generally yield better results than conversions the other way around. In particular, TEXMACS may reasonably well be used to write articles, which need to be converted to LATEX for submission purposes. In this chapter, we will describe more precisely the conversion mechanisms, which will help you to obtain a result as satisfactory as possible.

12.1.1. Conversion from TEXMACS to LATEX

The most common situation is that you want to convert an article from TEXMACS to LATEX, in order to submit it to some journal. Given a TEXMACS file `name.tm`, you may convert it into a LATEX file `name.tex` using File → Export → Latex. At a first stage, you may try to run LATEX on `name.tex`, and see whether you obtain a satisfactory result. If so, then you should submit `name.tex` together with the style file `TeXmacs.sty`, which can be found in the directory `$TEXMACS_PATH/misc/latex`.

Often, the journal to which you submit uses its own style file, say `journal.sty`. In that case, you should also copy the file

```
$TEXMACS_PATH/styles/article.ts
```

to

```
~/.TeXmacs/styles/journal.ts
```

and use `journal` as your document style in Document → Style → Other. You may optionally edit `journal.ts`, so that the article layout becomes closer to the journal's style. In some cases, you also have to create a new copy of `TeXmacs.sty`, and modify some of the environments for compatibility with the journal's style file `journal.sty`.

If your first try to convert your document into LATEX did not yield a satisfactory result, then you will usually observe that only minor parts of the texts were not converted correctly. This may be due to three main reasons:

- Your text uses some specific TEXMACS features.

- You used a TEXMACS feature, which has not yet been implemented in the conversion algorithm.
- You found a bug in the conversion algorithm.

These issues will be discussed in more detail in the next section.

In case of problems, a naive strategy would be to correct the produced LaTeX file and to send it to the journal. However, this strategy has the disadvantage that you have to make these corrections over and over again, each time that you convert your TEXMACS file `name.tm`, after having made some extra modifications. A better strategy is to use the Insert → Specific → Latex and Insert → Specific → Texmacs constructs to write text which is visible in the converted resp. original file only.

For instance, assume that the word "blauwbilgorgel" is hyphenated correctly in the TEXMACS source, but not in the LaTeX conversion. Then you may proceed as follows:

1. Select "blauwbilgorgel".
2. Click on Insert → Specific → Texmacs to make the text "blauwbilgorgel" TEXMACS-specific.
3. Click on Insert → Specific → Latex.
4. Type the latex code `blauw\-bil\-gor\-gel` with the correct hyphenation.
5. Press `return` to activate the LaTeX-specific text.

In a similar fashion, you may insert LaTeX-specific line breaks, page breaks, vertical space, style parameter modifications, etc.

12.1.2. Possible conversion problems

12.1.2.1. Specific TEXMACS features

Some TEXMACS typesetting primitives have no analogues in LaTeX, and the conversion algorithm will simply transform them into blank space. Some main features which are specific to TEXMACS are the following:

- Left primes.
- Big separators between big parentheses.
- Mosaics.
- Trees.
- Complex user macros.
- Vertical spaces "before" and "after".
- Indentation flags "before" and "after".

You should avoid to use these specific TeXmacs features, if your document needs to be converted into LaTeX. Nevertheless, in the far future, the conversion program might generate encapsulated postscript by default of a more intelligible translation.

12.1.2.2. Not yet implemented conversions

Although we try to keep the conversion algorithm as complete as possible for your needs, certain things have not yet been implemented. Some examples of not yet implemented issues are

- Non standard fonts.
- Conversion of tabulars.
- Style parameters.

Any suggestion about desirable extensions of the conversion algorithm should be reported to

`contact@texmacs.org`

and we will try to incorporate it as quickly as possible. It may take some time to implement the correct conversion of style parameters, since these are not the same in TeXmacs and LaTeX. Furthermore, layout differences between TeXmacs and LaTeX can not entirely be eliminated.

12.1.2.3. Bugs in the conversion algorithm

The most annoying situation if when a converted TeXmacs document produces lots of errors at the compilation or if the result has nothing to do with the original. In that case you have probably detected a bug in the conversion algorithm (or in the installation of LaTeX on your system). Please try to figure out the source of the bug in this case and report it by sending an email to

`TeXmacs@math.u-psud.fr`

12.1.2.4. Work-arounds

TeXmacs has not been designed to be fully compatible with LaTeX. As to the conversion from LaTeX to TeXmacs, our main aim is to *help* you in converting old documents to TeXmacs. As long as you did not define weird environments and as long as you did not use weird style files and commands, you should be able to convert your old documents reasonably well. Otherwise, we suggest to modify your old document in a way that is does convert reasonably well and to apply some final changes in the result.

12.1.3. Conversion from LaTeX to TeXmacs

The current aim of the conversion program from LaTeX to TeXmacs, is to *help* you in translating old documents into TeXmacs. *Grosso modo*, conversions from LaTeX to TeXmacs are more problematic than conversions the other way around. Nevertheless, as long as you restrict yourself to using the most common LaTeX commands, you should be able to convert your old documents reasonably well. For example, all TeXmacs help files have been written in LaTeX in order to validate the LaTeX to TeXmacs conversion program.

You may convert a LaTeX document `name.tex` into TeXmacs using File → Import → Latex and save it under `name.tm`. If your LaTeX document was written sufficiently well, then the converted result should be more or less acceptable, apart from certain unrecognized commands, which appear in red. A good solution would be to write your own style file for converted documents, based on the original style, and in which the unrecognized commands are defined.

However, in certain less fortunate cases, the converted document will look like a great mess. This usually stems from the fact that TeX and LaTeX allow users to modify the parser dynamically, for instance using the `\catcode` command. In this case, the conversion program may get confused, by making erroneous assumptions on the mode or the environment. As a result, text may be converted as mathematics, mathematics as verbatim, and so on. Nevertheless, the commands in your source file `name.tex` which confused the conversion program are usually easily localized by comparing the LaTeX source with its TeXmacs conversion. Modulo some hacking of the source, you should be able to remove the litigious code, so that the document converts reasonably well.

In the future, we also plan to extend the conversion program with a style file converter and some additional features which facilitate the translation of user defined commands, which are defined in another document than the one you want to convert.

12.2. CONVERSION OF TEXMACS DOCUMENTS TO HTML

We have started to implemented the conversion between HTML and TeXmacs. At this moment, it is only possible to import HTML documents using File → Import → Html. Most of HTML 2.0 and parts of HTML 3.0 are currently supported. However, no browsing facilities have been added yet. In the future, we plan to implement Math-ML.

When importing HTML documents, files whose names start with `http:` or `ftp:` will be downloaded from the web using `wget`. If you compiled TeXmacs yourself, then you can download `wget` from

```
ftp://ftp.gnu.org/pub/gnu/wget/
```

In the binary distributions, we have included `wget`.

Appendix A

Configuration of TEXMACS

A.1. Introduction

Before you start using TEXMACS, it may be wise to configure the program first in Edit → Preferences, so that it will fit your needs best. Most importantly, you should choose a "look and feel" in Edit → Preferences → Look and feel. This will enable you for instance to let the keyboard shortcuts used by TEXMACS be similar to what you are used to in other applications.

Also, TEXMACS comes with a powerful keyboard shortcut system, which attempts to optimize the use of the modifier keys like `shift` and `control` on your keyboard. However, on certain systems these modifier keys are not well configured, so that you may wish to redo this yourself.

A.2. Configuration of the modifier keys

TEXMACS uses five major keyboard modifiers: `shift`, `control`, `alternate`, `meta` and `hyper`, which are abbreviated as `S-`, `C-`, `A-`, `M-` and `H-`. The `shift` and `control` keys are present on virtually all keyboards and the `alternate` key on almost all. Most keyboards for PC's nowadays also have a `windows` key, which is usually equivalent to `meta` for TEXMACS.

Before reconfiguring your keyboard, you should first check that this is indeed necessary. If you have keys which correspond to `shift`, `control`, `alternate` and `meta` in a suitable way, then you probably do not want to do anything. A possible exception is when you want to use a simple key like `caps-lock` for typing mathematical symbols. In that case, you should map `caps-lock` to `hyper`.

In order to reconfigure the keyboard, you simply select the logicial modifier that you want to correspond to a given physical key in Edit → Preferences → Keyboard. For instance, selecting Windows key → Map to M modifier, the `windows` key will correspond to the `meta` modifier. Similarly, when selecting Caps-lock key → Map to H modifier, the `caps-lock` key will correspond to the `hyper` modifier.

Unfortunately, X Window only allows system-wide reconfiguration. Consequently, if you reconfigure the `caps-lock` key inside TEXMACS, then the new behaviour of `caps-lock` will affect all other applications too. It is therefore important to reconfigure only those keys which you do not use for something else in other applications. For instance, the `windows` key is not used by many applications, so it generally does not do any harm to reconfigure it. You may also prefer to perform an appropriate system-wide configuration. This can be done using the `xmodmap` command; see the corresponding manual page for more information.

In certain cases, you already have keys on your keyboard which correspond to `alter`, `meta` and `hyper`, but not in the way you want. This can be done by remapping the `A-`, `M-` and `H-` prefixes to other logical modifiers in the first group of submenus of Edit → Preferences → Keyboard.

For instance, for Emacs compatability, you might want to permute the `meta` or `windows` key with `alter` without making any system-wide changes. This can be done by finding out which modifiers correspond to these keys; usually this will be `Mod1` for `alter` and `Mod4` for `meta` or `windows`. We next perform the necessary permutation in Edit → Preferences → Keyboard, by selecting A modifier → Equivalent for Mod4 and M modifier → Equivalent for Mod1.

A.3. Notes for Russian and Ukranian users

In order to type Russian (and similarly for Ukranian) text, you may several options:

- Select Russian as your default language in Edit → Preferences → Language → Russian. If TEXMACS starts with Russian menus, then this is done automatically if the Russian locale is set.
- Select Russian for an entire document using Document → Language → Russian.
- Select Russian for a portion of text in another document using Format → Language → Russian.

If your X server uses the xkb extension, and is instructed to switch between the Latin and Russian keyboard modes, you need not do anything special. Just switch your keyboard to the Russian mode, and go ahead. All the software needed for this is included in modern Linux distributions, and the xkb extension is enabled by default in `XF86Config`. With the xkb extension, keysyms are 2-byte, and Russian letters are at 0x6??. The keyboard is configured by `setxkbmap`. When X starts, it issues this command with the system-wide `Xkbmap` file (usually living in `/etc/X11/xinit`), if it exists; and then with the user's `~/.Xkbmap`, if it exists. A typical `~/.Xkbmap` may look like

```
ru basic grp:shift_toggle
```

This means that the keyboard mode is toggled by `l-shift r-shift`. Other popular choices are `ctrl shift` or `ctrl alt`, see `/usr/X11R6/lib/X11/xkb/` for more details. This is the preferred keyboard setup for modern Linux systems, if you plan to use Russian often.

In older Linux systems, the xkb extension is often disabled. Keysyms are 1-byte, and are configured by `xmodmap`. When X starts, it issues this command with the system-wide `Xmodmap` (usually living in `/etc/X11/xinit`), if it exists; and then with the user's `~/.Xmodmap`, if it exists. You can configure the mode toggling key combination, and use a 1-byte Russian encoding (such as koi8-r) in the Russian mode. It is easier to download the package `xruskb`, and just run

```
xrus jcuken-koi8
```

at the beginning of your X session. This sets the layout jcuken (see below) and the encoding koi8-r for your keyboard in the Russian mode. If you use such keyboard setup, you should select Options → international keyboard → russian → koi8-r.

It is also possible to use the Windows cp1251 encoding instead of koi8-r, though this is rarely done in UNIX. If you do use `xrus jcuken-cp1251`, select cp1251 instead of koi8-r.

All the methods described above require some special actions to "russify" the keyboard. This is not difficult, see the Cyrillic-HOWTO or, better, its updated version

`http://www.inp.nsk.su/~baldin/Cyrillic-HOWTO-russian/Cyrillic-HOWTO-russian.html`

Also, all of the above methods globally affect all X applications: text editors (emacs, nedit, kedit...), xterms, TEXMACS etc.

If you need to type Russian only once, or very rarely, a proper keyboard setup may be more trouble than it's worth. For the benefit of such occasional users, TEXMACS has methods of Russian input which require no preliminary work. Naturally, such methods affect only TEXMACS, and no other application.

The simplest way to type some Russian on the standard US-style keyboard with no software setup is to select Edit → Preferences → Keyboard → Cyrillic input method → translit. Then, typing a Latin letter will produce "the most similar" Russian one. In order to get some Russian letters, you have to type 2- or 3-letter combinations:

Shorthand	for	Shorthand(s)	for
`A-" e`	ё	`A-" E`	Ё
`y o`	ё	`Y o` `Y O`	Ё
`z h`	ж	`Z h` `Z H`	Ж
`j tab`	ж	`J tab`	Ж
`c h`	ч	`C h` `C H`	Ч
`s h`	ш	`S h` `S H`	Ш
`s c h`	щ	`S c h` `S C H`	Щ
`e tab`	э	`E tab`	Э
`y u`	ю	`Y u` `Y U`	Ю
`y a`	я	`Y a` `Y A`	Я

Table A.1. Typing Cyrillic text on a Roman keyboard.

If you want to get, e.g., "сх", and not "ш", you have to type `s / h`. Of course, the choice of "optimal" mapping of Latin letters to Russian ones in not unique. You can investigate the mapping supplied with TEXMACS and, if you don't like something, override it in your `~/.TeXmacs/progs/my-init-texmacs.scm`.

If you select jcuken instead of translit, you get the "official" Russian typewriter layout. It is so called because the keys "qwerty" produce "йцукен". This input method is most useful when you have a Russian-made keyboard, which has additional Russian letters written on the key caps in red, in the jcuken layout (a similar effect can be achieved by attaching transparent stickers with red Russian letters to caps of a US-style keyboard). It is also useful if you are an experienced Russian typist, and your fingers remember this layout.

Those who have no Russian letters indicated at the key caps often prefer the yawerty layout, where the keys "qwerty" produce "яверты". Each Latin letter is mapped into a "similar" Russian one; some additional Russian letters are produced by `shift`-digits. TEXMACS comes with a slightly modified yawerty layout, because it does not redefine the keys `$`, `£`, `\`, which are important for TEXMACS, are not redefined. The corresponding Russian letters are produced by some `shift`-digit combinations instead.

Appendix B

About GNU TEXMACS

B.1. Summary

GNU TEXMACS	
Supported systems	Most GNU/Linux systems
Copyright	© 1998–2002 by Joris van der Hoeven
License	GNU General Public License
Web sites	`http://www.texmacs.org` `http://www.gnu.org/software/texmacs`
Contact	`contact@texmacs.org`
Regular mail	Dr. Joris van der Hoeven Dépt. de Mathématiques (Bât. 425) Université Paris-Sud 91405 Orsay Cedex France

Table B.1. Summary of the principal information about GNU TEXMACS.

B.2. The philosophy behind TEXMACS

B.2.1. A short description of GNU TEXMACS

GNU TEXMACS is a free scientific text editor, which was both inspired by TEX and GNU Emacs. The editor allows you to write structured documents via a wysiwyg (what-you-see-is-what-you-get) and user friendly interface. New styles may be created by the user. The program implements high-quality typesetting algorithms and TEX fonts, which help you to produce professionally looking documents.

The high typesetting quality still goes through for automatically generated formulas, which makes TEXMACS suitable as an interface for computer algebra systems. TEXMACS also supports the Guile/Scheme extension language, so that you may customize the interface and write your own extensions to the editor.

TEXMACS currently runs on most GNU/Linux systems (a >200MHz processor and >32Mb of memory are recommended) and on sun computers. Converters exist for TEX/LATEX and they are under development for HTML/MathML/XML. In the future, TEXMACS is planned to evolve towards a complete scientific office suite, with spreadsheet capacities, a technical drawing editor and a presentation mode.

B.2.2. Why freedom is important for scientists

One major objective of TEXMACS is to promote the development of free software for and by scientists, by significantly reducing the cost of producing high quality user interfaces. If you plan to write an interface between TEXMACS and other software, then please contact us.

As a mathematician, I am deeply convinced that only free programs are acceptable from a scientific point of view. I see two main reasons for this:

- A result computed by a "mathematical" system, whose source code is not public, can not be accepted as part of a mathematical proof.
- Just as a mathematician should be able to build theorems on top of other theorems, it should be possible to freely modify and release algorithms of mathematical software.

However, it is strange, and a shame, that the main mathematical programs which are currently being used are proprietary. The main reason for this is that mathematicians often do not consider programming as a full scientific activity. Consequently, the development of useful software is delegated to "engineers" and the resulting programs are used as black boxes.

This subdivision of scientific activity is very artificial: it is often very important from a scientific point of view to know what there is in the black box. Inversely, deep scientific understanding usually leads to the production of better software. Consequently, I think that scientists should advocate the development of software as a full scientific activity, comparable to writing articles. Then it is clear too that such software should be diffused in a way which is compatible with the requirements of science: public availability, reproducibility and free usability.

B.3. The authors of TEXMACS

The GNU TEXMACS system, which is part of the GNU project, was designed and written by Joris van der Hoeven. The system was inspired both by the TEX system, written by D. Knuth, and by Emacs, written by R. Stallman. Special thanks goes to them, as well as to the C.N.R.S. (the French national institute for scientific research), which employs me and authorized me to freely distribute this program. Further thanks go to the contributors below.

B.3.1. Developers of TEXMACS

- Andrey Grozin has constantly helped us with many issues: interfaces to several computer algebra systems, support for Cyrillic, tools for the manipulation of dictionaries, etc.
- David Allouche replaced the gencc preprocessor by the more standard C++ template system. He also made many other patches, bug reports and he did a lot of the administration of TeXmacs.
- Dan Grayson helped me to implement communications with computer algebra systems via pipes. He also provided some money support for TEXMACS, and he made many useful comments and suggestions.

- Karim Belabas designed and developed with me the first protocol for interfacing TEXMACS with scientific computation or computer algebra systems. He also implemented the interface with the pari system.
- Stéphane Payrard made an important bugfix for destroying windows.
- Michael Graffam for his help with the GNU Octave interface.
- Michael Lachmann for his work on the upcoming GNU R interface.
- Gwenael Gabard for some fixes in the LATEX to TEXMACS converter.
- Felix Breuer for his help on XML support and a donation.
- Igor V. Kovalenko for his help on debugging TeXmacs and a few patches.
- Gareth McCaughan made several patches and comments.
- Jonas Lööf for a precise installation procedure on Cygwin.
- Rob Clark made a patch which improves the system time support.

B.3.2. Administration of TEXMACS and material support

- Jean-Claude Fernandez, Fabien Salvi and the other persons from the CRI host and administrate the TEXMACS website.
- Álvaro Tejero Cantero maintains up the TEXMACS Wiki.
- Loic Dachary made TEXMACS accessible on Savannah.

B.3.3. Porting TEXMACS to other platforms

- Marciano Siniscalchi ported TEXMACS to Cygwin.
- Martin Costabel ported TEXMACS to MacOSX.
- Bruno Haible helped with porting TEXMACS to the SUN system.
- Dan Martens and Stéphane Payrard are working on a Windows port.

B.3.4. Contributors to TEXMACS packages

- Ralf Treinen maintains the Debian package for TEXMACS.
- Christophe Merlet and Bo Forslund helped with making a portable RPM package.
- Lenny Cartier maintains the TEXMACS RPM for Mandrake Cooker.
- Jean Pierre Demailly and Yves Potin made TEXMACS part of the CNDP project to support free software.

B.3.5. Internationalization of TEXMACS

Czech: David Rezac.

Dutch: Joris van der Hoeven.

Finnish: Teemu Ikonen.

French: Michèle Garoche, Joris van der Hoeven.

German: Hans Dembinski, Jan Ulrich Hasecke, Joris van der Hoeven, Thomas Langen, Ralf Treinen.

Hungarian: András Kadinger.

Italian: Xav and Daniele Pighin.

Polish: Robert Janusz.

Portuguese: Márcio Laurini and Alexandre Taschetto de Castro.

Romanian: Dan Ignat.

Russian: Andrey Grozin.

Spanish: Álvaro Cantero Tejero, Pablo Ruiz Múzquiz, David Moriano Garcia, Offray Vladimir Luna Cárdenas.

Swedish: Harald Ellmann.

Ukrainian: Volodymyr Lisivka.

B.3.6. Other contributors

Final thanks go to all others who have contributed to TEXMACS, for instance by sending bug reports or by giving suggestions for future releases: Alexandre Abbes, Alessio Abogani, Till Adam, Murali Agastya, Guillaume Allègre, Larry D'Anna, Eizo Akiyama, Doublet Alban, Tom Alsberg, James Amundson, Ayal Anis, Javier Arantegui Jimenez, André Arnold, Uwe Assmann, Philippe Audebaud, Daniel Augot, Olaf Bachmann, Franky Backeljauw, Nick Bailey, Pierre Barbier de Reuille, Giovanni Maniscalco Basile, Luc Béhar, Odile Bénassy, Paul Benham, Roy C. Bentley, Attila Bergou, Christophe Bernard, Konrad Bernloehr, Karl Berry, Matthias Berth, Cédric Bertolini, Matthew Bettencourt, Anne-Laure Biolley, Benedikt Birkenbach, Jim Blandy, Christof Boeckler, Mohsen Bouaissa, Thierry Bouche, Adrien Bourdet, Didier Bretin, Jean-Yves Briend, Simon Britnell, Alexander M. Budge, Yoel Callev, Niclas Carlsson, Dominique Caron, António Carvalho, Michel Castagner, Topher Cawlfield, Carlo Cecati, Henri Cohen, Teddy Fen-Chong, Dominique Colnet, Claire M. Connelly, Christoph Conrad, Riccardo Corradini, Paulo Correia, Olivier Cortes, Robert J. Cristel, Maxime Curioni, Jason Dagit, Mike Davidson, Jean-Pierre Demailly, Peter Denisevich, Alban Doublet, Steingrim Dovland, Michael John Downes, Benjamin Drieu, Amit Dubey, Daniel Duparc, Tim Ebringer, Magnus Ekdahl, Ulf Ekström, Robin Fairbairns, Tony Falcone, Hilaire Fernandes, Juan Flynn, Jens Finke, Thomas Fischbacher, Cedric Foellmi, Christian Forster, Charlie Fortner, Stefan Freinatis, Michael P Friedlander, Nils Frohberg, Rudi Gaelzer, Maciej Gajewski, Lionel Garnier, Björn Gohla,

Patrick Gonzalez, Nirmal Govind, Michael Graffam, Frédéric Grasset, Wilco Greven, Cyril Grunspan, Laurent Guillon, Harri Haataja, Irwan Hadi, James W. Haefner, Ola Hamfors, Aaron Hammack, Guillaume Hanrot, Karl M. Hegbloom, Jochen Heinloth, Ralf Hemmecke, Alain Herreman, Andreas Horn, Chu-Ching Huang, Ed Hurst, Karl Jarrod Hyder, Richard Ibbotson, Benjamin T. Ingram, Alexander Isacson, Michael Ivanov, Maik Jablonski, Frederic de Jaeger, Pierre Jarillon, Paul E. Johnson, Pierre-Henri Jondot, Antoun Kanawati, Tim Kaulmann, Mukund S. Kalisi, Jeremy Kephart, Iwao Kimura, Simon Kirkby, Ronny Klein, Matthias Koeppe, Denis Kovacs, Jeff Kowalczyk, Ralph Krause, Neel Krishnaswami,, Friedrich Laher, Winter Laite, Russell Lang, Christopher Lee, Milan Lehocky, Joerg Lippmann, Pierre Lorenzon, V.S. Lugovsky, Duraid Madina, Yael Maguire, Paul Magwene, Jeremiah Mahler, Vincent Maillot, Giacomo Mallucci, Sylvain Marchand, Bernd Markgraf, Eric Marsden, Chris Marston, Evan Martin, Alisdair McDiarmid, Phil Mendelsohn, Sébastien de Menten, Jean-Michel Mermet, Jon Merriman, Herve le Meur, Amir Michail, Arkadiusz Miskiewicz, Sasha Mitelman, Dirk Moebius, Jack Moffitt, Julian Morrison, Bernard Mourrain, Stephan Mucha, Nathan Myers, Nix N. Nix, Eduardo Nogueira, Immanuel Normann, Jean-Baptiste Note, Ralf Nuetzel, Kostas Oikonomou, Bill Page, Pierre Pansu, Bernard Parisse, Frédéric Parrenin, Fernández Pascual, Yannick Patois, Alen L. Peacock, François Pellegrini, Antonio Costa Pereira, Jacob Perkins, Bernard Perrot, Jean Peyratout, Jacques Peyriere, Yves Pocchiola, Benjamin Poussin, Isaías V. Prestes, Rui Prior, Julien Puydt, Nguyen-Dai Quy, Ramakrishnan, Adrien Ramparison, Kenneth Reinhardt, Diego Restrepo, Christian Requena, Chris Retford, Staffan Ringbom, Will Robinson, Juan Pablo Romero, Juergen Rose, Mike Rosellini, Filippo Rusconi, Philippe Sam-Long, Duncan Sands, Breton Saunders, Claire Sausset, David Sauzin, Gilles Schaeffer, Guido Schimmels, Rainer Schöpf, David Schweikert, Rui Miguel Seabra, Sami Sieranoja,, Vasco Alexandre da Silva Costa, Marciano Siniscalchi, Daniel Skarda, Murray Smigel, Luke Snow, Rodney Sparapani, Bas Spitters, Bas Spitters, Starseeker, Harvey J. Stein, Bernard Stloup, Peter Stoehr, James Su, Przemyslaw Sulek, Ben Sussman, Roman Svetlov, Milan Svoboda, Dan Synek, Pan Tadeusz, Sam Tannous, John Tapsell, Dung TaQuang, Gerald Teschl, Eric Thiébaut, Nicolas Thiery, Helfer Thomas, Reuben Thomas, Kurt Ting, Janus N. Tøndering, Marco Trevisani, Andreas Umbach, Miguel A. Valle, Rémi Vanicat, Harro Verkouter, Sawan Vithlani, Guy Wallet, Adam Warner, Thomas Wawrzinek, Maarten Wegewijs, Lars Willert, Grayson Williams, Ben Wise, Damien Wyart, Volker Zell, Oleg Zhirov, Richard Zidlicky, Sascha Ziemann, Reinhard Zierke, Paul Zimmermann.

B.3.7. Contacting us

You can either contact us by email at

```
contact@texmacs.org
```

or by regular mail at

```
Joris van der Hoeven
Dépt. de Mathématiques (Bât. 425)
Université Paris-Sud
91405 Orsay Cedex
France
```

There are also several TEXMACS mailing lists:

```
texmacs-users@texmacs.org
texmacs-info@texmacs.org
texmacs-dev@gnu.org
```

B.4. Important changes in TEXMACS

Below, we briefly describe the most important changes which have occurred in TEXMACS since version 0.3.3.15. We also maintain a more detailed change log.

In general, when upgrading to a new version, we recommend you to make backups of your old TEXMACS files before opening them with the newer version of TEXMACS. In the unlikely case when your old file does not open in the correct way, please send a bug report to

`bugs@texmacs.org`

and send your old document as an attached file. Do not forget to mention your version of TEXMACS and the system you are using.

B.4.1. Keyboard (1.0.0.11 – 1.0.1)

The TEXMACS keybindings have been rationalized. Here follows a list of the major changes:

- The `E-` prefix has been renamed to `M-`.
- `escape` is equivalent to `M-` and `escape` `escape` to `A-`.
- Mode dependent commands are now prefixed by `A-`. In particular, accents are typed using `A-` instead of `E-`.
- Variants are now obtained using `tab` instead of `*` and you can circle back using `shift-tab`.
- Greek characters are now typed using `A-C-`, `F5`, or the hyper modifier, which can be configured in Edit → Preferences. You may also obtain Greek characters as variants of Latin characters. For instance, `p tab` yields π.
- The signification of the cursor keys in combination with control, alt and meta has changed.

You may choose between several "look and feels" for the keyboard behaviour in Edit → Preferences → Look and feel. The default is Emacs, but you may choose Old style if you want to keep the behaviour to which you may be used now.

B.4.2. Menus (1.0.0.7 – 1.0.1)

Several changes have been made in the menus. Here follows a list of the major changes:

- Buffer has been renamed as Go.
- Several items from File have been moved to View.
- The Edit → Import and Edit → Export items have been moved to Tools → Selections.
- The Insert menu has been split up into the menus Insert, Text and Mathematics.
- The Text and Paragraph menus have been merged together in one Format menu.
- Options has been spread out across Document, View, Tools and Edit → Preferences.

B.4.3. Style files (1.0.0.4)

Many changes have been made in the organization of the TEXMACS style files. Personal style files which depend on intermediate TEXMACS packages may require some slight adaptations.

We are working towards a stabilization of the standard style files and packages. At the end of this process, it should be easy to adapt existing LATEX style files for journals to TEXMACS by customizing these standard style files and packages. As soon as we have time, we plan to provide online documentation on how to do this at Help → Online documentation.

B.4.4. Tabular material (0.3.5)

The way tabular material is treated has completely changed. It has become much easier to edit tables, matrices, equation arrays, etc. Also, many new features have been implemented, such as background color, border, padding, hyphenation, subtables, etc. However, the upgrading of old tabular material might sometimes be erroneous, in which case we invite you to submit a bug report.

B.4.5. Document format (0.3.4)

The TeXmacs document format has profoundly changed in order to make TeXmacs compatible with XML in the future. Most importantly, the old style environments like

```
<assign|env|<environment|open|close>>,
```

which are applied via matching pairs `<begin|env>text<end|env>`, have been replaced by macros

```
<assign|env|<macro|body|open<body>close>>,
```

which are applied via single macro expansions `<expand|env|text>`. Similarly, matching pairs `<set|var|val>text<reset|var>` of environment variable changes are replaced by a `<with|var|val|text>` construct (close to XML attributes). From a technical point of view, these changes lead to several complications if the `text` body consists of several paragraphs. As a consequence, badly structured documents may sometimes display differently in the new version (although I only noticed one minor change in my own documents). Furthermore, in order to maintain the higher level of structure in the document, the behaviour of the editor in relation to multiparagraph environments has slightly changed.

Appendix C

Contributing to GNU TEXMACS

C.1. Use TEXMACS

One of the best ways to contribute to GNU TEXMACS is by using it a lot, talk about it to friends and collegues, and to report me about bugs or other unnatural behaviour. Please mention the fact that you wrote articles using TEXMACS when submitting them. You can do this by putting the `made-by-TeXmacs` tag somewhere inside your title using Text → Title → TeXmacs notice.

Besides these general (but very important) ways to contribute, your help on the more specific subjects below would be appreciated. Don't hesitate to contact us if you want to contribute to these or any other issues. In the Help menu you can find documentation about the source code of TEXMACS, its document format, how to write interfaces with other formats, and so on.

C.2. Making donations to the TEXMACS project

Making donations to TeXmacs through the SPI organization

One very important way to support TEXMACS is by donating money to the project. TEXMACS is currently one of the projets of SPI (Software in the Public Interest; see `http://www.spi-inc.org`). You may make donations of money to TeXmacs via this organization, by noting on your check or e-mail for wire transfers that your money should go to the TeXmacs project. You may also make donations of equipment or services or donations through vendors. See the SPI website for more information. We will maintain a webpage with a list of donors soon (if you agree to be on the list).

Details on how to donate money

To make a donation, write a check or money order to:

> *Software in the Public Interest, Inc.*

and mail it to the following address:

> Software in the Public Interest, Inc.
> P.O. Box 502761
> Indianapolis, IN 46250-7761
> United States

To make an electronic transfer (this will work for non-US too), you need to give your bank the routing number and account number as follows:

> The SPI bank account is at American Express Centurion Bank.
> `Routing Number: 124071889`
> `Account Number: 1296789`

Don't forget to note on your check or e-mail for wire transfers that the money should be spent on the TeXmacs projet. In addition you may specify a more specific purpose on which you would like us to spend the money. You may also contact us for a more detailed discussion on this issue.

Important notes

Let the SPI Treasurer (`treasurer@spi-inc.org`) know if you have problems. When you have completed the electronic wire, please send a copy of the receipt to the above address so there is a copy of your donation. The copy you send to the treasurer is important. You may also want to contact the TeXmacs team in order to make sure that the money arrived on the TeXmacs account.

Note: *The SPI address and account numbers may change from time to time. Please do not copy the address and account numbers, but rather point to the page* `http://www.spi-inc.org/donations` *to ensure that donors will always see the most current information.*

Donations in Europe can be done through our partner in Germany, ffis e.V. If you are interested in using their bank account (to save international money transfer costs), please check the instructions on `http://www.ffis.de/Verein/spi-en.html`.

C.3. CONTRIBUTE TO THE GNU TEXMACS DOCUMENTATION

There is a high need for good documentation on TEXMACS as well as people who are willing to translate the existing documentation into other languages. The aim of this site is to provide high quality documentation. Therefore, you should carefully read the guide-lines on how to write such documentation.

C.3.1. Introduction on how to contribute

High quality documentation is both a matter of content and structure. The content itself has to be as pedagogic as possible for the targeted group of readers. In order to achieve this, you should not hesitate to provide enough examples and illustrative screen shots whenever adequate. Although the documentation is not necessarily meant to be complete, we do aim at providing relatively stable documentation. In particular, you should have checked your text against spelling errors. The more experimental documentation should be put in the `incoming` directory or on the TEXMACS Wiki.

It is also important that you give your documentation as much structure as possible, using special markup from the `tmdoc` style file. This structure can be used in order to automatically compile printable books from your documentation, to make it suitable for different ways of viewing, or to make it possible to efficiently search a certain type of information in the documentation. In particular, you should always provide copyright and license information, as well as indications on how to traverse your documentation, if it contains many files.

C.3.2. Using CVS

The present TEXMACS documentation is currently maintained on `http://savannah.gnu.org` using CVS (Concurrent Version System). In order to contribute, you should first create an account there. When you are done with this, you should send me an email at `vdhoeven@texmacs.org` and ask me to add your name to the list of documenters. After that, you will be able to modify the documentation using CVS. For information on how to do that, you should consult `http://savannah.gnu.org/cvs/?group_id=1747`.

In fact, the CVS system is not ideal for our documentation purpose, because it is not very dynamic. In the future, we plan to create a dedicated publication website, which will allow you to save documents directly to the web. It should also allow the automatic conversion of the documentation to other formats, the compilation of books, etc.

C.3.3. Conventions for the names of files

Most documentation should be organized as a function of the topic in a directory tree. The subdirectories of the top directory are the following:

devel Documentation for developers.

examples Examples of TEXMACS documents.

incoming Incoming documentation, which is still a bit experimental.

main The main documentation.

meta How to write documentation and the compilation of documentation.

Please try to keep the number of entries per directory reasonably small.

File names in the main directory should be of the form `type-name.language.tm`. In the other directories, they are of the form `name.language.tm`. Here `type` is a major indication for the type of documentation; it should be one of the following:

adv Documentation for advanced users.

man For inclusion in the TEXMACS manual.

tut For inclusion in the TEXMACS tutorial.

You should try to keep the documentation on the same topic together, regardless of the type. Indeed, this allows you to find more easily all existing documentation on a particular topic. Also, it may happen that you want to include some documentation which was initially meant for the tutorial in the manual. The `language` in which is the documentation has been written should be a two letter code like `en`, `fr`, etc. The main `name` of your file should be the same for the translations in other languages. For instance, `man-keyboard.en.tm` should not be translated as `man-clavier.fr.tm`.

C.3.4. Copyright information & the Free Documentation License

All documentation on the `texmacs-doc` site falls under the GNU Free Documentation License. If you write documentation for TEXMACS on this site, then you have to agree that it will be distributed under this license too. The copyright notice

```
Permission is granted to copy, distribute and/or modify this
document under the terms of the GNU Free Documentation License,
Version 1.1 or any later version published by the Free Software
Foundation; with no Invariant Sections, with no Front-Cover
Texts, and with no Back-Cover Texts. A copy of the license
is included in the section entitled "GNU Free Documentation
License".
```

should be specified at the end of *each* file. This should be done inside the `tmdoc-license` macro, in a similar way as at the end of the present document. When automatically generating a printed book from several documentation files, this will enable us to include the license only once.

You keep (part of) the copyright of all documentation that you will write for TEXMACS on the official `texmacs-doc` site. When you or others make additions to (or modifications in, or translations of) the document, then you should add your own name (at an appropriate place, usually at the end) to the existing copyright information. The copyright notice should be specified using the `tmdoc-copyright` function just before the license information at the end of the document. The first argument of this function contains a year or a period. Each remaining argument indicates one of the copyright holders. When combining (pieces of) several documents into another one, you should merge the copyright holders. For cover information (on a printed book for instance), you are allowed to list only the principal authors, but a complete list should be given at a clearly indicated place.

C.3.5. Traversing the TEXMACS documentation

As a general rule, you should avoid the use of sectioning commands inside the TEXMACS documentation and try to write small help pages on well identified topics. At a second stage, you should write recursive "meta help files" which indicate how to traverse the documentation in an automatic way. This allows the reuse of a help page for different purposes (a printed manual, a web-oriented tutorial, etc.).

The `tmdoc` style provides three markup macros for indicating how to traverse documentation. The `traverse` macro is used to encapsulate regions with traversal information. The `branch` macro indicates a help page which should be considered as a subsection and the `continue` macro indicates a follow-up page. Both the `branch` and the `continue` macro take two arguments. The first argument describes the link and the second argument gives the physical relative address of the linked file.

Typically, at the end of a meta help file you will find several `branch` or `continue` macros, inside one `traverse` macro. At the top of the document, you should also specify a title for your document using the `tmdoc-title` macro. When generating a printed manual from the documentation, a chapter-section-subsection structure will automatically be generated from this information and the document titles. Alternatively, one might automatically generate additional buttons for navigating inside the documentation using a browser.

C.3.6. Using the tmdoc style

Besides the copyright information macros and traversal macros, which have been documented before, the `tmdoc` style comes with a certain number of other macros and functions, which you should use whenever appropriate:

`key` This macro is used to indicate keyboard input like `C-x C-s`. The specialized macros `kbd-gen`, `kbd-text`, `kbd-math`, `kbd-symb`, `kbd-big`, `kbd-large`, `kbd-ia`, `kbd-exec` and `kbd-table` are used for keyboard input corresponding to a specific type of action or mode. For instance, `kbd-math` corresponds to keyboard shortcuts for mathematical operations, such as `A-f`, which starts a fraction.

`menu` This function with an arbitrary number of arguments indicates a menu like File or Document → Language. Menu entries are automatically translated by this function.

`markup` This macro is used in order to indicate a macro or a function like `section`.

`tmstyle` This macro indicates the name of a TEXMACS style file or package like `article`.

`tmpackage` This macro indicates the name of a TEXMACS package like `std-markup`.

`tmdtd` This macro indicates the name of a TEXMACS d.t.d. like `number-env`.

Notice that the contents of none of the above tags should be translated into foreign languages. Indeed, for menu tags, the translations are done automatically, so as to keep the translations synchronized with the translations of the actual TEXMACS menus. In the cases of markup, styles, packages and d.t.d.s, it is important to keep the original name, because it often corresponds to a file name.

The following macros and functions are used for linking and indexing purposes, although they should be improved in the future:

`simple-link` This macro takes an URL x as argument and is a hyperlink with name and destination x.

`hyper-link` This macro is a usual hyperlink.

`concept-link` This macro takes a concept as argument. Later on an appropriate hyperlink might be created automatically from this and the other documentation.

`only-index` Index a simple string.

`def-index` Definition of a new concept; the text is printed in italic and indexed.

`re-index` Reappearance of an already defined concept; the text is printed in roman and put in the index.

The following tags are also frequently used:

`icon` Link to an icon in a central directory like `$TEXMACS_PATH/doc/images/pixmaps`.

`screenshot` Link to a screenshot. The actual screenshots are stored in a central directory like `$TEXMACS_PATH/doc/images/screenshots`.

`scheme` The SCHEME language.

`framed-fragment` For displaying a piece of code in a nice frame.

`scheme-code` For multi-paragraph SCHEME code.

`tm-fragment` For a piece of TEXMACS markup code in SCHEME format.

`descriptive-table` For descriptive tables; such tables can be used to document lists of keyboard shortcuts, different types of markup, etc.

The `tmdoc` style inherits from the `generic` style and you should use macros like `em`, `verbatim`, `itemize`, etc. from this style whenever appropriate.

C.4. INTERNATIONALIZATION

The support of a maximal number of foreign languages is another major challenge in which your help would be appreciated. Making the translations to support a new language usually requires several days of work. We therefore recommend you to find some friends or collegues who are willing to help you.

The procedure for adding a new language is as follows

- You copy the file `english-new.scm` to `english-`*`yourlanguage`*`.dic` in `langs/natural/dic` and fill out the corresponding translations. You may want to use Andrey Grozin's dictionary tool at

  `http://www.texmacs.org/Data/dictool.py.gz`

  In order to use it, may sure that Python is installed on your system, download the file, gunzip it, make it executable and run it.

- You tell me about any special typographical rules in your language and handy keystrokes for producing special characters.

- I take care of the hyphenation and typographical issues, but you test them.

- If you have enough time, you may also consider the translation of (part of) the existing documentation.

Of course, the support for languages get out of date each time that new features are added to TEXMACS. For this reason, we also maintain a file `miss-english-`*`yourlanguage`*`.dic` with all missing translation for your language, once that it has been added. Please do not hesitate to send inclomplete versions of `english-`*`yourlanguage`*`.dic` or `miss-english-`*`yourlanguage`*`.dic`; someone else may be willing to complete them.

C.5. WRITING DATA CONVERTERS

If you are familiar with TEX, LATEX, Html, Xml, Sgml, Mathml, Pdf, Rtf, or any other frequently used data format, please consider contributing to writing good converters for one or more of these formats.

Writing a specific converter for Pdf should not be very difficult, by adapting the file `src/Window/PsDevice/printer.cc`. Converters for other formats may be more complicated to write, and might sometimes require a closer collaboration with the main TEXMACS authors. In Help → Source code → Data format you will find details about the TEXMACS data format and in Help → Source code → Data conversion we give some suggestions which might be helpful for these projects.

C.6. PORTING TEXMACS TO OTHER PLATFORMS

Having only access to PC/Linux and SUN systems, I am interested in people who want to port TEXMACS to other Unix systems with X Window and to maintain the corresponding distributions. If you want to do this, you should take a look at the files

```
configure.in
src/Basic/fast_alloc.cc
```

Specialists on `autoconf`, redhat and rpm packages are also welcome to communicate their suggestions, patches, etc.

Besides porting TeXmacs to other Unix-based systems, it would be nice to port TeXmacs to Windows (and Mac OS). Please join the `texmacs-dev@gnu.org` mailing list if you want to help. Discussions have been going on about how to do the porting and in particular about which portable graphical user interface (like Gtk, Qt, Wxwindows or GNUstep) we should use. Our strategy will be to first put all GUI dependent code in a cleanly specified TMGUI API and then do the actual porting. In fact, this will allow us to support multiple graphical toolkits. More details can be found in the archives of the `texmacs-dev@gnu.org` mailing list.

C.7. Interfacing TeXmacs with other systems

It is quite easy to write interfaces between TeXmacs and computer algebra systems or other scientific programs with structured output. Please consider writing interfaces between TeXmacs and your favorite system(s). TeXmacs has already been interfaced with several other free systems, like Giac, Macaulay 2, Maxima, GNU Octave, Pari, Qcl, gTybalt, Yacas. Detailed documentation on how to add new interfaces is available in the Help → Interfacing menu.

C.8. Become a TeXmacs developer

Apart from the kind of contributions which have been described in more detail above, there are many more issues where your help would be appreciated. Please take a look at our plans for the future for more details. Of course, you should feel free to come up with your own ideas and share them with us on the `texmacs-dev@gnu.org` mailing list!

Appendix D

Interfacing TeXmacs with other programs

D.1. Studying the "mycas" example

The best way to start implementing a new interface with TeXmacs is to take a look at the sample "computer algebra system" `mycas`, which can be found in the directory `$TEXMACS_PATH/misc/mycas`. The file `mycas.cc`, which is listed at the end of this section, contains a very simple program which can be interfaced with TeXmacs. In order to test the program, you should compile it using

```
g++ mycas.cc -o mycas
```

and move the binary `mycas` to some location in your path. When starting up TeXmacs, you should then have a Mycas entry in the Insert → Session menu.

D.2. Studying the source code step by step

Let us study the source code of `mycas` step by step. First, all communication takes place via standard input and output, using pipes. In order to make it possible for TeXmacs to know when the output from your system has finished, all output needs to be encapsulated in blocks, using three special control characters:

```
#define DATA_BEGIN   ((char) 2)
#define DATA_END     ((char) 5)
#define DATA_ESCAPE  ((char) 27)
```

The `DATA_ESCAPE` character followed by any other character c may be used to produce c, even if c is one of the three control characters. An illustration of how to use `DATA_BEGIN` and `DATA_END` is given by the startup banner:

```
int
main () {
  cout << DATA_BEGIN << "verbatim:";
  cout << "-------------------------------------------------------\n";
  cout << "Welcome to my test computer algebra system for TeXmacs\n";
  cout << "This software comes with no warranty whatsoever\n";
  cout << "(c) 2001  by Joris van der Hoeven\n";
  cout << "-------------------------------------------------------\n";
  next_input ();
  cout << DATA_END;
  fflush (stdout);
```

The first line of `main` says that the startup banner will be printed in the "verbatim" format. The `next_input` function, which is called after outputting the banner, is used for printing a prompt and will be detailed later. The final `DATA_END` closes the startup banner block and tells TeXmacs that `mycas` is waiting for input. Don't forget to flush the standard output, so that TeXmacs will receive the whole message.

The main loop starts by asking for input from the standard input:

```
  while (1) {
    char buffer[100];
    cin >> buffer;
    if (strcmp (buffer, "quit") == 0) break;
```

The output which is send back should again be enclosed in a DATA_BEGIN-DATA_END block.

```
    cout << DATA_BEGIN << "verbatim:";
    cout << "You typed " << buffer << "\n";
```

Inside such a block you may recursively send other blocks, which may be specified in defferent formats. For instance, the following code will send a LaTeX formula:

```
    cout << "And now a LaTeX formula: ";
    cout << DATA_BEGIN << "latex:" << "$x^2+y^2=z^2$" << DATA_END;
    cout << "\n";
```

For certain purposes, it may be useful to directly send output in TeXmacs format using a Scheme representation:

```
    cout << "And finally a fraction ";
    cout << DATA_BEGIN << "scheme:" << "(frac \"a\" \"b\")" <<
DATA_END;
    cout << ".\n";
```

In order to finish, we should again output the matching DATA_END and flush the standard output:

```
    next_input ();
    cout << DATA_END;
    fflush (stdout);
  }
  return 0;
}
```

Notice that you should never output more than one DATA_BEGIN-DATA_END block. As soon as the first DATA_BEGIN-DATA_END block has been received by TeXmacs, it is assumed that your system is waiting for input. If you want to send several DATA_BEGIN-DATA_END blocks, then they should be enclosed in one main block.

A special "channel" is used in order to send the input prompt. Channels are specified as special DATA_BEGIN-DATA_END blocks:

```
static int counter= 0;

void
next_input () {
  counter++;
  cout << DATA_BEGIN << "channel:prompt" << DATA_END;
  cout << "Input " << counter << "] ";
}
```

Inside the prompt channel, you may again use DATA_BEGIN-DATA_END blocks in a nested way. This allows you for instance to use a formula as a prompt. There are three standard channels:

`output.` The default channel for normal output.

`prompt.` For sending input prompts.

`input.` For specifying a default value for the next input.

D.3. GRAPHICAL OUTPUT

It is possible to send postscript graphics as output. Assume for instance that you have a picture `picture.ps` in your home directory. Then inserting the lines

```
cout << "A little picture:\n";
cout << DATA_BEGIN << "ps:";
fflush (stdout);
system ("cat $HOME/picture.ps");
cout << DATA_END;
cout << "\n";
```

at the appropriate place in the main loop will display your image in the middle of the output.

D.4. THE COMPLETE LISTING

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>

#define DATA_BEGIN   ((char) 2)
#define DATA_END     ((char) 5)
#define DATA_ESCAPE  ((char) 27)

static int counter= 0;

void
next_input () {
  counter++;
  cout << DATA_BEGIN << "channel:prompt" << DATA_END;
  cout << "Input " << counter << "] ";
}

int
main () {
  cout << DATA_BEGIN << "verbatim:";
  cout << "-----------------------------------------------------\n";
  cout << "Welcome to my test computer algebra system for TeXmacs\n";
  cout << "This software comes with no warranty whatsoever\n";
  cout << "(c) 2001  by Joris van der Hoeven\n";
  cout << "-----------------------------------------------------\n";
  next_input ();
  cout << DATA_END;
  fflush (stdout);
```

```
  while (1) {
    char buffer[100];
    cin >> buffer;
    if (strcmp (buffer, "quit") == 0) break;
    cout << DATA_BEGIN << "verbatim:";
    cout << "You typed " << buffer << "\n";

    cout << "And now a LaTeX formula: ";
    cout << DATA_BEGIN << "latex:" << "$x^2+y^2=z^2$" << DATA_END;
    cout << "\n";

    cout << "And finally a fraction ";
    cout << DATA_BEGIN << "scheme:" << "(frac \"a\" \"b\")" << DATA_END;
    cout << ".\n";

    next_input ();
    cout << DATA_END;
    fflush (stdout);
  }
  return 0;
}
```

D.5. Writing your first interface with TEXMACS

In order to write your first interface to TEXMACS, we recommend you to follow the following steps:

1. Create a `--texmacs` option for your program which will be used for calling your program from inside TEXMACS.
2. Modify your output routines in a such a way that the appropriate output is send to TEXMACS when your program is started with the `--texmacs` option.
3. Create a `mycas` script in your path which executes your program with the `--texmacs` option.

After doing this, your program will be available under the name Mycas in Insert → Session. We will explain later how to make your system listed under its own name, how to customize it, and how to get the interface incorporated into the main TEXMACS distribution.

Usually, step 2 is the most complicated one and the time it will cost you depends on how your system was designed. If you designed clean output routines (including the routines for displaying error messages), then it usually suffices to modify these by mimicking the `mycas` example and reusing existing LATEX output routines, which most systems provide.

At the moment, we only implemented LATEX as a standard transmission format for mathematical formulas, because this is the format which is most widely used. In the future, we intend to implement more semantically secure formats, and we recommend you to keep in mind the possibility of sending your output in tree format.

Nevertheless, we enriched standard LATEX with the `\*` and `\bignone` commands for multiplication and closing big operators. This allows us to distinguish between

```
a \* (b + c)
```

(or a multiplied by $b+c$) and

```
f(x + y)
```

(or f applied to $x+y$). Similarly, in

```
\sum_{i=1}^m a_i \bignone + \sum_{j=1}^n b_j \bignone
```

the `\bignone` command is used in order to specify the scopes of the `\sum` operators.

It turns out that the systematic use of the `\*` and `\bignone` commands, in combination with clean LaTeX output for the remaining constructs, makes it *a priori* possible to associate an appropriate meaning to your output. In particular, this usually makes it possible to write additional routines for copying and pasting formulae between different systems.

D.6. SUPPORTING YOUR SYSTEM INSIDE TEXMACS

Assume that you have successfully written a first interface with TeXmacs as explained in the previous section. Then it is time now to include support for your system in the standard TeXmacs distribution, after which further improvements can be made.

The easiest way to get your system supported is to send us a copy of your software with the new interface with the request to incorporate the necessary support into TeXmacs. By default, we will call your program using the `--texmacs` option; if this is not appropriate, then you should say so.

You may also write the support for your system yourself. Assuming that the name of your system is `yourcas`, you should follow the following steps:

1. Create a file of the form `yourcas.scm` in the directory `$TEXMACS_PATH/progs/cas`. You should load this file on startup, by editing the global `init-texmacs.scm` (or your personal `my-init-texmacs.scm`) initialization file.

2. Include a line of the form

   ```
   (connection-declare "yourcas" "shell-cmd")
   ```

   in `yourcas.scm`, where `shell-cmd` is the shell command to launch your system (including special command line options which allow your program to recognize that it was called from inside TeXmacs). Usually `shell-cmd` is `yourcas --texmacs`.

3. Add a line of the form

   ```
   (connection-format "yourcas" "input-format" "output-format")
   ```

   to `yourcas.scm` if your input and output is not merely verbatim text. Usually, the `input-format` is `verbatim` and the `output-format` is `generic`. Other avilable formats are `scheme`, `latex`, `html` and `ps`.

4. Add a command like

   ```
   (set! session-menu (menu-merge session-menu
     '(("yourcas" (make-session "yourcas")))))
   ```

   to `yourcas.scm`, in order to be able to start a TeXmacs-session from the Insert → Session menu, in which you can evaluate expressions of your language. You may also redefine the keyboard for such sessions.

5. Create a style file `yourcas.ts` in `$TEXMACS_PATH/packages/session` environments 'yourcas-session', 'yourcas-input' and 'output-yourcas'. Alternatively, you may put this file in `$TEXMACS_HOME_PATH/packages`.

6. If everything works well, and you wish to make it possible for others to use your system inside the official TeXmacs distribution, then contact me at `vdhoeven@texmacs.org`.

D.7. Further customization of the interface

Having written a working interface between your system and TeXmacs, you may want to improve it further. Below, we will discuss a few directions for possible improvement.

First of all, you may want to customize the keyboard behavior inside a `yourcas`-session and add appropriate menus. The mechanisms for doing that are described in the chapter about the Guile/Scheme extension language and you may add such support to the file `yourcas.scm`. We recommend you to take a look at some of the files in the directory `$TEXMACS_PATH/progs/cas` in order to see how this is done for other systems.

Certain output from your system might require a special markup. For instance, assume that you want to associate an invisible type to each subexpression in the output. Then you may create a macro `exprtype` with two arguments in `yourcas.ts` and send LaTeX expressions like `\exprtype{1}{Integer}` to TeXmacs during the output.

In the case when you connected your system to TeXmacs using pipes, you may directly execute TeXmacs commands during the output from your system by incorporating pieces of code of the form

```
[DATA_BEGIN]command:scheme-program[DATA_END]
```

in your output. Inversily, when the cursor is inside a session of your system, you may use the scheme command

```
(extern-exec cas-command)
```

in order to execute a command of your system.

D.8. Linking your system as a dynamic library

Instead of connecting your system to TeXmacs using a pipe, it is also possible to connect it as a dynamically linked library. Although communication through pipes is usually easier to implement, more robust and compatible with gradual output, the second option is faster.

D.9. Connections via dynamically linked libraries

Let us now describe the steps you have to go through in order to link your system as a dynamic library.

1. Modify the architecture of your system in such a way that the main part of it can be linked as a shared library; your binary should typically become a very small program, which handles verbatim input and output, and which is linked with your shared library at runtime.

2. Copy the include file `$TEXMACS_PATH/include/TeXmacs.h` into the include directory of your system's source and write the input/output routines as required by the last TeXmacs communication protocol as explained below.

3. Include a line of the form

   ```
   (package-declare "yourcas" "libyourcas.so" "get_name_package"
   "init")
   ```

   in a file `yourcas.scm` like in steps 2, 3 and 4 above. Here `libyourcas.so` is the corresponding shared library, `get_name_package` the function which will be called by TEXMACS in order to link your system to TEXMACS, and `init` some initialization string for your package.

4. Proceed in a similar way as in the case of pipes.

D.10. The TeXmacs communication protocol

The TEXMACS communication protocol is used for linking libraries dynamically to TEXMACS. The file `$TEXMACS_PATH/include/TeXmacs.h` contains the declarations of all data structures and functions used by the protocol. Actually, we foresee a succession of different protocols. Each of these protocols have the abstract data structures `TeXmacs_exports` and `package_exports` in common, with information about the versions of the protocol, TEXMACS and your package.

The n-th concrete version of the communication protocol should provide two data structures `TeXmacs_exports_n` and `package_exports_n`. The first structure contains all routines and data of TEXMACS, which may be necessary for the package. The second structure contains all routines and data of your package, which should be visible inside TEXMACS.

In order to link your system to TEXMACS, you have to implement a function

```
package_exports* get_my_package (int version);
```

This function takes the highest TEXMACS communication protocol supported by your TEXMACS system on input. It should return a pointer to an instance of a concrete structure `package_exports_n`, where `n` is inferior or equal to `version`.

D.11. Version 1 of the TeXmacs communication protocol

In the first version of the TeXmacs communication protocol, your package should export an instance of the following data structure:

```
typedef struct package_exports_1 {
  char* version_protocol; /* "TeXmacs communication protocol 1" */
  char* version_package;
  char* (*install) (TeXmacs_exports_1* TM, char* options, char**
errors);
  char* (*evaluate) (char* what, char* session, char** errors);
  char* (*execute) (char* what, char* session, char** errors);
} package_exports_1;
```

The string `version_protocol` should contain `"TeXmacs communication protocol 1"` and the string `version_package` the version of your package.

The routine `install` will be called once by TEXMACS in order to initialize your system with options `options`. It communicates the routines exported by TEXMACS to your system in the form of `TM`. The routine should return a status message like

```
"yourcas-version successfully linked to TeXmacs"
```

If installation failed, then you should return `NULL` and `*errors` should contain an error message. Both `what` and the returned string have a special format, in which it is possible to encode arbitrary TEXMACS documents. This format will be explained in the next section.

The routine `evaluate` is used to evaluate the expression `what` inside a TEXMACS-session with name `session`. It should return the evaluation of `what` or `NULL` if an error occurred. `*errors` either contains one or more warning messages or an error message, if the evaluation failed. The command

```
(package-format "yourcas" "input-format" "output-format")
```

is used in order to specify the input and output formats for evaluations, in a similar way as in the case of pipes.

The routine `execute` has a similar specification as `evaluate`, except that it is not used for the evaluation of expressions inside a TEXMACS-session, but rather for other communication purposes between TEXMACS and your package.

Remark D.1. All strings returned by the routines `install`, `evaluate` and `execute`, as well as all warning and error messages should be allocated using `malloc`. They will be freed by TEXMACS using `free`.

The first version of the TEXMACS communication protocol also requires TEXMACS to export an instance of the data structure

```
typedef struct TeXmacs_exports_1 {
  char* version_protocol; /* "TeXmacs communication protocol 1" */
  char* version_TeXmacs;
} TeXmacs_exports_1;
```

The string `version_protocol` contains the version `"TeXmacs communication protocol 1"` of the protocol and `version_TeXmacs` the current version of TeXmacs.

D.12. Future projects

There are many improvements to be made in the TEXMACS interface to computer algebra systems. First of all, the computer algebra sessions have to be improved (better hyphenation, folding, more dynamic subexpressions, etc.). As to the real interface with computer algebra systems, the upcoming changes fall into two categories:

- Changes to the existing interface.
- Support for communication between computer algebra systems.

An example of a change of the first type would be support for the automatic completion of commands. The second project is far more ambitious and targets a semantically safe way to communicate mathematical data. We plan to write a stand-alone program for this, which can be used independently from TEXMACS.

Index