identified two such projects, these being (1) the specification of a canonical TEX kit, and (2) the implementation of an extended TEX (to be known as e-TEX) based on the present WEB implementation. It was also *agreed* that Marek Ryćko & Bogusław Jackowski would be asked if they were willing to co-ordinate the first of these activities, and that Peter Breitenlohner would co-ordinate the second.

The ideas behind the two proposals are as follows.

(1) The canonical TEX kit: at the moment, the most that can be assumed of any site offering TEX is (a) iniTEX; (b) plain TEX; (c) LATEX; and (d) at least sixteen Computer Modern fonts. Whilst these are adequate for a restricted range of purposes, it is highly desirable when transferring documents from another site to be able to assume the existence of a far wider range of utilities. For example, it may be necessary to rely on BIBTEX, or on MakeIndex; it may be useful to be able to assume the existence of BM2FONT; and so on. Rather than simply say "all of these can be found on the nearest CTAN archive", it would be better if all implementations contained a standard subset of the available tools. It is therefore the aim of this project to identify what the elements of this subset should be, and then to liaise with developers and implementors to ensure that this subset is available for, and distributed with, each TEX implementation.

(2) Extended TEX (e-TEX): whilst the test bed and production system approach is philosophically very sound, the reality at the moment is that the group lacks the resources to bring it to fruition. None the less, there are many areas in which a large group of existing TEX users believe that improvements could be made within the philosophical constraints of the existing TEX implementation. E-TEX is an attempt to satisfy their needs which could be accomplished without a major investment of resources, and which can pursued without the need for additional paid labour.

Finally the group agreed to individually undertake particular responsibilities; these are to be:

Peter Breitenlohner: Remove any existing incompatibilities between TEX–XET and TEX, with the idea of basing further e-TEX developments on TEX–XET; liaise with Chris Thompson concerning portability of the code; produce a catalogue of proposed extensions to e-TEX.

Joachim Lammarsch: liaise with vendors and publishers in an attempt to raise money for the implementation of NTS proper; arrange a further meeting of interested parties; liaise with Eberhard Mattes concerning the present constraints on the unbundling of emTEX; negotiate with leading academics concerning possible academic involvement in the project.

Mariusz Olko: take responsibility for the multilingual aspects of e-TEX and NTS; discuss the possibility of siting the NTS programming team in Poland; discuss the possibility of academic involvement with leading Polish academics.

Bernd Raichle: endeavour to get TEX–XET integrated into the standard UNIX distribution; prepare a list of proposed extensions to e-TEX; lead discussions on NTS-L.

Friedhelm Sowa: primary responsibility for finance; prepare proposals for a unified user interface and for unification of the integration of graphics; liaise with the Czech/Slovak groups concerning possible siting of the NTS programming team in the Czech Republic or Slovakia; discuss possible academic involvement with leading academics.

Philip Taylor: Overall technical responsibility for all aspects of the project; liaise with other potential NTS core group members; prepare and circulate a summary of the decisions of this and future meetings.

◇ Philip Taylor
The Computer Centre, RHBNC
University of London, U.K.
<P.Taylor@Vax.Rhbnc.Ac.Uk>

Software & Tools

Two Extensions to GNU Emacs that Are Useful when Editing TEX Documents

Thomas Becker

Introduction

One of the most outstanding features of the GNU Emacs editor is the fact that it is customizable in the best and widest sense of the word. In this note, we present two extensions to GNU Emacs that are particularly useful when editing TEX or LATEX documents; these extensions were written by the author while typesetting a 574 page book

in LaTeX. The first package actually consists of a single function that provides an intelligent way of automatically blinking matching opening dollars each time a dollar sign is inserted. The second one improves an existing general feature of GNU Emacs, namely, keyboard macros. These are particularly but not exclusively interesting for mathematical typesetting with TeX and LaTeX.

Super-tex-dollar

As a GNU Emacs user, you know that when you insert a closing delimiter such as) in a buffer, Emacs will blink the matching opening delimiter for one second or until new input arrives. In fact, you can declare any character to be a closing delimiter and tell Emacs what the matching opening delimiter is supposed to be. Emacs also knows that there is at least one self-matching delimiter known to humankind, namely, TeX's dollar sign. Emacs' regular tex-mode makes the dollar sign a self-matching delimiter. The effect of this is that each time a dollar is inserted, the preceding dollar will blink. This blinking will skip a dollar that immediately precedes the one that is being inserted. This behavior is undoubtedly helpful when editing TeX or LaTeX documents. I have also seen tex-modes for GNU Emacs that tried to be more intelligent about the dollar sign. However, everything that I have seen thus far along these lines has been, in one way or another, incomplete or outright annoying.

The function super-tex-dollar tries to provide a clean, safe, and intelligent way of dealing with the dollar sign when editing TeX or LaTeX documents. The function is to be bound to the `$`-key whenever a `.tex` file is being visited, so that it is invoked every time a dollar is inserted. (The mini-manual that comes with super-tex-dollar explains how to achieve this.) This is of course the kind of software that should not and does not require studying a manual before it can be used. You install it, continue to work as usual, and see if you like what is happening on your screen. The following short description of super-tex-dollar is meant to help you decide if you want to try this at all.

TeX requires that all open dollars be closed at the end of a paragraph. Therefore, super-tex-dollar's basic strategy is to investigate the dollar situation between the beginning of the current paragraph and the current cursor position (*point* in Emacs terminology) and then decide what to do about the dollar that is being inserted. Now there are quite a few ways to start a paragraph in TeX or LaTeX, many of them unpredictable, so super-tex-dollar simply assumes that there is always at least one blank line between paragraphs. In order to get meaningful results and good performance, you must therefore make sure that a command like `\chapter` in LaTeX is always preceded or followed by a blank line. This is certainly not a bad idea anyway, but if you are not comfortable with it, then super-tex-dollar is not for you.

If super-tex-dollar finds that all opening dollars have been closed in the present paragraph up to the cursor position, then it will simply insert a dollar. When you type the closing dollar after having inserted your math formula, a dollar will be inserted and the opening dollar will blink for one second or until you continue typing. The next opening dollar will once again be inserted plainly. It should be clear that this behaviour gives you a lot more information than Emacs' default blinking as described above; in particular, if you have created a mess by deleting things in previously written text, you can locate the trouble by erasing and reinserting dollars.

Before we discuss super-tex-dollar's handling of `$$`'s, a few comments about displayed formulas in LaTeX are in order. If you are a LaTeX user, then you probably use

```
\begin{displaymath}
⟨formula⟩
\end{displaymath}
```

or `\[`⟨*formula*⟩`\]` to create displayed formulas. It is true that `\begin{math}`⟨*formula*⟩`\end{math}` and `\(`⟨*formula*⟩`\)` are both equivalent to `$`⟨*formula*⟩`$`, while

```
\begin{displaymath}
⟨formula⟩
\end{displaymath}
```

and `\[`⟨*formula*⟩`\]` are not exactly the same as `$$`⟨*formula*⟩`$$`. There are sometimes minuscule differences in vertical spacing, but I do not know of a situation where the double dollar produces something unwanted. The only real difference I can see is that the double dollar is more convenient to type and offers more flexibility because of the `\eqno` feature.

If you type an opening dollar and then another one immediately following it, then super-tex-dollar will insert this second one without any blinking: you have created an opening `$$`. Trying to insert a third dollar following the double dollar will have no effect whatsoever. When you type a dollar after having inserted your displayed formula, this dollar will automatically be doubled and the (first of the)

opening double dollars will blink. Trying to insert a third dollar after the closing double dollar will blink the opening one but not insert anything. In particular, if, out of habit, you close the opening double dollar by typing two dollars in succession, this will have the same effect as typing a single dollar.

If you have typed $\langle formula \rangle$ and then decide that you really want this to be a displayed formula, then you can achieve this by typing two dollars at this point. The first one will of course be interpreted as the closing one for the opening dollar at the beginning of the formula. The second one, however, will cause that opening dollar to blink and be doubled automatically, so that you are now looking at $$\langle formula \rangle$$. Again, trying to insert a third dollar will do nothing but blink the opening double dollar.

There is one situation in connection with double dollars for which there does not seem to be a perfect solution. Suppose you want to type

```
$$
y = \cases{x & if\quad $x>0$\cr
           0 & otherwise.\cr}
$$
```

The first two dollars, i.e., the opening $$, will be inserted plainly. The third dollar will be seen by super-tex-dollar as an attempt to close the double dollar: it will be automatically doubled, and the opening double dollar will blink. To get what you want, you must now delete a character backwards. From then on, however, super-tex-dollar will once again know what is going on. The fourth dollar will be interpreted correctly as the closing for the preceding one. The attempt to insert another dollar immediately following the fourth one will be denied, and you will get the message "Dangling $$. Closing it now would leave an uneven number of $'s in between." When the fifth dollar is inserted, this will again be interpreted as an attempt to close the opening double dollar and handled accordingly by automatic doubling and blinking. Deleting one character backwards will enable you to insert more pairs of single dollars, with the same behavior as in the case of the first pair. Instead of deleting a dollar backwards, you may of course always enforce plain insertion of single dollars by typing `C-q $`.

How does super-tex-dollar cope with garbage encountered when checking the dollars in the current paragraph? When super-tex-dollar encounters a triple dollar, it concludes that no meaningful conclusions are possible. It assumes that all $'s and $$'s have been closed at this point, continues its regular operation based on that assumption, and displays an appropriate warning including the number of the line that contains the triple dollar. I do not know of a situation where the sequence `$xxx$$` — with the first dollar being an opening one — is meaningful in TEX. When super-tex-dollar encounters it, it will implicitly assume that the opening dollar has been closed before the double dollar. It will also display a warning that informs you of the problem and the number of the line where it occurs.

The handling of %, \%, and \$ is as follows. If the cursor position is preceded by a % on the same line, then a $ is inserted like an ordinary character. When super-tex-dollar encounters a % earlier in the paragraph, it ignores the rest of that line. Moreover, it fully recognizes the fact that a \ quotes a $ as well as a %. However, it will see \\$ and \\% as quoted $ and % as well.

The time that it takes super-tex-dollar to decide what to do increases linearly with the length of the region from the beginning of the paragraph to the cursor position, and with the number of dollars therein. A delay is not noticeable under normal circumstances, and it is negligible under all circumstances that are anywhere close to normal (i.e., on today's personal computers and workstations, and assuming that you do not write ridiculously long paragraphs with absurdly many dollar signs). As with Emacs' blinking of matching opening delimiters, the blinking is always interrupted when the user continues to type. The byte-compiled code of super-tex-dollar takes up 2.5 kB when loaded into Emacs. The space consumption of the program at runtime is always negligible: the position of each encountered opening $ or $$ will be forgotten as soon as it has been closed.

For information on how to obtain super-tex-dollar, see Section "Availability" below.

Emacros

When TEX is being criticized for not providing WYSIWYG, TEX buffs like to retort by saying that WYSIWYG is for wimps. I tend to agree. On the other hand, I have had some weak moments when I got tired of typing

```
$$
\begin{array}{rccc}
 : & & \longrightarrow & \\
   & & \longmapsto     &
\end{array}
$$
```

for the umpteenth time just to get something like

$$\begin{array}{rccc} f: & [0,1] & \longrightarrow & [0,1] \\ & x & \longmapsto & x^2. \end{array}$$

Even something like

```
\begin{corollary}

\end{corollary}
```

gets to be a drag after a while. There is of course the possibility of using TEX macros — with parameters if necessary — in this situation. On the other hand, there are very good reasons not to define a TEX macro every time you find yourself typing something more than three times. I was soon led to the conclusion that the appropriate solution in this situation is the use of *keyboard macros* on the editor level, where you issue some short, mnemonic command to insert a long and complicated string, with the cursor moving to a particular position if appropriate.

GNU Emacs provides keyboard macros.[1] However, I soon found out that Emacs' keyboard macros are the only feature that is somewhat underdeveloped in an otherwise perfect editor. I have therefore written a package called *Emacros* that adds a number of conveniences such as easy saving and reloading of macros and help with remembering macronames. A detailed manual comes with the package; in the sequel, we give a short general description of its capabilities.

Emacros' way of saving macro definitions to files is based on the idea that macro definitions should be separated by major modes to which they pertain. The macros used when editing a TEX-file, for example, will not be needed when working on a C-program. Moreover, within each mode, there will be macros that should be available whenever Emacs is in that mode, and others that are relevant for specific projects only. Consequently, each mode should allow one global macro file and several local ones in different directories as needed. This arrangement saves time and space and makes it easy to keep track of existing macro definitions.

A keyboard macro really consists of two components: the (complicated) string which is to be inserted and the (short) command which invokes this insertion. Here, we will refer to the string as the *macro*, and to the command as its *name*. In GNU Emacs, the key sequence `C-x (` starts the definition of a macro: the keystrokes following the command have the usual effect on the current buffer, while they are at the same time memorized to be inserted automatically as a macro later on. The key sequence `C-x )` ends this process; the macro can now be inserted before the cursor by typing `C-x e`. Note that a macro may not only contain self-insert commands, i.e., ordinary text, but arbitrary keyboard input. You can, for example, define a macro that creates

```
\begin{corollary}
-
\end{corollary}
```

on the screen, with the cursor, represented by the underscore, at the beginning of the blank line.

To be able to use the macro after defining another one, it must be given a name. This can be done by means of the Emacs function name-last-kbd-macro. This function is adequate if the macro is to be used in the current session only and if, moreover, there are very few macros around so that one can easily memorize them all. Otherwise, this is where Emacros comes in. The macro can now be named using the new function emacros-name-last-kbd-macro-add. This function first prompts the user for a name, enforcing appropriate restrictions. Next, the function saves the macro definition to a file named *mode*`-mac.el`, where *mode* is the current major mode, for reloading in future sessions. This file can be in the directory for global macros, in which case the macro will be available whenever *mode* is the major mode, or it can be in the current directory, in which case the macro will be locally available whenever *mode* is the major mode and the file that is being visited is from this directory. The function will ask you to choose between `l` for local and `g` for global. When the function is called with prefix argument, then you will be prompted to explicitly enter the name of a file to save the macro to.

Once a macro *macro* has a name *macroname*, this name is in fact a command which causes the macro to be inserted before the cursor: typing `M-x` *macroname* `RET` inserts *macro*. This has the disadvantage that completion takes into account all command names rather than just macro names. Emacros therefore provides a function specifically for executing keyboard macros. As a further convenience for the impatient (which was motivated by the attempt to make macro insertion no more tedious than using a TEX macro), there is a function called emacros-auto-execute-named-macro. This function will prompt for the name of a macro in the minibuffer. The cursor will stay at its position in the current buffer. As soon as the sequence that you have entered matches the name of a macro, this

[1] Using an editor like GNU Emacs to the full extent of its capabilities does of course require some effort and a certain computer maturity; but then, we are not wimps like the rest of them, remember?

macro is inserted and regular editing is resumed without the need to type a RET.

Every time you read a file into Emacs, *Emacros* invokes a function that will load those macros that have been saved to files named *mode*-mac.el in the current directory and in the directory for global macros. Here, *mode* is the major mode which Emacs has chosen for the visited file. Macro files that have been loaded before during the same session will be disregarded. If you have been editing a file and then read another one with a different mode and/or from a different directory, then the macros pertaining to the new file will be loaded, and all others that were loaded previously will remain active as well. If there are not too many macros around, this is probably what you want. In the long run, however, especially when you are one of those users that never leave Emacs, you would end up with all macros being loaded, thus rendering the separation into different files pointless. The function emacros-refresh-macros takes care of this problem. It will erase all previously loaded macros and load the ones pertaining to the current buffer, thus creating the same situation as if you had just started Emacs and read in the file that the current buffer is visiting.

There are three functions that allow you to manipulate macro definitions that have already been saved. The function emacros-rename-macro assigns a new name to a previously named macro, making the change effective in the current session and in the local or global macro file pertaining to the current buffer, as appropriate. The function emacros-move-macro moves macro definitions between the local and global file pertaining to the current buffer. Finally, the function emacros-remove-macro deletes macros from the current macro files and disables them in the current session.

Three functions provide help with keyboard macros. (The manual tells you how to make these available as help options.) The first of these will display in Emacs' help window a list of all currently defined macronames and the corresponding macros. The second one prompts you for a macro and then tells you its name. The third one acts like the second one, except that it also inserts the macro whose name you were asking for after the point in the current buffer, assuming that you were asking because you wanted to use the macro. The possibility to complete when entering the macro makes this an attractive way to insert, making it worthwhile using macros even if you never ever remember the name of one.

When I wrote Emacros, I made a strong effort to conform with Emacs' general style, both in terms of source code and in terms of look-and-feel. Completion is supported whenever an existing macro or macroname is to be entered, defaults are offered whenever there is the remotest chance of anticipating what the user wants to do next, and messages appear whenever the user tries to do something meaningless or dangerous. The byte-compiled code takes up 16 kB; otherwise, the space consumption is only a trifle more than what is needed to store your macros and their names.

Super-tex-dollar and Emacros Combined

There are two things that need to be said about using super-tex-dollar and Emacros together. When a dollar sign occurs in a keyboard macro, it should always be inserted as C-q $ when defining the macro. That way, you do not get the blinking and, possibly, doubling of dollars when the macro is being executed. With this in mind, you will find that the unwanted doubling when placing single dollars between a pair of double dollars (see Section "Super-tex-dollar" above) becomes a rather rare occurrence. For example, I have a macro named cas, so that — with the function emacros-auto-execute-named-macro bound to M-\ — I can type M-\ cas, and voilà, I have

```
\cases{_& if\quad $ $\cr
       & if\quad $ $\cr
       & otherwise.\cr}
```

on the screen, with the cursor in the position indicated by the underscore. All I have to do now is to fill in things and perhaps delete or copy the middle line. The whole thing is most likely to be in a displayed formula; the double dollars will now be handled correctly by super-tex-dollar.

Availability

Both the Superdollar package and the Emacros package are available via ftp from

alice.fmi.uni-passau.de

where they are to be found in the directory pub/emacs_contrib. The Emacros package will also be made part of the GNU Emacs distribution in the near future. Both packages come with manuals explaining installation and usage.

◇ Thomas Becker
Fakultät für Mathematik und Informatik
Universität Passau
94030 Passau
Germany
becker@alice.fmi.uni-passau.de