The LaTeX-pack

Or LaTeX-editing made easy

by Jörg Fischer
Version 0.5, August 13, 2002
http://nedit.gmxhome.de

Contents

Description

The LaTeX-pack is a package or set of macros, i.e., small programs, written in the NEdit macro or scripting language. It will turn NEdit, the Nirvana text editor, into an advanced LaTeX editor. NEdit version 5.2 is required, version 5.3 is recommended. NEdit is a Unix program for the X Window System under GPL. It is free to use and open source. You can download the latest version from NEdit.org.

Notice, that this manual may partly be not up to date – please, be patient. For a short note on what's new, go to 6. Notice that some macros require NEdit to be in server mode (you get this by starting NEdit with `nedit -server` or `nc -noask` – to drop the `-noask` set `nc.autoStart: True` in your X resource file), see 4.10 and 4.16.

Notice also that there is a set of *example files* to get you started quickly. It illustrates the basic features of the LaTeX-pack, containing editing mathematical equations and multi-file documents (references, sectioning, bookmarks). *You should have a look at it*!

Notational Conventions

I will try to keep to the following notations, in order to ease reading:

- Keys are written in small capital letters, e.g., SHIFT+SPACE means to hit the SPACE key while holding the SHIFT key.
- Names of files and directories are enclosed in '.', e.g., '.nedit'.
- Names of menu entries are enclosed in ".",e.g., "Run>Preview".
- Commands, variables and generally source code are written in typeface, e.g., `nedit -import .nedit`.
- Emphasized words are *italic* and very important points are red.

1. Introduction

At first, I didn't want to write a manual at all. I don't like having to read long manuals to learn how to use something. The AUC-TEX manual starts like this:

> Although AUC-TEX contains a large number of features, there are no reasons to despair. You can continue to write TEX and LATEX documents the way you are used to, and only start using the multiple features in small steps. AUC-TEX is not monolithic, each feature described in this manual ...

In my opinion the reason to despair is not that there are many features — there are many features in LATEX itself. Also, everyone knows that he can write his LATEX-files as before, so we do not need AUC-TEX, do we?

But seriously, in my view the problem, that the above manual tries to disguise, is to figure out, how AUC-TEX works and I assume that this takes at least a lot of time. And then, does AUC-TEX make editing really quick and easy?

I am convinced that useful things have to be easy to use. That's the way this macro package is written. You can start using it without having to learn some awkward handling. Just write your text, then select it with the mouse and invoke a macro to let it do the rest.

Suppose, for example, that you want to give in a small 2×2-matrix as text-formula with round brackets and the entries a and b in the first row and c and d in the second row. It looks like this:

This is a small matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$.

You do it in the following way: Directly after "This is a small matrix" you type once[1] $. Then you type A SPACE B RETURN C SPACE D. Then you select from a to d with the mouse and click the right mouse button. In the so called Window Background menu that shows up, you choose Matrices. A dialog menu pops up, where you see the different types of brackets and below there is a row of buttons labeled "OK", "small", "ltx", "small-ltx" and "Cancel". Select round brackets and press the button labeled "small".

[1] If "smart indent" is on.

In the same way (almost) all the actions are handled, i.e., for equations, lists, environments, format, snippets and so on. That's all you need to know to work with these macros!

I think learning by doing is fastest. That's why there is a small example file named 'example.tex', where you can try a few things out for yourself in order to get you started.

Although you don't need to read a manual to learn how to work with the LaTeX-pack macros, there are several other reasons to have a manual. One reason is that things may not work and the manual can help to figure out what goes wrong. So make sure to *read* in the manual, if a macro shouldn't work.

Notice further that the example above needs the $\mathcal{AMS}$-LaTeX package. Standardly the macros fill in the $\mathcal{AMS}$-LaTeX commands for matrices and equations. That's because I am working with $\mathcal{AMS}$-LaTeX and this is just another reason to have this manual – the macros are mostly suited to my needs and probably these will differ from your needs. So you may wonder, why these macros are doing things this way and not that way, and I feel there has to be at least some explanation about it and, above all, the remark that the macros can easily be changed to suit your needs. Of course in that case you have to

learn a bit about NEdit, about its macro language and about regular expressions (see 4.7.1, for example).

2. Installation

I assume that you already have NEdit. If not and you have some Unix version available, go to NEdit.org and download version 5.2 or later. NEdit consists only of a single binary, so there is no need of a special installation. Only unpack and execute it.

You are running only MS Windows? Well, although I would prefer Unix, you can run NEdit on MS Windows 95, 98, ME, NT, 2000 and probably XP, together with MiKTEX. (So do I, sometimes.) I recommend, that you use the easy to install package from my home page.

Although NEdit is a Unix program, it is fully working under Windows. I mention this, because I've heard several times from folks who like Emacs, that they don't use the Emacs' ports to Windows, because they don't work properly. I can't comment on this further, because I don't use Emacs, neither on Unix nor on Windows...

There are slight differences in the installation of the LATEX-PACK between Unix and Windows. Notice that for the Windows part, I assume that you are using the package from my home page.

2.1. Unix/Linux

Copy the files 'dot_nedit' and 'dot_neditmacro' as '.nedit' and '.neditmacro' to your home directory, so that NEdit can find them.

Also, copy the folder 'nedata' to your home directory or set the global variable `$data` in the file '.neditmacro' appropriately. In principle, that's already all that you need to do to install it.

If there are already '.nedit' and '.neditmacro' files of your own, you could temporarily rename these in order to see how the normal setup of the LaTeX-Pack looks like. Since you probably want to keep your NEdit preferences, you can do the following thereafter: Append the contents of 'dot_neditmacro' to your '.neditmacro' file (if there should be functions or global variables with the same name you have to rename them.) Then import the 'dot_nedit' file with the command `nedit -import dot_nedit`. The menu definitions will be placed on top of your definitions. You have to save defaults once now, since this is not done automatically (i.e., the imports would be lost if you restart NEdit). Notice that the accelerator key definitions are *not* stored in 'dot_nedit' in order to avoid interferences with key definitions in your own preference file.

X defaults: In addition some X defaults have to be set. In order to

do this, copy the contents of 'dot_Xdefaults' to your '.Xdefaults' or '.Xresources' file. You have to execute for example

```
xrdb␣-all␣.Xresources
```

or to log in again for the new settings to take effect.

2.2. Windows (Cygwin Port)

Open both the files 'dot_nedit' and 'dot_neditmacro' in NEdit! The file 'dot_neditmacro' needs to be edited as described there. Then save the file 'dot_nedit' as *'nedit.ini'*[2] to the NEdit home directory (confirm the installation of the Cygwin port), thereby, and this is essential, choosing Unix format! Do the same procedure to 'dot_neditmacro', i.e., save it as *'neditmacro.nm'* in Unix format. If you don't choose Unix format, it is likely that the files are interpreted as DOS format, so that NEdit can't read them when restarting. This will lead to a whole lot of error messages and a program stop. If you already should have 'nedit.ini' and 'neditmacro.nm' files, then what is described under Unix applies here, too.

[2] If you use the binary from me!

Also, don't forget to copy the folder 'nedata' to the NEdit home directory.

Notice that there is a small shell script 'yap-nc.sh'. This must be copied to the 'cygwin\bin' folder, in order to use the 'Run > LaTeX'-macro (4.10). Also 'yap' has to be set up appropriately, i.e., you find the command line to invoke the editor in the file 'yap-com.txt'.

X defaults: Under Windows the X defaults are set in the file '.nedit', because not all X servers will read the X default file. This is, by the way, the reason for renaming the preferences files, so that the Xdefaults won't be overwritten when preferences are changed. So save the contents of 'dot_Xdefaults' as '.nedit' to the NEdit home directory.

2.2.1. Remark on X Defaults

If you are a pure MS Windows user, you probably wonder, what this X defaults thing is about. NEdit is an application for the X Window system, the standard graphics system under Unix, which is based on the client/server model and is indeed independent of operating systems. There are many X servers for Windows. In order to use NEdit you'll need some X server.

One of the advantages[3] of the the X Window system is, that you can have complete control over an application, for example what keyboard handling is concerned. Have a look at the 'dot_Xdefaults' file. It would be new to me, if one has the same control over keys in MS Word. For a useful application see 4.7.

[3] There are also some disadvantages.

3. Overview

After installation you will find the macros located at the Macro and Window Background menu (which shows up when clicking the right mouse button). You can customize them by going to "Preferences > Default Settings > Customize Menus" and then select the type of menu. *Notice*, that almost all macros are made language dependent, that means you will see them only when editing LaTeX-files.

With the macros contained in the LaTeX-pack you can

- define named snippets, for inserting in your text,
- comment and uncomment parts of your file,
- define abbreviations for longer words or commands, that you need to write often,
- let partly written words or commands be completed,
- insert matrices and equations in the easy way described in the introduction,

- display lists of, e.g., the labels or the user-defined commands in, and the sectioning of, your file,
- insert all kinds of environments, formattings, sections, templates, mathematical symbols, ...,
- put mathematics automatically at separate lines and let closing brackets automatically be inserted.

In addition, since you will probably need to adapt the macros to your needs and taste, there are some macros for use with the NEdit macro language included. You can see them located at the Window Background menu, showing up when right clicking the mouse, when editing a NEdit macro file. These macros give you information about the built-in macro function and variables you can use.

Notice in this context that some of the LaTeX-pack-macros are of general use. So expanding abbreviations is certainly not only good for LaTeX-editing. Indeed, the same macro is used to fill in constructs like while- and for-loops of the NEdit-macro language while editing '*.nm' files.

Moreover, a Tcl/Tk script called *HelpSystem* together with an on-

line help about LaTeX was included, since this could (eventually[4]) be more than only a browsable on-line help.

Independent from this, editing your files with NEdit will reduce the amount of your errors dramatically, because there is a capable syntax highlighting[5] available. Notice, that there is an improved highlighting pattern included in the package.

[4] It isn't so far.

[5] Not only keyword coloring, please!

4. Description of the Macros

The LaTeX-pack-macros need various data-files that are put in a separate folder named 'nedata' in the home directory. If you want to change this name, open the file '.neditmacro' in your home directory and set the global variable `$data` appropriately.

Often you will not need to know much about the paticular macro in order to use it. But notice that all of the macros are text files. You can read them. You can read all of the files in the 'nedata' folder, except for the few Gif-files, of course.

Notice that I do not consider the set of macros to be complete or perfect. They are perhaps not even good.[6] Although the whole auc-TeX-distribution, including ref-TeX and the BibTeX-mode, could be emulated with NEdit, this isn't the intention of the LaTeX-pack. Especially the LaTeX-pack contains no macros for formatting your documents (because *I* think this is useless) and there is no automated re-parsing of your documents or the attempt to implement automated things in general (because in my experience you can't get a program

[6] Although there are people around that try to make money out of something lesser.

to do automatically what you want – it will mostly do automatically what you don't want).

Notice at this point that I did not program these macros for you. These macros are mostly as I use them, so they have a strong tendency toward mathematics, or better that kind of mathematics that I need to write. There are other things added only to show how to get them. But these things need probably improvements. So if you are missing something or if you think some macros or even all are not good enough, do not hesitate. Add your own macros or improve the existing ones. There is no problem. You can freely distribute it. I have put the whole package under GPL, but the reason for that is only that I think that changing and redistributing shouldn't be done in a totally disordered way.

The intention of releasing such a set of macros is to give you a *starting point*, to give you examples of how to do it. You needn't be afraid. The reason for choosing NEdit is that it is unbeaten for its combination of functionality and ease-of-use. It's macro language has a learning curve of about half an hour (or perhaps one hour, if you have never written programs yourself, as myself until I started with this here).

4.1. Snippets

This macro lets you maintain a collection of named snippets. There are several ways to insert often needed commands or constructs or just parts of text that you need to write from time to time.

For regularly used commands that you don't want to write again and again, the automatic completion macro (expander) or the word-/code completion macro are probably more adequate and faster. The snippets macro is more for (longer) parts of text that you only need to write time after time, so that you are more likely to forget about them. That's why you can give names to the snippets (instead of just a short abbreviation) and the beginning and ending parts of the snippets are shown together with the names in the list dialog menu from which you choose them.

4.2. Theorems

Write your theorem, proposition or the like, slect the text thereafter and invoke the macro by clicking the right mouse button and choosing "Theorems" from the Window Background menu. Of course, what you need to get filled in depends on your `\newtheorem`-definitions. So

you need the 'theorem.dat' data file. Or you could rewrite the macro to look for the definitions in the file you are editing. (You could use the "label"-macro for this, see 4.9.)

4.3. Equations

Only the $\mathcal{AMS}$-LaTeX versions of equations are supported, because they are clearly better than the (pure) LaTeX ones. The best way to see how things work is to try the examples in the 'example.tex' file.

Here is a only short description. I've tried to make as much automation as possible. Normally you fill in only the entries of the equation even without alignment mark (&). If your single equation doesn't fit on one line (in the TeX-output) you insert a newline (RETURN key) at the place where the equation should be split and write the next line and so on. Thereafter you select the whole equation and invoke, as usual, the "Equations" macro from the Window Background menu. Select single equation and choose "OK" or "No-number". Then the macro will look first whether there is more than one line of input, otherwise there isn't much to do. If there is more than one line, the macro knows that the equation must be split and needs alignment.

The alignment is done in the following way: Each line processed separately. First the macro looks for a semicolon. If found the first semicolon is changed to a &. If no semicolon is found, the macro checks for a binary relation and inserts a alignment mark (&) in front of the first binary relation that is found in the line. If neither a semicolon nor a binary relation (you probably have to add some more binary relations to the search string) is found, then a & is inserted at the start of the line if it is not the first line. If nothing is selected, a template is inserted at cursor position.

4.4. Matrices

Just give in the entries of your matrix, separate columns with a blank space, if this is not possible, define a different separator, e.g., a semicolon.

Tables are supported only with a template under Insert misc-templ. This is because I don't need them usually. If you do you can take the matrices macro as an example how to program a macro inserting tabulars. If nothing is selected, a template is inserted at cursor position.

4.5. Lists

This is for quick inserting of various lists such as enumerations or descriptions. You needn't type in `\item`, that is done automatically. Only hit the short-cut SHIFT+RETURN for a newline.

4.6. Format and Sections

Various font sizes and also the verbatim environments, footnotes and quotation marks are inserted. This is mainly in case you forgot the appropriate commands and otherwise would need to look them up somewhere.

4.7. Expander Macros

The Expander macros are located at the macro menu. At the entry "Expander" you can see the macros "Init", "On", "Off" and "Edit dat". Below these is a separator and the macros "lists on" and "lists off".

In order to check if they work initialize[7] them with ALT+U or click on "Init". Then give in `'e` and hit space. You should get an `\epsilon` now. To see all the already defined abbreviations click on "Edit dat" or hit CTRL+SHIFT+E. You can add your own abbreviations or edit the given ones. If you open the file where the abbreviations are stored with the *macro* (CTRL+SHIFT+E), edit and save it, the newly defined abbreviations will be present without re-initialization. Otherwise, you have to re-initialize.

The macros that you see there only initialize the Expander macro, turn it on or off and edit the data file, where the abbreviations for expansion are stored. The entries "lists on" and "lists off" turn the *lists completion macro* on or off. This is a small macro that automatically inserts `\item` or `\bibitem`, when you are in a list environment such as *enumeration* or *thebibliography*. It is not related to the automatic completion macro and can be turned on or off separately. There is nothing more to say about it.

What you can't see in the menu are the macros that are doing the work, i.e., you don't see neither the automatic completion macro nor

[7] If you would like to initialize the expander macro automatically, set the global variable `$update` at the start of the '.neditmacro' file to one.

the list macro. This is because you needn't execute them, so I've hidden them.[8]

Execution of the completion macro is bound to the SPACE key and the list macro is bound to SHIFT+RETURN. That means, every time you hit SPACE or SHIFT+RETURN, not a blank or newline is inserted in your document, but one of these macros is run.[9] The binding of SPACE is done by way of key translations[10], whereas SHIFT+RETURN is bound as accelerator key. See 7.1 for the technical details about this. If you believe it is dangerous to bind the completion macro to SPACE you can and should of course change the key bindings.

The automatic completion macro is generally usable to define abbreviations that the editor completes. There are several variations of such macros around, one of them is the so-called *expander*. The

[8]This is simple. Set the menu entries for the macros to an undefined language mode (I chose "@keys", because the key-bindings have been re-defined for it), so you can never see them in the menu.

[9]Don't worry. These macros will not forget to insert a blank or newline in your document.

[10] For this reason you must add the contents of the 'dot_Xdefaults' file to your '.Xdefaults' file.

expander is described in detail at the NEdit home page.[11] I've included some of its features in this version. To see the abbreviations that will be completed, click on "Edit dat". This will open the file 'expand_$language_mode.dat' in the 'nedata' folder. You can simply edit this file, change abbreviations or add your own ones. Then save the file and initialize. Note that to run the macro you must first execute "Init" from the menu or define a short cut for it. Then the file is load to a string in memory, so that the file need not be opened every time you hit the space key.

The expander's features included are that you can recursively[12] expand abbreviations and that you can expand an abbreviation with a selection. Recursive expansion means the following: If you put an already defined abbreviation enclosed between |>...<| in the definition of another abbreviation, then this field will be recursively expanded. Moreover you can set the cursor at any place in the expansion by putting an empty field |><| at the desired position. Also, you can give in fields that aren't defined as abbreviations. After expanding,

[11] Notice, that parts of the expander are external C programs.

[12] See the files 'templates_lat.dat' and 'persoenlich.dat' for one possible application of recursive expansion.

the first such field will be selected, so that you can type in your text for the field. Afterward you can jump to the next such field with "Next field" or define a short cut such as CTRL+X, for example. Expansion with a selection means that you have the following situation: `abbrev␣word`, where `abbrev` is the abbreviation to expand and `word` is some word. The cursor is directly after word. When you hit CTRL+SPACE now, then the word is taken as selection and the abbreviation is expanded. For example, you can define the abbreviation `bg` as

```
\begin{|>s<|}
|><|
\end{|>s<|}
```

Then `bg verbatim` will be expanded – well, I think you understand it. The `|>s<|` stands for the selection. But what when you would like to have more than one word as selection? No problem, select all the words you want behind the abbreviation and press CTRL+SPACE.

Notice that there is a small problem, because the expansion macro is bound to the SPACE key. So hitting space directly after `bg` in the above example will expand it before you can type in the rest. One

solution would be to have a different key binding and you can change this to your taste. My solution is that with SHIFT+SPACE there will be inserted a blank without the call to the expansion macro.

This solution is also useful, because the automatic completion macros capitalize automatically the first word of a sentence, if you forgot to do so. This will cause no pain for TEX-editing. If there is an abbreviation, e.g., e.g., you normally adjust spacing with ~ or \␣, because TEX interprets a dot as end of sentence. Or otherwise, again, you can type SHIFT+SPACE after the abbreviation.

4.7.1. Remark

A remark on the correction macro above. This is done with the help of *regular expressions*. What is this? I think this is best understood by that easy example. Suppose you want to correct the first word of a sentence, if you forgot to capitalize it. You could of course easily search for a dot ending a sentence and also for a dot followed by a space, but what then. There are quite a few letters in the alphabet. Of course you could search for `.␣a` and then for `.␣b` and so on. But what you are really searching for is a dot followed by a space and then a letter. This is indeed a simple example of a search pattern. And such

search patterns are described with the help of regular expressions. The regular expression describing this search pattern is `\.␣\l`, where the dot is described by `\.` the space is described by itself and `\l` stands for some single letter. Now have a look at the source of the above macro!

4.8. Word-/Code Completion

In order to help you with your writing, there are the word- or code completion macros. You write the start of your word or of a command and invoke the macro with F4. The macro looks through your current file *and through external completion files, that you can define*, and completes the word. A list of English and German top words and a LaTeX-version of the German top words is already included. There is also an example of a user-defined word completion file. Moreover, there is a code completion file for LaTeX-commands included.

Notice that the decision whether a word- or a code completion is attempted is simple – if your word starts with a backslash, a code-completion is done, otherwise it is a word completion.

Notice that for a completion file, a whole line is taken as comple-

tion, so that you can give in several words, where you need only to write the start of the first word in your file and press F4 to get the whole completion. If there are several possible completions, you can cycle through them by pressing repeatedly F4 or you press F5 which will show a list of all possible completions, where you can choose the wished one. If you have set a main file, user-defined commands will also be completed.

4.9. Labels, Commands and Sectioning

A common problem with larger files is for example that you need many text references, which are created with the command `\label` (or, if we think of pdf, with `\hypertarget`). Other examples are that you have many user-defined macros or environments in your text (`\newcommand` and `\newenvironment`, respectively).

Even if the file is not really large, you are likely to forget what labels you have already created and where they are. Of course, you can search them manually or let the TEX-compiler tell you that there are double labels and then you must search and change the double ones later on.

Better is, we let your computer work. So these macros will search for labels and display an alphabetically ordered list of the names of your labels (hyper-targets, sections, user defined commands or whatever you like).

Often it is more comfortable to split large files into several smaller ones for single chapters for example. Then one creates a main file for the common declarations (user defined styles, commands and environments) and includes the files containing the actual text into the main file with the commands `\include` or `\input`. So it is necessary that the macros search recursively through all your include files. Notice that the included file names are expected to be kept in the same folder as the main file. Notice that these macros might need further customization.

4.10. Running TEX and Previewers

Notice that the "Run > LATEX" macro assumes, that you are running NEdit through 'nc', the NEdit client, i.e., NEdit must be in server mode. Otherwise opening an output window doesn't work in the way the macro does it (with a shell command invoking nc). So you could

change that, if you don't want to run NEdit in server mode.

By the way, I've written these macros mainly for using under Windows. Under Unix I work with a shell and set the environment variable TEXEDIT to `nc -noask -line %d %s`.

If you have defined a main file, see 4.12, invoking "Run > LaTeX" will compile your main file, otherwise the current file is compiled. The same holds for the preview macro.

Notice that these macros run with teTeX as well as with MiKTeX. If you are running Windows, it is even assumed, that you use MiKTeX! The previewer is defined as 'yap' and you can use source specials, i.e., clicking in the 'yap' window brings NEdit to the appropriate source line, confirm the MiKTeXmanual. *Make sure* to set up 'yap' appropriately, see 2.2.

If you are missing something, such as running BibTeX, makeindex or a "build" macro, the provided macros have enough information how to do this yourself.

4.11. Source Specials

By *Source Specials* is meant, that line numbers and the names of your source files are included in the Dvi output. This information allows to relate pieces of output, i.e. paragraphs or formulas, with the place in your source file that created the output. Thus by clicking in the previewer window, you can jump to the appropriate place in your source file. This is called *reverse search*. Moreover, it is also possible to call the previewer from inside your editor to show the output related to the current cursor position. This is called *forward search*. In order to make use of reverse and forward search, you need a Dvi file with source specials and a previewer that supports these. Moreover, for forward search you need at least an editor that can call external applications.

If you are running Windows and using MiKTEX, this is not much of a problem. MiKTEX's compiler supports the direct production of Dvi files with source specials by way of a compiler option and 'yap' also supports source specials, both from version 1.2 on. So, you need essentially only to tell 'yap' what editor to call.

If you are running Unix it is possible that you don't have a recent beta version of the teTEXdistribution. You can check with `tex -help`

whether a option for direct creation of source specials is supported (something like `-src`). If not, the easiest way is to get the 'poor man's solution', that is the LaTeX package 'srcltx', that if included with `\usepackage` in your document creates a Dvi file with source specials. Moreover, it is possible that the 'xdvi' version that you have doesn't support source specials (you need at least version 22.38), so that you have to get the latest 'xdvi' version. But then you can use source specials, too.

4.12. Main File and Bookmarks

This small macro let you define a main file. So if you are working on larger projects, the labels, commands, sectioning, running TeX and preview macros are more comfortable to use. The main file is defined through the dialog. In the dialog there is a list of file names displayed that either are currently open or have been bookmarked.

Especially, when working with multi-file documents or projects, it will be comfortable to bookmark some of the files that you regularly need. This is independent from the 'Open previous'-list inside NEdit that is constantly changing. It also allows to easily set the main file,

see above. You can bookmark the current file simply with Ctrl+B. You can see a list of the bookmarked files to open a bookmarked file or to delete an entry from the list with Shift+Ctrl+B.

4.13. Comments

These macros are for quoting or unquoting parts of your file. It is just a slight variation of default macros included with NEdit. The 'docstrip' macro will delete all comments in a selection or in the whole document.

The "ues2tex" macro has nothing to do with that. It is a macro that changes the German umlaute to TEX-style and vice versa. This can serve as an example for other things, too. I just did not know where else to put it.

4.14. Insert

These macros simply insert environments, miscellaneous templates of LATEX-constructs, symbols, mathematical symbols and BibTEX entries in your text. If you miss more templates you can simply add them to the data files.

Special note for the mathematical symbols. There are some editors that try to impress you with a kind of graphical user interface, i.e., they show pictures of math symbols that you can click on and the appropriate command is inserted in your text. Indeed, this is a rather simple thing, as was shown by a small Tcl/Tk-script in a former release of the LaTeX-pack. Now, the help system, see 4.16, is used for such things. The advantage compared to a program such as LaTeX-helper is that you only need Tcl/Tk and thus it can run everywhere (without Gnome libraries for example).

4.15. Dollars, Brackets and the like

In order to ease editing TeX-files, NEdit's standard key bindings are changed by the LaTeX-pack, for technical details about it see 7.1.

With this changed key bindings, the following will happen when editing a TeX-file:

- Maths environments starting with $ are automatically placed at new lines and an additional $ is inserted. The cursor will be placed between the dollar signs.

- Maths environments starting with `\[` or `\(` are automatically placed at new lines. The appropriate closing bracket is inserted and the cursor is moved in between. You give in your equation that then is placed at a new line, i.e., you needn't type RETURN, this is done automatically.

- Generally typing in an open bracket produces the appropriate closing bracket and the cursor is placed between the two brackets.

- Typing in two underscores in a row will input sub and super indices.

- Hitting ALT+" inserts US or German quotation marks (depending on language setting).

Notice that, *if "Smart Indent" is turned on*, you can move through brackets or the dollar sign by just typing in spaces, i.e., typing in two spaces in front of a closing bracket or the dollar sign will move the cursor and the last space behind the bracket or the dollar sign. You should type in a punctuation mark, that should come after a bracket

or dollar, inside the brackets or the dollars – moving through will take it behind automatically.

4.16. Help/Assistant

There is an on-line help for LaTeX included. This is based on a Tcl/Tk-script of a Russian author, which is under GPL. For the details invoke this help system from "Macro > Help > About Help". You will need Tcl/Tk installed on your system. Of course, it works on MS Windows, too. Notice that you may change the command 'wish' in this case.

Another possibility to invoke this help system is from "Macro > Insert > Greek letters". This displays all Greek letters in the help system and you can simply click on them in order to insert them in your document. This insertion is done by an interaction of the Tcl/Tk-script and the NEdit-macro language on the other side. Read the file 'latex.tcl' for details. *Notice that "Smart Indent" has to be turned on and you have to run NEdit in server mode for this to work*!

One of the advantages of the above approach is that everything is free software and open source. You can read and write all the files. The help system, i.e., the '*.help' files are HTML-files with some

additional directives. The pictures of the Greek letters are Gif-files in base64 encoding. You can do such things simply yourself.

4.17. Spell-Checker Handling

The problem when trying to spell-check your LaTeX-document is, of course, that there are many commands in. Some spell-checkers, like Ispell, have a LaTeX-mode to avoid the worst, but this says it all. There are also commercial or proprietary software environments, that have a spell-checking feature built-in. However, we are using free software and create our LaTeX-document with a pure text editor.

The natural approach to me seems to simply filter all of mathematics and other commands from your document *before* sending it to the spell-checker, so that the spell-checker only needs to check the real words as it should be. So, the main part of the spell-checker handling macro is indeed a set of patterns (regular expressions) that are supposed to strip off (most) of the LaTeX-commands from your file.

Notice that you need either Ispell or Aspell installed on your system to run this macro! Notice also that there is no claim that it is much better than pure Ispell or Aspell in LaTeX-mode – it is only an attempt.

5. General Remarks

These macros are only for use with NEdit. So you could say, that you and many others are using different editors and therefore it is bad to have something intended to help with editing LaTeX-files, which can only be used with a special editor. I'm not going to discuss about what is the best editor. Also, I do not try to sell anything nor to convince anybody of using NEdit for text editing.

My view is as follows: I had to choose an appropriate tool for the purpose of making editing LaTeX-files easy. I chose this editor, because that purpose can be achieved with it.[13] Perhaps there are other editors around achieving that purpose, too. But this doesn't make my choice wrong, because NEdit makes editing LaTeX-files easy. If you stick to your editor, because it helps with editing LaTeX-files, too, then just make your contributions.

Many of these macros are not perfect in any way. First of all, I am not a programmer. If I was, I would not have released this, because I would not want to lose my job. Then, I have not written these macros for others. Actually, I have written them for me. This implies that

[13] And NEdit is GPL, of course.

these macros are most suited to my needs, e.g., have a strong tendency toward TEX'ing mathematics, whereas other things may be missing in part or even completely. So, in order to get the most out of it, I strongly recommend that you read not only in this manual from time to time, but also start to read all the files in this distribution. There are only plain text files that you can read and see, what they do. For making this easier there is a *documented* source file with name 'latex_pack.nm' included. Of course, you will need to learn about NEdit's macro language. But this is neither difficult nor an awkward thing to do. Indeed, one of the reasons for choosing NEdit lies in this well-chosen and straightforward programming language. Another reason is NEdit's advanced regular expression facility together with a real, and nevertheless fast, syntax highlighting. If they say, NEdit would be lightweight, as I hear sometimes, they must mean its size, not its features – or they don't know what they are talking about.

It is my conviction, that the LATEX-PACK will serve as an example, how to make editing easier. Similar sets of macros could be written, or are already there but not contributed, for editing any kind of plain text files, for example XML, HTML, C, Java, Perl, Tcl/Tk ....

6. Changes – what's new

In version 0.5 there are several improvements. The 'labels'-routine has been simplified and 'sectioning' has been included. The Word-/Code-completion macro has been corrected. The HelpSystem has been updated to version 1.4 and more examples have been included. There is also a corrected version of the LaTeX-highlighting patterns for NEdit. Moreover, the multi-file handling has been improved, see 4.12. There are also some smaller things, e.g., hitting the underscore-key two times in a row will insert sub- and super indices, hitting Alt+' inserts German quotation marks (can be changed to US ones). Also, this documentation was updated.

Notice that some key bindings have changed and that there are changes in the expander macro, see 4.7.

In version 0.3 there are only minor changes to version 0.2. The spell-checker handling macro is slightly corrected. The assistant system can give now a first vague impression of what it should become. Have a look at "Macro > Insert > Greek letters". There is a technical section in this manual to explain how this is done, see 4.16. Moreover,

there is an illustration of the use of the recursive expansion feature of the expander macro, see 4.7.

In version 0.2 there are some more macros of general use included. For example the word completion macro (that can be used for code completions, too) and an alpha version of a macro that handles an external spell-checker like Aspell or Ispell (which you must have installed on your system). Moreover, the smart indent macros (no good word for what they are) have been changed almost completely. I have changed the key-bindings, so that to every opening bracket there will be the appropriate closing bracket inserted. If you don't like it, you can hold the ALT-key down, or delete the key-bindings (in the X defaults file).

In addition, there is a macro included, that saves the current cursor position in a file, when you close it, so that you can go on editing at the same position next time. This is also done by changing key-bindings, i.e., the macro is executed when hitting CRTL-Q or CTRL-W, before the file is closed.

Finally I decided to include an alpha version of a kind of assistant system (or it could become something like this), although it is far

from completed. The reason is that I don't have much time (actually not even enough time to update this manual), so I couldn't bring it to the state that I would like it to be, but it should be clear from the provided macros, how this can be done.

Make sure to check out 4.7. This macro has been renamed and the data files that it uses have been renamed, too. It is possible now, to include several data files for abbrevations and corrections by simply adding their names to the arrays at the start of the initialization, see the "Expander > Init" macro.

Notice, that short-cut keys have been changed, too. They are listed in the file 'key_bindings.txt'. Confirm also the X defaults settings for NEdit. Finally this manual isn't up-to-date, thus you may need to read the macros themselves.

In version 0.1, the LaTeX-pack runs with MiKTeX. For Windows 95, 98 and ME you should use the binaries from my home page. New macros are for running LaTeX, previewers, showing help files and inserting BibTeX entries. The automatic completion macros have been improved and the 'expansion with selection'-feature of the so-called expander was added. There were also a few slight improvements of

other macros. In order to ease installation, all (proposed) accelerator keys were shifted to 'dot_Xdefaults', so you have to define key bindings for yourself. I did this in order to avoid interferences with accelerator keys that you may have already defined when importing the NEdit-preference file.

7. Technical Notes about NEdit.

These sections cover technical remarks about key bindings in NEdit and some recent developments going on.

7.1. Key Bindings

There are two different ways to define key bindings in NEdit. One is by way of key translations in your X resource file, the other is by defining accelerator keys in the macro, background and shell menus. Both ways of defining key bindings are quite different. The key translations cannot be changed on the fly, i.e., in order to change them you have to edit your X resource file, to run `xrdb` and to restart NEdit. So, if in the X resource file keys like SPACE, RETURN or the bracket keys are bound to macros (as is done in the LaTeX-PACK) and these macros shouldn't be there, you have a problem. On the other hand, the definitions of accelerator keys in the menus can be changed on the fly and they overrule the key translations. Moreover, these accelerator bindings are only active, when the respective menu is active. This means that you can bind the same accelerator to a lot of macros, if these macros are for different language modes. Thus the decision seems to be clear:

Forget about key translations and use the accelerator bindings.

However, why are these bindings called *accelerator* bindings? Can they not be used to bind SPACE, RETURN or the bracket keys? The answer is: Yes, they can, *but* you should not do so, because these bindings are indeed only intended for accelerator keys. The problem is that their definition is global to the window. You can see this, if you bind for example the SPACE key to a macro. Then invoke the incremental search bar (CTRL+I) and try to give in a blank. This won't work, because the macro bound to SPACE will be executed. So, binding normal keys to macros will make the incremental search bar useless (if you bind RETURN to a macro, you can't even search incrementally).

This is the reason to use both type of key bindings. For all the bindings of normal keys such as SPACE and the bracket keys to macros, that are done by way of key translations, there is also a definition included in the X resource file to get their original behavior. Simply hold additionally the ALT key down. Only for the SPACE key you have to hold a SHIFT key down.

But, of course, you *can and should* change the key bindings to your taste and needs! Notice that I even didn't include accelerator keys for

most macros. So, if you don't want to invoke the macros through the menu each time, you have to define accelerator keys for them, which is simply done in the macro and window background commands menus ("Preferences > Default Settings > Customize Menus").

7.2. Recent Developments

The latest stable release of NEdit is version 5.3. It is already worthwhile to update to this version, because the on-line help has become much more readable. There also have been a couple of bug fixes, one causing a crash when a syntax highlighting pattern contains characters with an ASCII code greater than 128.

More exciting is the current development going on. In the latest sources there are the so-called 'call-tips' included. These are little display boxes to show short help messages or compiler errors. There is also a big efficiency improvement in the regular expressions algorithm. In addition another parenthetical construct (look-behind) was included. This makes it possible to further improve the LaTeX-patterns. Moreover, a new environment variable NEDIT_HOME was introduced to simplify the exchange of NEdit's preference files (in

which the menu definitions, macros and syntax highlighting patterns are contained).

Finally, there are two patches available for version 5.3, one allowing optionally tabbed windows and the other allowing to color the background of ranges in the NEdit window. This can be used to mark mathematical formulas or other special environments. It can also be used to mark lines where TEX sees a problem.