LS-Emacs

Language Sensitive Emacs Version 1.2
Second, revised Edition, January 1998

by Christian Tanzer

Published by
Christian Tanzer
Glasauergasse 32
A–1130 Vienna, Austria, Europe
Phone +43 1 876 62 36
Fax +43 1 877 66 92
E-Mail tanzer@swing.co.at

1 Introduction to LS-Emacs

LS-Emacs (*Language Sensitive Emacs*) is an extension to the popular GNU Emacs editor. LS-Emacs provides language specific templates and on-line help on language features. This document describes how to use LS-Emacs for editing, how to design language specific templates, and how to use LS-Emacs features inside elisp programs. It does not describe general Emacs concepts — please read the Emacs documentation to learn standard Emacs.

Originally, LS-Emacs was loosely based on ideas embodied in Digital's Language Sensitive Editor (DEC LSE). LS-Emacs is based on the idea of language specific templates, which supply the syntactical structure of a document or program[1] of some language. These templates are defined declaratively, i.e., no programming is necessary to define a new language or to change an existing one.

LS-Emacs templates provide a fast and efficient way of editing. Besides, they offer the following advantages:

- **Ease of use**

  LS-Emacs simplifies the use of complex languages like TEX or SGML. Comprehensive templates allow you to use such languages with little knowledge.

- **Cognitive support**

  You don't have to remember each and every syntactic detail. LS-Emacs knows the details and supplies them automatically. In addition, LS-Emacs supplies context sensitive help whenever asked for.

- **Document consistency**

  The use of standardized templates results in structural and stylistic consistency of documents written by different authors for different projects.

  Documentation standards can thus be guaranteed without causing bureaucratic nightmares for those writing programs or documentation.

- **Reduced typing effort**

  You are spared the tedious typing of markup, keywords, etc.

- **Error prevention**

  LS-Emacs reduces the probability of syntax errors and omissions.

Yet, LS-Emacs does not force you into using the language specific templates. If you know what you're doing you are free to edit your document in whatever way you want. Thus, you have the choice where to rely on language specific templates and where to use free typing. With practice, you will quickly adapt a personal style according to your preferences which results in maximum productivity for you.

[1] Henceforth, I will use the term document to refer to any kind of file you may edit, be it a love letter or a computer program.

The language specific functionality of LS-Emacs is provided by elisp-libraries and complements the features of standard GNU Emacs. The language specific functionality for a specific language is hooked into the appropriate major mode of GNU Emacs.

Note to DEC LSE Users

Digital's LSE offers a number of features beyond language templates, for instance integration with language compilers, integration with DEC Source Code Analyzer (SCA) and DEC Code Management System (CMS), and support for pseudocode. LS-Emacs does not directly deal with such issues.

Nevertheless, standard Emacs supports many of these features in a similar way:

- Integration with language compilers including diagnostic review (see GNU Emacs Manual, chapter "Compiling and Testing Programs").
- Tag tables provide similar functionality as DEC SCA: not quite as powerful, but much more efficient (see GNU Emacs Manual, chapter "Editing Programs", section "Tag Tables").
- Emacs provides integration with both important version management systems of the Unix world: `RCS` and `SCCS`.
- Emacs does not support pseudocode, but its outline mode achieves a similar effect. Personally, I never used LSE pseudocode support after some initial experimentation.

In addition, Emacs offers a lot of advantages over DEC LSE:

- Emacs is free. You get the complete source code, you're allowed to use and change it, and you don't have to pay outrageous license fees.
- Emacs is portable. You can get it for many platforms: (any kind of) Unix, VMS, PC's, ...
- The documentation of Emacs is better and much more complete than DEC LSE's documentation. And there is always the source code to look at, if something is not made entirely clear in the documentation.
- Emacs is very easy to program. Without any prior knowledge of Lisp, I was able to implement LS-Emacs in surprisingly short time. In my experience, it takes a lot longer to learn `TPU`[2] than Emacs Lisp (although my backgrounds would seem to favor the `TPU` language rather than Lisp). After two weeks of Elisp programming, my productivity was substantially higher than after 5 years of `TPU` programming.
- Emacs is much more extensible than DEC LSE. The whole design of the editor strongly favors extensibility. It would hardly be possible to implement the features of LS-Emacs with the means of `TPU` as it is delivered to the Digital customer.
- Emacs offers many nice features not found in DEC LSE, for instance:[3]

[2] `TPU` is the editing programming language provided by DEC LSE.

[3] I don't think that the reverse is true.

- multiple undo
- dynamic abbrevs
- incremental search
- arbitrary prefix-keys
- completion of commands, file names, buffer names, ...
- sorting functions
- integration of spell checking
- integration with mail
- powerful file handling commands and lisp functions
- calendar and diary

- I would contend that Emacs contains fewer and less annoying bugs than DEC LSE. For years, my company paid high maintenance fees for DEC LSE, but most of the bugs were never removed.

Regarding support for language templates, LS-Emacs clearly surpasses DEC LSE:

- Features more powerful in LS-Emacs than in DEC LSE:
  - Auto-replication: LS-Emacs can auto-replicate any number of occurrences of a fill-in, LSE auto-substitutes just the next one.
  - Un-expansion: LS-Emacs supports multiple un-expansions, LSE un-expands only the last expansion.
  - Automatic indentation: more flexibility and power.
  - Separation of duplicate fill-ins: much more flexibility.
  - Definition of leading and trailing text to be deleted along with a fill-in: LS-Emacs allows regular expressions, LSE only strings.
  - Integration of fill-ins and tokens: superior in LS-Emacs.
- Features unknown to DEC LSE:
  - Replication of fill-ins.
  - Re-expansion of un-expanded fill-ins.
  - Continuation of fill-in replacement.
  - Recovery of last position.
  - Function-fill-ins and non-replaceable fill-ins.
  - Execution of any lisp function during fill-in or token expansion.
  - Fill-In replacement-leading and -trailing.
  - Fill-In completion actions and killing actions.
  - Template reuse: sharing of templates between languages and sharing of sub-templates between templates.

- Availability of all LS-Emacs functions for the Emacs lisp programmer.
- Automatic loading of languages when needed.

2 Example Sessions with LS-Emacs

This chapter demonstrates how to use LS-Emacs with step-by-step examples. One of the examples shows how you would write a letter using TEX, the other example presents the LS-Emacs language `bash` for the development of a simple shell script. If you are familiar with DEC LSE, you might want to skip this chapter. To avoid too much boredom it might be a good idea to start LS-Emacs and try the examples on-line while reading.

In this manual, I will assume that LS-Emacs runs under the *X window system*. You can also use LS-Emacs on a plain ascii terminal, but due to the lack of an `ALT` key, you have to configure LS-Emacs to use different key bindings.

2.1 Writing a Letter in LATEX with LS-Emacs

When you start editing in a new buffer with extension '`.tex`', LS-Emacs will automatically put you into LATEX -mode. Initially, the buffer will look like this:

```
<<latex>>
```

The position of the point — indicated by ▌ — should be inside the fill-in `<<latex>>`.[1] When you expand the fill-in by pressing `TAB`, LS-Emacs will pop up a menu like this in a second window:

```
#> article                    Latex article document style
   letter                     Latex letter document style
   report                     Latex report document style
```

When you move the point to the second line and press the `RET`-key, LS-Emacs will expand the fill-in into the following text:

[1] For some reason, LS-Emacs sometimes fails to position the point inside the fill-in. If that happens, just use the command `lse-goto-next-fill-in` which is bound to `A-n` in LS-Emacs.

```
\documentstyle<<optionlist>>{letter}
  <<pre_letter_cmds>>···
\begin{document}
  \begin{letter}{<recipient>}
    \opening{<paragraph_text>···}

    <<environments>>···

    \closing{<paragraph_text>···}
    <<cc>>
    <<encl>>
  \end{letter}
\end{document}
```

Assuming that you do not need any option, you kill the fill-in `<<option-list>>` by pressing `A-k`. Next, you will expand the fill-in `<<pre_letter_cmds>>···`. Again, a menu pops up:

```
#> \address
   \location
   \signature
   \telephone
```

This time, you select the first entry, which expands into:

```
\documentstyle{letter}
  \address{<paragraph_text>···}
  <<pre_letter_cmds>>···
\begin{document}
  \begin{letter}{<recipient>}
    \opening{<paragraph_text>···}

    <<environments>>···

    \closing{<paragraph_text>···}
    <<cc>>
    <<encl>>
  \end{letter}
\end{document}
```

Now, the point is inside the (first occurrence of the) fill-in `<paragraph_text>···`. Typing the return address of the letter leaves the buffer in a state like this:[2]

[2] I've stolen the text of this example from the LaTeX book by Leslie Lamport.

```
\documentstyle{letter}
  \address{1234 Ave.\ of the Armadillos \\
           Gnu York, G.Y. 56789
           <<paragraph_text>>···}
  <<pre_letter_cmds>>···
\begin{document}
  \begin{letter}{<recipient>}
    \opening{<paragraph_text>···}

    <<environments>>···

    \closing{<paragraph_text>···}
    <<cc>>
    <<encl>>
  \end{letter}
\end{document}
```

Next, you press `A-k` to remove the trailing `<<paragraph_text>>···`; then expand `<<pre_letter_cmds>>` a second time by pressing `TAB`. Choose the menu entry `\signature` and type your signature. Now, press `A-k` two times — this kills the next two fill-ins (`<<paragraph_text>>` and `<<pre_letter_cmds>>`) and leaves the point inside the fill-in `<recipient>`. Pressing `TAB` expands that fill-in and positions the point inside the fill-in `<addressee>`:

```
\documentstyle{letter}
  \address{1234 Ave.\ of the Armadillos \\
           Gnu York, G.Y. 56789}
  \signature{R. (Ma) Dillo \\ Director of Cuisine}
\begin{document}
  \begin{letter}{<addressee><<linebreak>><<address>>}
    \opening{<paragraph_text>···}

    <<environments>>···

    \closing{<paragraph_text>···}
    <<cc>>
    <<encl>>
  \end{letter}
\end{document}
```

When you type `TAB` again, LS-Emacs displays the message "Enter the name of the addressee" in the message window. This means that there is no further expansion: now you have to type the appropriate name. After completing the recipient by typing some text the buffer may look like this:

```
\documentstyle{letter}
  \address{1234 Ave.\ of the Armadillos \\
           Gnu York, G.Y. 56789}
  \signature{R. (Ma) Dillo \\ Director of Cuisine}
\begin{document}
  \begin{letter}{Dr.\ G. Nathaniel Pickering \\
                 Acme Exterminators \\
                 33 Swat Street \\
                 Hometown, Illinois 62301
                }
    \opening{<paragraph_text>···}

    <<environments>>···

    \closing{<paragraph_text>···}
    <<cc>>
    <<encl>>
  \end{letter}
\end{document}
```

The position of the point is just behind the zip-code. To go to the next fill-in, press `A-n`. Typing the rest of the letter takes much less time than explaining in detail how to do it. When the letter is complete, there won't remain any fill-in in the buffer. Using Lamport's example, you would see:

```
\documentstyle{letter}
  \address{1234 Ave.\ of the Armadillos \\
           Gnu York, G.Y. 56789}
  \signature{R. (Ma) Dillo \\ Director of Cuisine}
\begin{document}
  \begin{letter}{Dr.\ G. Nathaniel Pickering \\
                 Acme Exterminators \\
                 33 Swat Street \\
                 Hometown, Illinois 62301
                }
    \opening{Dear Nat.}

    I'm afraid that the armadillo problem is still with us. I did everything
    \dots\ and I hope you can get rid of the nasty beasts this time.

    \closing{Best regards,}
    \cc{Jimmy Carter \\ Richard M. Nixon}
  \end{letter}
\end{document}
```

There wasn't any need for typing markup — you could concentrate on the contents of the letter while letting LS-Emacs do the dirty work. For languages like TEX and its friends which use a lot of strange and error-prone markup this is a substantial advantage.

2.2 Writing a Bash Shell Script with LS-Emacs

This section shows how to use LS-Emacs to write shell-scripts for the bash shell. The example develops a script for compiling LS-Emacs languages. The parameters to the shell script specify the languages to be compiled. The shell script assembles an Emacs command file and starts Emacs. We want to allow wild-cards for the language names.

To start using the templates for bash, issue the command `lse-bash-mode`.[3] If the buffer was previously empty, LS-Emacs will insert the fill-in `<<bash>>` into the buffer:

```
<<bash>>
```

Expanding this fill-in by pressing `TAB` results in:

```
#! /bin/bash
#<<header-comment>>
<<command>>···
```

Normally, you would now expand the header comment, but in the interest of brevity (of the resulting buffer) we will postpone that. Instead, press `A-n` to position to the fill-in `<<command>>···`. Expanding that fill-in, you see a small menu:

```
#> bash-command
   function-command
   unix-command
   unix-root-command
```

You could select one of the entries, which would immediately pop up another menu. Selecting a command this way is rather time consuming and boring. Instead, abort the expansion by typing `C-g`. We will type the next commands without expanding fill-ins.

First, we have to initialize a variable in which we will assemble the Emacs command. So we type over the fill-in `<<command>>···`. This results in the buffer:

[3] You can configure automatic use of the bash-templates by adding a suitable entry to '`lse-mode-alist.el`'.

```
#! /bin/bash
#<<header-comment>>
emacs_cmd=""
<<command>>···
```

Next, we have to loop over all arguments supplied on the command line. We could expand the fill-in, but it is faster to use a token instead. Knowing that we need a for-loop, we type `for` and press `A-e`. Now, the buffer looks like:

```
#! /bin/bash
#<<header-comment>>
emacs_cmd=""
for <name> <<in-word>>
do
    <command>···
done
<<command>>···
```

The point is inside the fill-in `<name>`. We type the name `arg` and kill the fill-in `<<in-word>>` by pressing `A-k`, as the for-loop will implicitly loop over the argument array. Now, the point is inside the fill-in `<command>···`. To parse wild-cards we need a second for-loop. Again, we are going to use a token. This time, we replace the fill-in `<name>` by `file` and now we have to expand the fill-in `<<in-word>>`. This results in:

```
#! /bin/bash
#<<header-comment>>
emacs_cmd=""
for arg
do
    for file in <text>
    do
        <command>···
    done
    <<command>>···
done
<<command>>···
```

The master language files are located in a directory specified by the environment variable `EMACSLSESRC` and are named '`lse-language-<name>.lse`'. We type the appropriate string. Inside the nested loop we have to extract the language name from the file-name, test for the existence of the file and add it to the `emacs_cmd`. Typing the code — again using just a few tokens — the result may look like this:

```
#! /bin/bash
#<<header-comment>>
emacs_cmd=""
for arg
do
    for file in $EMACSLSESRC/lse-language-$arg.lse
    do
        f=‘echo $file | sed 's!^.*/lse-language-!!
                             s!\.lse!!
                            ’‘
        if [ -s $file ]
        then
            emacs_cmd="$emacs_cmd(lse-language:compile \"$f\")  "
            echo -n "$f "
        else
            echo "Lse language $f does not exist"
            exit 1
        fi
    done
    <<command>>···
done
<<command>>···
```

Now, we have completed the loop for assembling the Emacs command and delete the fill-in remaining in the outer loop by pressing `A-k`. Typing the call of Emacs is a minute matter and leaves the buffer as:

```
#! /bin/bash
#<<header-comment>>
emacs_cmd=""
for arg
do
    for file in $EMACSLSESRC/lse-language-$arg.lse
    do
        f=‘echo $file | sed ’s!^.*/lse-language-!!
                             s!\.lse!!
                            ’‘
        if [ -s $file ]
        then
            emacs_cmd="$emacs_cmd(lse-language:compile \"$f\")  "
            echo -n "$f "
        else
            echo "Lse language $f does not exist"
            exit 1
        fi
    done
done

echo ""
if [ ":$emacs_cmd:" != "::" ]
then
    cmd_file="/tmp/#lse_compile_language#$$"
    echo "$emacs_cmd" > "$cmd_file"
    emacs -batch -l ls-emacs -l "$cmd_file"
    rm "$cmd_file"
fi
```

To complete the shell script, we should now add the header comment. By pressing `A-p` we position the point to the fill-in `<<header-comment>>`. This is the last fill-in remaining in the buffer and we expand it by pressing `TAB`. This yields:

```
#! /bin/bash
# (c) 1994 Swing Informationssysteme GmbH. All rights reserved.
#++
# Name
#    test
#
# Purpose
#    <text>···
#<<header-comment-parameters>>
# Revision Dates
#    29-Jun-1994 (CT) Creation
#    <<revision-date>>···
#--
emacs_cmd=""
for arg
do
    for file in $EMACSLSESRC/lse-language-$arg.lse
    do
        f=`echo $file | sed 's!^.*/lse-language-!!
                             s!\.lse!!
                            '`
        if [ -s $file ]
        then
            emacs_cmd="$emacs_cmd(lse-language:compile \"$f\")  "
            echo -n "$f "
        else
            echo "Lse language $f does not exist"
            exit 1
        fi
    done
done

echo ""
if [ ":$emacs_cmd:" != "::" ]
then
    cmd_file="/tmp/#lse_compile_language#$$"
    echo "$emacs_cmd" > "$cmd_file"
    emacs -batch -l ls-emacs -l "$cmd_file"
    rm "$cmd_file"
fi
```

Now we add a short abstract and a parameter description. That's all. The final result is:

```
#! /bin/bash
# (c) 1994 Swing Informationssysteme GmbH. All rights reserved.
#++
# Name
#    test
#
# Purpose
#    Compile lse language definitions
#
# Parameters
#    $*  [language-name] Name(s) of languages to compile
#
# Revision Dates
#    29-Jun-1994 (CT) Creation
#--
emacs_cmd=""
for arg
do
    for file in $EMACSLSESRC/lse-language-$arg.lse
    do
        f=`echo $file | sed 's!^.*/lse-language-!!
                            s!\.lse!!
                           '`
        if [ -s $file ]
        then
            emacs_cmd="$emacs_cmd(lse-language:compile \"$f\")  "
            echo -n "$f "
        else
            echo "Lse language $f does not exist"
            exit 1
        fi
    done
done

echo ""
if [ ":$emacs_cmd:" != "::" ]
then
    cmd_file="/tmp/#lse_compile_language#$$"
    echo "$emacs_cmd" > "$cmd_file"
    emacs -batch -l ls-emacs -l "$cmd_file"
    rm "$cmd_file"
fi
```

Of course, it won't work. But there is just one little error. You have to specify the directory where Emacs will find '`ls-emacs`'. And of course some environment variables have to be defined correctly. See the chapter on installation of LS-Emacs.

3 Editing with LS-Emacs

This chapter describes the important concepts of LS-Emacs and the interactive commands provided by LS-Emacs. The primary commands of LS-Emacs are bound to keys.

In addition to unprefixed keys, LS-Emacs uses two prefix keys. The `GOLD` prefix is bound to the upper-left key of the application keypad (labeled `PF1` on Digital terminals, `NUMLOCK` on PC's). The `BLUE` prefix is bound to the right neighbor of the `GOLD` key (labeled `PF2` on Digital terminals, `/` on PC's). You can rebind `GOLD` and `BLUE` by using `global-set-key`, too.

For many keys, the combination `GOLD <key>` will undo the effects of `<key>`; while `BLUE <key>` will undo the effect of `GOLD <key>`.

You can use the most important language specific functions of LS-Emacs via the Emacs menu bar.

3.1 Concepts of LS-Emacs

The fundamental concepts of LS-Emacs are *language*, *fill-in*, and *token*. A **language** can be a programming language, a markup language of a text processing system like TEX, or any other formal or not-so-formal language for which LS-Emacs provides templates. It is easy, if a bit boring, to define a LS-Emacs language. It is even easier to change an existing one.[1]

A **fill-in** is inserted into the buffer by LS-Emacs as you expand another fill-in or a token. You recognize a fill-in by the special delimiters surrounding it. A fill-in indicates that some work remains to be done to complete the document. When you start editing a new document, LS-Emacs will normally provide the root fill-in of the corresponding language. As you expand the root fill-in, LS-Emacs will insert more and more other fill-ins into the buffer.

A **token** is in some respects similar to an `abbrev` as supported by standard Emacs. A token is characterized by a name and an expansion. When you type (part of) the name of a token into a buffer and issue the command `lse-expand` (`A-e`), LS-Emacs will expand the token according to its type. Tokens allow fast and direct access to language specific templates.

3.2 Using Languages

The language specifies which fill-ins and tokens are defined, and controls many aspects of the behavior of LS-Emacs including the amount of indentation used and the definition of word boundaries. The language definition may control any of the variables used by standard Emacs or LS-Emacs.

The choice of language is buffer-specific and closely related to the concept of major modes of standard Emacs. Many languages are chosen automatically by hooks related to a major mode. For

[1] Of course, it's not quite as easy to fine-tune a language for maximum effectiveness and ease of use, but that's got nothing to do with LS-Emacs itself.

instance, the language `texinfo` is chosen whenever the major mode `Texinfo` is activated. Any file with extension '`.texi`' will thus automatically be edited with language `texinfo`.

You can change the language used for editing the current buffer at any time by using the command `lse-language:use`. Changing the language with `lse-language:use` does not change the major mode, although changing the major mode may change the current language.

3.3 Working with Fill-Ins

Fill-ins are automatically inserted into the buffer by LS-Emacs. Fill-ins help you by supplying the appropriate syntactical structure for a specific context. You (and LS-Emacs) recognize a fill-in by the special delimiters surrounding it. By default, guillemets are used as basic fill-in delimiters. (If your version of Emacs does not support 8-bit character sets like ISO-Latin-1, you will probably use modified angle brackets (`:<`, `>:`) instead. In this manual, '`<`' and '`>`' are used to represent basic fill-in delimiters.) For different forms of fill-ins the basic delimiters are extended by additional characters (see Section 3.3.2 [Fill-In-Forms], page 18).

There are three important aspects of fill-ins:

state Describes the temporal development of a fill-in. State transitions are caused by the user by issuing the appropriate commands.

form Describes the spatial properties of a fill-in. The form of a fill-in is determined by the context in which it is used.

type Describes the invariant expansion properties of a fill-in. The type is defined by the designer of the fill-in.

3.3.1 States of Fill-Ins

While you edit a document, its fill-ins will change their state in response to the commands you issue. Each fill-in starts its life in the *flat* state.[2]

LS-Emacs displays a flat fill-in by inserting the name of the fill-in inside the appropriate delimiters into the buffer (see Section 3.3.2 [Fill-In-Forms], page 18). Depending on your decision a flat fill-in may change its state to either the *deep* or the *dead* state. When you expand or replace the fill-in, its state will change to the *deep* state (see Section 3.3.5 [Fill-In-Replacement], page 19, and Section 3.3.6 [Fill-In-Expansion], page 20). When you kill a fill-in, its state changes to the *dead* state (see Section 3.3.8 [Fill-In-Killing], page 22).

You can compare the states of fill-ins to the aggregate states of materials. The flat state of a fill-in corresponds to the liquid state of a material. Expansion of the flat fill-in corresponds to the

[2] It is possible that a fill-in is expanded automatically in a specific context and never appears as a flat fill-in in the buffer, but there is no way for the user of the language to detect this. Only the template designer knows what is going on then.

freezing of the liquid — whereas the liquid can take many shapes, may evaporate or crystallize, the shape of the solid — just like the contents of the deep fill-in — is fixed.

You can toggle between the flat and the deep state and between the flat and the dead state. Yet, there are no direct state transitions between the deep and the dead state.

3.3.2 Forms of Fill-Ins

Fill-Ins are used in different forms. The form determines the behavior of a fill-in and the operations you can apply to it. The form of a fill-in is context-dependent. One and the same fill-in will occur in different forms when used in different contexts.

`<gnu>` This is a required fill-in. You cannot remove a required fill-in with the command `lse-kill-fill-in`.

`<<gnat>>` This is an optional fill-in. You may remove any optional fill-in from the buffer by the command `lse-kill-fill-in`.

`<gnu-list>···`
This is a (required) list-fill-in. When you expand or replace the list-fill-in, LS-Emacs automatically duplicates it (using an optional list-fill-in as duplicate).

`<|armadillo|>`
This is a non-replaceable fill-in. That means, you can expand the fill-in, but you cannot replace it. This form is rarely used, but very effective.

3.3.3 Navigating between Fill-Ins

You are not forced to work on the fill-ins in the sequence in which they are inserted into the buffer by LS-Emacs (although that is the normal procedure). Use the command `lse-goto-next-fill-in` (`A-n`) to position the point to the next fill-in (the point moves forward). Use the command `lse-goto-prev-fill-in` (`A-p`) to position the point to the previous fill-in (the point moves backward).

The command `lse-goto-last-position` (`gold A-n`) moves the point to the position where the last `lse-goto-next-fill-in` or `lse-goto-prev-fill-in` was issued.[3] Of course, you can also move the point to a fill-in by standard Emacs commands like `forward-char`.

The point is considered to be inside a fill-in if it is between the fill-in delimiters or just at the first character of the trailing delimiter. If you move the point by standard Emacs commands, LS-Emacs may not always detect immediately that the point is inside a fill-in. In some situations you have to give an explicit LS-Emacs command for that.

LS-Emacs keeps a list of markers for the positions of all fill-ins expanded or replaced in a buffer. For each fill-in, LS-Emacs keeps two markers: one (called the head) points to the first

[3] The last occurrence of such a command could have occurred implicitly. LS-Emacs uses the command `lse-goto-next-fill-in` when it expands or kills fill-ins.

character of the fill-in's expansion or replacement, the other (called the tail) points to the last character of the deep fill-in. Use the commands `lse-fill-in-marks:goto-next-head` (`M-n`) and `lse-fill-in-marks:goto-next-tail` (`M-s-n`) to go to the head or tail of the next fill-in (the point moves forward), respectively. Use the commands `lse-fill-in-marks:goto-prev-head` (`M-p`) and `lse-fill-in-marks:goto-prev-tail` (`M-s-p`) to move the head or tail of the next fill-in in the backward direction.

The expansion or replacement of a fill-in is also marked by text properties. The commands `lse-goto-next-expansion` (`s-n`) and `lse-goto-prev-expansion` (`s-p`) move the point to the next or previous character of the buffer where the value of this text property changes, respectively. The command `lse-goto-parent-expansion-head` moves point to the first character of the fill-in which expansion contained the current fill-in.

3.3.4 Getting Help on Fill-Ins

With the point inside a fill-in you may request help about the meaning of the fill-in. Use the command `lse-describe-fill-in` (`A-o`) to call up a short description which appears in the message window. Use the command `lse-help-fill-in` (`gold A-o`) to call up a help window for the fill-in. The command `lse-window:restore-temp-hidden` (`blue A-f`) removes the help window and restores the previously viewed buffer.

3.3.5 Replacing Fill-Ins

Replacement of a fill-in means that the fill-in is replaced by the text you enter. You can start replacement by issuing the command `lse-replace-fill-in` (`A-r`). Or, when LS-Emacs knows you are inside a fill-in, you can replace the fill-in by just typing text. In either case, LS-Emacs will automatically remove the flat fill-in from the buffer and insert your text in its place.

You terminate the replacement by issuing any LS-Emacs command dealing with fill-ins, e.g., by issuing `lse-goto-next-fill-in` or by repeating `lse-replace-fill-in`. The text between the position where the replacement started and the position where you terminate the replacement is considered the deep-state contents of the fill-in.

While the replacement is in progress, LS-Emacs displays the name of the replaced fill-in enclosed in angle brackets in the window's status line. When using the X window system, LS-Emacs highlights the region of the deep fill-in in a different color.

The deep-state contents of a fill-in is removed by the command `lse-unreplace-fill-in` (`gold A-r`)[4], which restores the flat state of the fill-in. The deep-state contents is also used by the command `lse-replicate-fill-in` (see Section 3.3.7 [Fill-In-Replication], page 21).

[4] You can also use `lse-unexpand-fill-in` to remove the deep-state contents.

If the replacement of a fill-in was terminated too early, you can use `lse-unreplace-fill-in` immediately followed by `lse-rereplace-fill-in` (`blue A-r`). Then you can extend the replacement by typing additional text.

When replacement begins, LS-Emacs may insert leading and trailing text defined for the fill-in — typical examples are quotes or brackets surrounding the replacement. Leading and trailing text are specified by the definition of the fill-in.

When replacement is terminated, LS-Emacs may perform some completion actions associated to the fill-in. For instance, LS-Emacs may insert blanks to align a field properly. If the fill-in is defined to *auto-replicate*, LS-Emacs will replace the following occurrences of the fill-in by the replacement value (how many is specified by the definition of the fill-in).

3.3.6 Expanding Fill-Ins

You use the command `lse-expand` (`A-e`) to initiate the expansion of a fill-in. The point has to be inside a flat fill-in when you issue this command. Expansion of a fill-in means that LS-Emacs interprets the definition of the fill-in and acts accordingly. The type of the expanded fill-in determines what LS-Emacs will do:

- When expanding a *replacement-fill-in*, LS-Emacs removes the flat fill-in from the buffer and inserts the predefined replacement as deep-state contents.
- When expanding a *menu-fill-in*, LS-Emacs displays a menu of alternatives and expands the alternative you select (which could be a literal replacement text or any type of fill-in).
- For a *function-fill-in* LS-Emacs executes the lisp-code defined as expansion. In this case, the flat (state of the) fill-in is **not** removed. Normally, function-fill-ins are defined as non-replaceable.
- For a *terminal fill-in* LS-Emacs displays a help line in the message window. In this case, the flat (state of the) fill-in is **not** removed. To remove the flat state of a terminal fill-in from the buffer you have to use either of the commands `lse-replace-fill-in` or `lse-kill-fill-in`.

You can undo an expansion with the command `lse-unexpand-fill-in` (`gold A-e`) which replaces the deep-state contents of the fill-in by its flat state. The command `lse-unexpand-fill-in` always un-expands the most recently expanded fill-in. If you use it repetitively, LS-Emacs will un-expand earlier and earlier expansions. With the command `lse-reexpand-fill-in` (`blue A-e`) you can undo the un-expansions.

The un-expansion/re-expansion mechanism works best if you use it soon after the expansion concerned. Due to implementation constraints, problems can occur when you delete text from the boundaries of the range of the fill-in by normal Emacs commands. In addition, you should not use the standard Emacs' `undo` command for dealing with fill-ins. Doing so destroys LS-Emacs' information about the fill-ins concerned.

3.3.6.1 Expanding Replacement-Fill-Ins

A replacement-fill-in is characterized by a predefined replacement text and/or lisp-code. On expansion, LS-Emacs inserts the replacement text into the buffer and executes any lisp-code associated with the replacement. This is very similar to normal replacement: in that case you supply the replacement text, now it is supplied by LS-Emacs automatically. As for manual replacement, LS-Emacs may insert leading and trailing text and perform completion actions and automatic replication according to the definition of the fill-in.

LS-Emacs controls the indentation of the replacement according to the definition of the expanded fill-in. Normally, you do not have to — and should not — worry about indentation when expanding fill-ins (or tokens).[5]

3.3.6.2 Expanding Menu-Fill-Ins

A menu-fill-in is characterized by a number of alternatives. Each alternative can be either a literal replacement text or another fill-in to be expanded.

On expansion of a menu-fill-in, LS-Emacs displays the menu in another window. You can select a menu-entry by using the cursor or mouse keys or by typing the name of the entry. When you enter the name by typing, LS-Emacs does completion for you. The currently selected menu-entry is marked by an arrow and additionally shown in the status line of the menu-window. When using the X window system, LS-Emacs highlights the current selection in a different color.

To choose the selected entry, just hit `RET` or `TAB` or `A-e`. LS-Emacs then expands the chosen menu-entry. If the chosen menu-entry is itself a menu-fill-in, another menu will be displayed. To cancel the expansion while the menu window is shown, use `C-g`.

For more information on LS-Emacs completion, see Section 3.5 [Using LS-Emacs Completion], page 23.

3.3.7 Replicating Fill-Ins

Replication of a fill-in means that LS-Emacs uses the replacement of a previous occurrence of the same fill-in. You can request replication by the command `lse-replicate-fill-in` (`A-s`). If you want to replicate an earlier occurrence of the fill-in just continue to use (`A-s`). Each repetition of `lse-replicate-fill-in` will replicate earlier and earlier occurrences of the fill-in.

---

[5] If the language specific templates are defined properly. If the indentation of a fill-in does not satisfy you, you should try to tune the template definition.

3.3.8 Killing Fill-Ins

You can remove an optional fill-in with the command `lse-kill-fill-in` (`A-k`). If you try to kill a required fill-in, LS-Emacs will ask for confirmation — if you answer with `y`, LS-Emacs will kill it. You cannot kill a non-replaceable fill-in.

If you were too eager in killing a fill-in, you can restore it by using the command `lse-unkill-fill-in` (`gold A-k`). Unlike un-expansion, you cannot un-kill more than once.

You can remove all optional fill-ins between the point and the end of the buffer by the command `lse-kill-all-optional-fill-ins` (`blue gold A-k`).

You should always use `lse-kill-fill-in` to remove a fill-in from the buffer rather than using standard Emacs commands for killing. Otherwise, information about the fill-in history will most probably be lost![6]

3.4 Working with Tokens

Fill-ins provide the basic structure of a document. By expanding fill-ins you can save a lot of typing effort and you are able to use languages with little knowledge of the language details. Yet, fill-ins alone are insufficient:

- You are restricted to expanding fill-ins already in the buffer. If you want to extend or modify an existing document, there won't be any fill-in in the buffer.
- Expanding a fill-in can lead you through many menu-levels until you finally arrive at terminal fill-ins. This may be fine if you are inexperienced with the language, but it is rather inefficient and boring for an experienced user.

Tokens deal with these shortcomings of fill-ins. You can type a token anywhere in a buffer, no matter if there are flat fill-ins around or not. You don't have to type the complete name of the token — any unique abbreviation will do. If you type an ambiguous token name, LS-Emacs will pop up a completion window showing all alternatives which match the ambiguous name. You can then select one of the alternatives (see Section 3.5 [Using LS-Emacs Completion], page 23).

You initiate token expansion by the command `lse-expand-token` (`A-e`) with the point at the end of the token name. LS-Emacs will always try to expand the longest match for a token name (the maximum length of a token name is 5 words on the same line). Un-expansion and re-expansion of tokens work exactly as for fill-ins.

Most tokens are related to fill-ins — although the names quite often are different. If you expand such a token, LS-Emacs will act as if the corresponding fill-in had been expanded. Other tokens expand into text directly or cause some lisp-function to be executed.

[6] Any volunteers to enhance the implementation?

3.5 Using LS-Emacs Completion

LS-Emacs uses a special completion mode whenever you have the choice between a number of alternatives. This completion mode was inspired by the completion mechanism offered by standard Emacs. In LS-Emacs completion mode, LS-Emacs pops up a window displaying the menu of alternatives. For each entry LS-Emacs displays the name of the alternative and, if available, a help text which explains the meaning of the entry. To obtain more help on an entry, press one of the keys `f1`, `gold ?`, or `help`.

In LS-Emacs completion mode, you can either type the name of the alternative or select it by moving the point to it. You can mix both ways of selecting an alternative. For details on the working of the completion mechanism see the following subsections.

LS-Emacs displays the current match in the left field of the status line of the completion window. As more than one entry may fit the current match, the right field of the status line shows the menu entry which would be selected if you used a key to exit the completion. The current line of the menu is marked by an overlay cursor (and displayed in a different color on a X window display).

3.5.1 Typing the name of an alternative

When you type the name of an alternative, LS-Emacs will automatically extend the current match as much as possible. That means, typing one character could be enough to specify the selection completely if only one alternative starts with the character you typed. LS-Emacs will narrow the menu to those entries which match the string already completed.[7]

You can delete part of the current match by using the keys `DEL`, `LFD`, and `A-u`: `DEL` deletes one character, `LFD` deletes the part of the match caused by the last character you typed, and `A-u` deletes the entire match.

3.5.2 Moving the point to select an alternative

You can use `[up]` and `[down]` to move to the next or previous alternative, `[prior]` and `[next]` to move the point up or down by about half of the window height, and `[gold kp5]` and `[gold kp4]` to move to the first or last menu entry, respectively.

Moving the point clears the current match.

You can use `[find]` and `[gold pf3]` to search for a line containing a string; use `[pf3]` to search for the next occurrence of the string.

3.5.3 Key bindings of LS-Emacs completion mode

`A-e` Exit completion mode and use current selection as completion.

[7] This will work only if the menu is sorted.

`C-g` Abort completion: no value is used as completion.

`TAB` Exit completion mode with current selection.

`A-j` Delete the part of the current match which was caused by the last character typed (can be done repeatedly).

`A-k` Abort completion.

`RETURN` Exit completion with current selection.

`A-u` Delete the current match entirely.

`[del]` Delete last character of current match.

`[down]` Select next menu entry (one line down).

`[find]` Search for line containing a string.

`[gold ??]` Display help for menu entry. This command pops up a second window with help information.

`[gold ?s]` Sort the menu entries.

`[gold kp4]`
Select last menu entry.

`[gold kp5]`
Select first menu entry.

`[gold pf3]`
Search for line containing a string.

`[help]` Display help for menu entry. This command pops up a second window with help information.

`[left]` Scroll window to the left.

`[next]` Move the point by about half the height of the window down.

`[pf3]` Search for next occurrence of last search string (does not work if you are still in the line found last).

`[prior]` Move the point by about half the height of the window up.

`[right]` Scroll window to the right.

`[up]` Select previous menu entry (one line up).

3.5.4 Example of completion

Suppose the current language defines three tokens starting with the character 's': 'section', 'subsection', and 'subsubsection'. If you try to expand the token 's', LS-Emacs pops up a window with these three words.

Initially, the current match consists of the string 's'. If you typed 'e' in response, the current match expands to 'section' and the completion window is restricted to one line. Typing `A-j` cuts the current match back to 's'.

Typing 'u' narrows the completion window to two lines displaying the choices 'subsection' and 'subsubsection'. The current match is now 'subs'.

3.6 Other Editing Features

This section describes other features of LS-Emacs which are not directly concerned with language sensitivity. Many of these features emulate the behavior of Digital editors like `EDT`, `EVE`, and `LSE`; others were implemented in `TPU` by the author and ported to Emacs as part of the implementation of LS-Emacs.

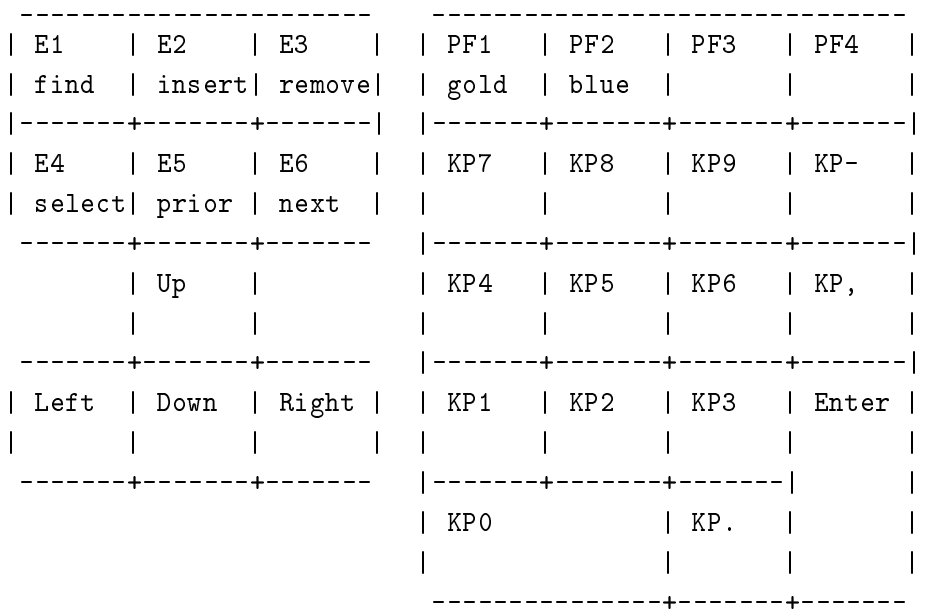
If you are familiar with the style of the above-mentioned editors, you won't need much explanations — just enjoy these features. If you are used to standard Emacs, perhaps you will find some nice features to use — you can combine those you like with the features of standard Emacs familiar to you.

The implementation of some of the features described in this section was taken from the elisp-library '`tpu-edt.el`', version 3.2, by Rob Riepel.

3.6.1 Key bindings

LS-Emacs relies heavily on a set of keys found on Digital terminals like the `VT-200` to the right of the main keyboard, specifically the so-called application keypad and the mini-keypad. You find a rather similar keyboard layout on the standard PC keyboard — the keypad lacks one key, and the keys are named differently. Most contemporary keyboards can emulate a VT-style keyboard.

The following diagram shows the layout and the keynames as used in Digital's editors:



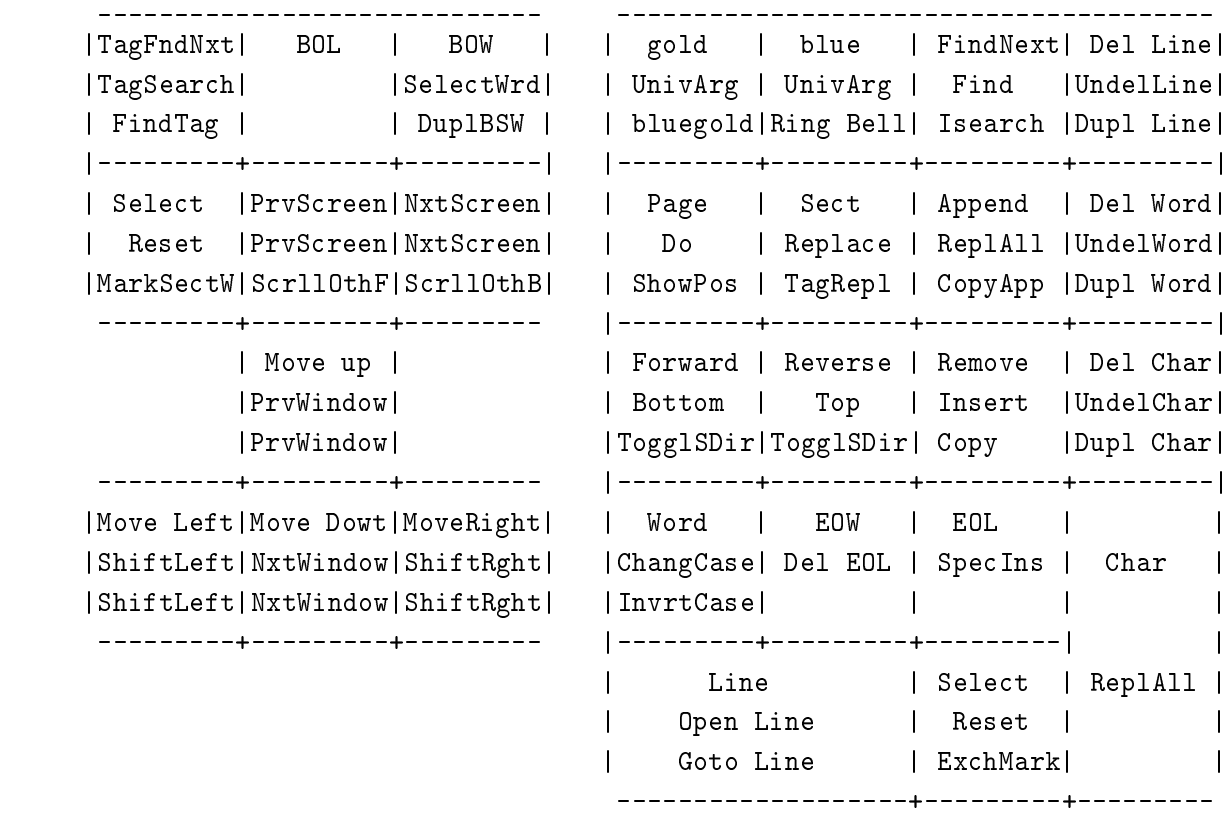
The application keypad can be in either of two modes: in numeric mode the keys generate digits, in application mode they generate escape sequences which are mapped to editor commands. By default, LS-Emacs uses application mode. The keys of keypad and mini-keypad are bound to the most commonly used editor commands. Using them you can move through the buffer; delete and undelete characters, words, and lines; search and substitute; select and manipulate a region; and more. LS-Emacs binds the keys a bit differently than the standard key-bindings of Digital editors.[8]

Of special importance is the key labeled `PF1`, called the *gold-key*. It is used as prefix key to bind a second function to each of the keypad keys. In addition, LS-Emacs uses a second prefix-key, the *blue-key* which is bound to `PF2`.[9]

The following diagram shows the keypad-bindings of LS-Emacs. The commands bound to the keypad keys will be explained in the following sections. The first line in each key specifies the command bound to the unprefixed key, the second line the binding of the key prefixed by the gold-key, the third line the binding of the key prefixed by the blue-key:

[8] Those bindings were originally designed for the `VT-100` terminal, which lacks a mini-keypad. When the `VT-200` terminal was introduced, that lead to redundant key-bindings. And, some original bindings were to functions I've never used.

[9] `EDT` and friends don't use a *blue-key*; instead they bind the help-command to the `PF2` key.



The following table provides a short explanation of the bindings of the keypad keys:

`find` Bound to `tags-loop-continue` (see GNU Emacs Manual, chapter 'Editing Programs', section 'Tag Tables').

`gold find` `tags-search`.

`blue find` `find-tag`.

`insert` `lse-tpu:next-beginning-of-line`.

`remove` `lse-tpu:goto-prev-bs-word-head`.

`gold remove`
`lse-select-current-word`.

`blue remove`
`lse-tpu:duplicate-previous-bs-word`.

`select` `lse-tpu:select`.

`gold select`
`lse-tpu:unselect`.

`blue select`
`mark-section-wisely` (see GNU Emacs Manual).

```
blue gold select
          lse-select-current-bs-word.

prior     (lse-previous-screen 2).

gold prior
          lse-previous-screen.

blue prior
          lse-scroll-other-window-forw.

next      (lse-next-screen 2).

gold next  lse-next-screen.

blue next  lse-scroll-other-window-back

up        lse-tpu:previous-line.

gold up   lse-previous-window.

blue up   lse-previous-window.

blue gold up
          enlarge-window.

down      lse-tpu:next-line.

gold down  lse-next-window.

blue down  lse-next-window.

blue gold down
          shrink-window.

left      lse-tpu:backward-char.

gold left  (lse-indent-rigidly (- (lse-tab-increment))).

blue left  (lse-indent-rigidly (* -3 (lse-tab-increment))).

blue gold left
          lse-tpu:pan-left.

right     lse-tpu:forward-char.

gold right
          (lse-indent-rigidly (lse-tab-increment)).

blue right
          (lse-indent-rigidly (* 3 (lse-tab-increment))).

blue gold right
          lse-tpu:pan-right

pf1       Gold prefix key.
```

`gold pf1` `universal-argument` (see GNU Emacs Manual).

`blue pf1` Blue-gold prefix key.

`pf2` Blue prefix key.

`gold pf2` `universal-argument`.

`blue pf2` `lse-ring-bell`.

`pf3` `lse-tpu:search-again`.

`gold pf3` `lse-tpu:search`.

`blue pf3` `isearch-forward` (see GNU Emacs Manual).

`pf4` `lse-tpu:delete-next-line`.

`gold pf4` `lse-tpu:undelete-line`.

`blue pf4` `lse-tpu:duplicate-previous-line`.

`blue gold pf4`
`lse-tpu:delete-next-line-append`.

`blue gold blue pf4`
`lse-tpu:duplicate-word-in-previous-line`.

`kp0` `lse-tpu:line`.

`gold kp0` `lse-open-line`.

`blue kp0` `goto-line`.

`kp1` `lse-tpu:goto-word-head`.

`gold kp1` `lse-tpu:change-case`.

`blue kp1` `lse-tpu:invert-case`.

`kp2` `lse-tpu:goto-bs-word-tail`.

`gold kp2` `lse-tpu:delete-tail-of-line`.

`kp3` `lse-tpu:end-of-line`.

`gold kp3` `lse-tpu:special-insert`.

`kp4` `lse-tpu:advance-direction`.

`gold kp4` `lse-tpu:move-to-end`.

`blue kp4` `lse-tpu:toggle-search-direction`.

`kp5` `lse-tpu:backup-direction`.

`gold kp5` `lse-tpu:move-to-beginning`.

```
blue kp5   lse-tpu:toggle-search-direction.

kp6        lse-tpu:cut-region.

gold kp6   lse-tpu:paste-region.

blue kp6   lse-tpu:copy-region.

kp7        lse-tpu:page.

gold kp7   lse-command:do.

blue kp7   lse-show-position.

blue gold kp7
           line-to-top-of-window.

kp8        lse-tpu:scroll-window.

gold kp8   lse-tpu:replace.

blue kp8   tags-query-replace.

blue gold kp8
           lse-tpu:replace-all.

kp9        lse-tpu:cut-append-region.

gold kp9   lse-tpu:replace-all.

blue kp9   lse-tpu:copy-append-region.

kp-        lse-tpu:delete-next-word.

gold kp-   lse-tpu:undelete-word.

blue kp-   lse-tpu:duplicate-previous-word.

blue gold kp-
           lse-tpu:delete-next-word-append.

kp,        lse-tpu:delete-next-char.

gold kp,   lse-tpu:undelete-char.

blue kp,   lse-tpu:duplicate-previous-char.

blue gold kp,
           lse-tpu:delete-next-char-append.

kp.        lse-tpu:select.

gold kp.   lse-tpu:unselect.

blue kp.   lse-tpu:exchange-point-and-mark.

blue gold kp.
           lse-blink-select-mark.
```

`enter` `lse-tpu:forward-char`.

`gold enter` `lse-tpu:replace-all`.

In addition to the keypad, Digital editors use a number of control keys bound to common editing functions. The use of these keys conflicts with the standard Emacs conventions. Therefore LS-Emacs binds these keys to the corresponding `ALT <key>` combinations.

The following table provides a short explanation of the bindings of the alt-keys:

`A-a` `lse-tpu:toggle-overwrite-mode`.

`A-b` `repeat-complex-command`.

`A-d` `dabbrev-expand`.

`A-h` `lse-tpu:next-beginning-of-line`.

`A-j` `lse-tpu:delete-prev-word`.

`A-l` `lse-tpu:insert-formfeed`.

`A-u` `lse-tpu:delete-head-of-line`.

`A-v` `lse-tpu:quoted-insert`.

`A-w` `redraw-display`.

`DEL` `lse-tpu:delete-prev-char`.

3.6.2 Editing-Direction

LS-Emacs allows you to use two directions of editing. The default direction is *forward*, which means that the point moves to the right or down in the buffer. The other direction is *backward* or *reverse*, which means that the point moves to the left or up in the buffer. Many commands of LS-Emacs take the current direction into account and behave accordingly, e.g., `lse-tpu:char`, `lse-tpu:end-of-line`, `lse-tpu:goto-word-head`, `lse-tpu:line`, `lse-tpu:search`.

LS-Emacs displays the — buffer-local — direction in the status line: '`vvv`' signifies forward-direction, '`^^^`' signifies backward-direction. You can change the current direction with the commands `lse-tpu:advance-direction` ([`KP4`]) and `lse-tpu:backup-direction` ([`KP5`]).

3.6.3 Words

LS-Emacs handles words differently than standard Emacs. First, commands dealing with the current word look from the point backwards, i.e., the begin of the current word is before the point, not behind the point as in standard Emacs. Second, the syntax of a word is not defined via the syntax-table, but by sets of characters. This allows the simultaneous support of more than one

word syntax. Finally, there are commands for moving either to the begin or the end of a word — they move the point according to the current direction.

There are two different modes how LS-Emacs determines the begin and the end of a word. In the *normal mode*, LS-Emacs distinguishes three categories of characters:

word-components
: This category defines the characters which may occur inside a word. You can define the word-components by setting the buffer-local variables `lse-tpu:ident-chars` and `lse-tpu:ident-group-chars` (the syntax corresponds to that of character sets in regular expressions — without the enclosing brackets). The union of these two variables defines the characters considered as word-components. Normally, `lse-tpu:ident-chars` has the value '`A-Za-z0-9`', for use with European languages it will additionally contain umlaut-characters and other diacriticals.

  Each LS-Emacs language defines this category according to the characteristics of the language.

punctuation
: This category defines characters delimiting words. In contrast to white-space characters, LS-Emacs will not skip over multiple occurrences of punctuation characters when searching for the boundary of a word.

white-space
: This category defines characters delimiting words. In contrast to punctuation characters, LS-Emacs will skip over multiple occurrences of white-space characters when searching for the boundary of a word.

In the *other mode*, LS-Emacs will consider only white-space characters as word delimiters, i.e., it will consider all punctuation characters to be part of the category word-components.

3.6.4 Selection

LS-Emacs provides commands for selecting a block of text in the buffer and for manipulating such a selection. This mechanism is similar to the mark/region mechanism of standard Emacs, but it provides more varied commands for the manipulation of the selection.[10]

You start a selection by the command `lse-tpu:select` ([`KP.`]). The value of the point at that time becomes one boundary of the selection — the start-position. Moving the cursor lets you change the other boundary of the selection — the end-position. You can interchange start- and end-position of the selection with the command `lse-tpu:exchange-point-and-mark` ([`blue KP.`]). You can cancel the selection with the command `lse-tpu:unselect` ([`gold KP.`]).

When using the X window system, LS-Emacs highlights the selection in a different color.

[10] The current implementation uses the *mark* provided by standard Emacs. This may change in the future.

While a selection is active, several commands apply to the selection. Some of these are only valid for an active selection:

- `lse-blink-select-mark`
- `lse-tpu:cut-region`
- `lse-tpu:copy-region`
- `lse-tpu:cut-append-region`
- `lse-tpu:copy-append-region`
- `lse-tpu:unselect`

Others manipulate the active selection, if any, or else they manipulate a simple unit of text like a line or a word:

- `lse-fill-range`
- `lse-indent-rigidly`
- `lse-insert-braces`
- `lse-insert-brackets`
- `lse-insert-dquotes`
- `lse-insert-parentheses`
- `lse-insert-squotes`
- `lse-remove-braces`
- `lse-remove-brackets`
- `lse-remove-dquotes`
- `lse-remove-parentheses`
- `lse-remove-squotes`
- `lse-tpu:capitalize-strongly`
- `lse-tpu:capitalize-weakly`
- `lse-tpu:change-case-lower`
- `lse-tpu:change-case-upper`
- `lse-tpu:change-case`
- `lse-tpu:spell-check`
- `lse-untabify-buffer`

Several commands select some range of the buffer:

- `lse-select-angle-range`
- `lse-select-brace-range`
- `lse-select-bracket-range`

- `lse-select-current-bs-word`
- `lse-select-current-word`
- `lse-select-guillemot-range`
- `lse-select-paren-range`

3.6.5 Deletion

LS-Emacs provides commands for deletion and un-deletion of four different kinds of entities: characters, words, lines, and selected ranges. Each of these entities has its own deletion-history. Contrary to the kill/yank mechanism of standard Emacs, LS-Emacs maintains just the last deletion of each of these different kinds and these are independent from each other.

When you un-delete an entity, LS-Emacs chooses the position of the point according to the direction of the deletion — if you deleted in reverse direction, LS-Emacs will leave the point at the end of the un-deletion; if you deleted in forward direction, LS-Emacs will leave the point at the beginning of the un-deletion.

The commands for deletion normally replace the last value of the deletion-history by the newly deleted text; yet, LS-Emacs provides other commands which append the new deletion to the existing deletion-history. The commands for un-deletion don't change the deletion-history — you can un-delete the same text as often as you want.

3.6.6 Mark-Stacks

LS-Emacs provides a number of mark-stacks:

- The global mark-stack is not bound to any specific buffer or window.
- A buffer mark-stack is bound to a specific buffer — each buffer has its own mark-stack.
- A window mark-stack is bound to specific window — each window (except mini-buffer windows) has its own mark-stack.

You can push the current position of the point (buffer inclusively) onto a mark-stack with the command `lse-push-mark-<stack-name>`, where `<stack-name>` is either `global`, or `window`, or `buffer`. You can return to the top-most mark of a mark-stack with the commands `lse-goto-mark-and-pop-<stack-name>` and `lse-toggle-mark-<stack-name>`.

Besides the marks stacked explicitly by `lse-push-mark-<stack-name>`, LS-Emacs maintains two additional marks per mark-stack: the home-mark points to the home-buffer of a window mark-stack or to the first file-buffer created during the editing session; the last-mark points to the position before the execution of the last `lse-goto-buffer` command. You can set the home-mark of a window mark-stack with the command `lse-set-home-mark-window`.

3.6.7 Command-Menu

LS-Emacs provides a completion menu for the most commonly used commands. You can use this feature with the command `lse-command:do` (`gold kp7`). For more information on LS-Emacs completion, see Section 3.5 [Using LS-Emacs Completion], page 23.

You can add any interactive command to that menu with the function `lse-command:add`. This function may not be used interactively, so you may want to use it in your '`.emacs`' initialization file.

3.6.8 Buffer-Ring

LS-Emacs remembers the order of creation of all buffers created on demand by the user. These buffers build a ring. LS-Emacs provides commands to navigate in this buffer-ring. From a given buffer, you can switch to the next older and the next younger buffer by using the commands `lse-goto-prev-buffer` and `lse-goto-next-buffer`, respectively.

The first buffer is considered the home buffer. With the command `lse-goto-home-mark-global` you can switch to that buffer at any time.

4 Defining Templates for a LS-Emacs Language

This chapter describes how you can define and change language templates. It is advantageous if you have a basic understanding of how to program in Emacs Lisp, but such knowledge is not really necessary. If you do not know Emacs Lisp, it might be a good idea to study the definitions of existing LS-Emacs language templates in conjunction with reading this chapter. As LS-Emacs provides templates for the definition of LS-Emacs language templates you need not worry about syntactic detail; LS-Emacs will take care of all the gory details for you.

In principle, one could easily write a program to translate a formal language specification, e.g., in Backus-Naur form, into the syntax required by LS-Emacs. Yet, templates generated in this way would not be user-friendly and rather ineffective.[1] Therefore it is better to perform the translation by hand. For this process, a Backus-Naur form is a very good starting point.

You should always try to consider how the user will work with the templates you are defining. Normally, it is necessary to do a fair amount of prototyping to arrive at effective templates. Alternatively, you can start with simple templates and fine-tune them during the first weeks of use. The latter alternative probably requires more total effort but it is easier if you don't have recent experience in template design.

4.1 Template Definition Files

A LS-Emacs language comprises a number of general parameter settings and a set of fill-ins and tokens.

The definition of a LS-Emacs language consists of a number of files. The language master file defines the characteristics of the language and specifies which template files define the fill-ins and tokens for the language. Because many languages can use the same template file, this allows sharing of templates between different languages. Sharing of templates has three primary advantages:

- Reuse. Each template must be defined only once.
- Consistency between different languages. The user of LS-Emacs has less templates to learn and can transfer the knowledge gathered in using one language to other languages.
- Ease of change. If you want to change something, you have to do less work, and there is no danger of introducing inconsistencies. Fine-tuning of one language benefits all others which share the tuned templates.

Both language master files and template files have extension '`.lse`'. The master file of a language named `<name>` is named '`lse-language-<name>.lse`'. The template files are named '`lse-templates-<x>.lse`'. Normally, one of these files uses the language-name in place of '`<x>`'. The template file '`lse-templates-generic.lse`' is used by each and every language by default.

[1] If you don't believe me, try to use the templates offered by Digital for DEC LSE!

To speed-up the loading of language templates, the definitions are stored in compiled form. For a language named `<name>`, the compiled definitions are stored in the file '`lse-language-<name>.lsc`'. To load a compiled language both the source ('`.lse`') and the compiled ('`.lsc`') language master files are necessary. The first one defines the language characteristics, the second one the compiled fill-ins and tokens. To compile a language, use the command `lse-language:compile`; to load a language, use `lse-language:use`. **Do not use Emacs functions like `load-file`, `load-library`, or `require` to load either language master or template files!**

4.2 Template Definition Directories

LS-Emacs uses two environment variables to find the language definition files. `EMACSLSESRC` contains the pathname of the directory containing the source files (extension '`.lse`'). `EMACSLSEDIR` contains the pathname of the directory containing the compiled language templates (extension '`.lsc`').

4.3 Definition of LS-Emacs Languages

This section explains the parameters or characteristics you can define for a language.

The language characteristics are defined in the language master file '`lse-language-<name>.lse`' which contains exactly one call of the function `lse-language:define`. It **must not** contain any definitions of fill-ins or tokens. In this file you should also define language specific lisp functions, e.g., a hook-function linking the language with a major mode of Emacs.

lse-language:define *name properties hooks files* &optional fill-in-size token-size — Function

With this function, you define the characteristics of a LS-Emacs language.

name — The name of the language (used as part of the file name too). This name is used for manipulating the language inside LS-Emacs. Language related commands of LS-Emacs all require the language name as parameter. Specify a string inside double quotes.

properties — A list of property settings. The primary properties are explained in Section 4.3.1 [Language-Properties], page 38. In addition to LS-Emacs-specific language properties you can define the value of any Emacs variable for the language.[2]

hooks — A list of language hooks. These hooks are Emacs lisp functions which are executed when the language is activated for a buffer. For instance, some languages use `auto-fill-mode` as language hook. You can define any lisp function (which may be called) without parameters as language hook.

[2] Of course, such a variable should be defined as buffer-local.

files A list of files containing template definitions. Do not specify the extension '`.lse`'. The file '`lse-templates-generic.lse`' must not be included in this list — it is added automatically by LS-Emacs. The template files are loaded in the order specified; later template files can mask definitions in earlier files. '`lse-templates-generic.lse`' is loaded before all other template files.

fill-in-size The size of the fill-in (hash-) table. This should be a prime number.

token-size The size of the token (hash-) table. This should be a prime number.

4.3.1 Properties of LS-Emacs Languages

Language properties control the behavior of LS-Emacs. This section describes the properties specific to LS-Emacs. In addition, you can use any buffer-local Emacs variable as language property.

`lse-language:expand-initial`
: Controls whether LS-Emacs will expand the root fill-in of the language when you start editing a new document. By default, LS-Emacs does not expand the root fill-in.

`lse-language:initial-fill-in`
: LS-Emacs inserts the value of this property into each new document of the language. Normally, this is just a flat fill-in. But you can specify any string for this property. Consider setting `lse-language:expand-initial`, when you need a lengthy string for this property.

`lse-language:tab-increment`
: Controls the amount of indentation used per indentation level. Functions like `lse-indent+1` change the indentation level by just this value. Normally, a value of either 2 or 4 is used. See Section 4.6 [Indentation], page 49.

`lse-tpu:ident-chars`
: This property controls — together with `lse-tpu:ident-group-chars` — what LS-Emacs considers to be a word. The default value are the alphanumeric characters `A-Za-z0-9`. If the language allows additional letters like ä, ö, ü, etc., you define this property accordingly.

  `lse-tpu:ident-chars` defines the characters that comprise a simple word.

`lse-tpu:ident-group-chars`
: This property defines characters like a hyphen which are used to combine simple words into a compound word. The default value is '`-_*@+`', which is rather lispy. For a natural language, use the value '`-`'. Many programming languages need the value '`_`'.

`lse_comment_delim_char_set`
: Defines all characters which may occur in — leading and trailing — comment delimiters. LS-Emacs uses this property to skip over empty comments and to identify lines consisting of an empty comment only. The default value is '`;`'.

`lse_comment_head_delim`
: Defines the string which is used to start a comment. LS-Emacs uses this property to insert comments into a buffer. If a language allows more than one set of comment delimiters — like `C++` — choose the one you want to use primarily. The default value is '`;`'.

`lse_comment_head_delim_pattern`
: Defines a regular expression which matches all possibilities of comment starters. LS-Emacs uses this property to find the start of comments. This property is often defined so that whitespace following the comment starter is included in the match. The default value is '`;+ *`'. See Section 4.7 [Comment-Expansion], page 50.

`lse_comment_tail_delim`
: Defines the string which is used to terminate a comment. LS-Emacs uses this property to insert comments into a buffer. If a language allows more than one set of comment delimiters — like `C++` — choose the one you want to use primarily. The default value is to have no comment terminator.

`lse_comment_tail_delim_pattern`
: Defines a regular expression which matches all possibilities of comment terminators. LS-Emacs uses this property to find the end of comments. This property is often defined so that whitespace preceding the comment terminator is included in the match. The default value is no terminator, i.e., comments are terminated by the end of the line. See Section 4.7 [Comment-Expansion], page 50.

4.4 Definition of Fill-Ins

The definitions of fill-ins are stored in template files. You define a fill-in by the command `lse-define-fill-in`. This section explains how to use this function.

lse-define-fill-in *fill-in-name* &rest *body* Function

This function is the primary work-horse for defining LS-Emacs templates. It (re-)defines a fill-in with name *fill-in-name* and definition *body*.

The *fill-in-name* is used to refer to the fill-in wherever it is needed. LS-Emacs is case-insensitive when dealing with names of languages, fill-ins and tokens. A *fill-in-name* can contain any printable character except the characters used by fill-in delimiters.[3] Specify a string inside double quotes as *fill-in-name* and try to choose a name which explains the meaning of the fill-in.

The *body* is a list of properties of the fill-in. These properties can be defined in any order. Each property is a cons cell; the first element is a symbol naming the property to be defined, the other elements specify the value of the property. A minimum definition

[3] That is one of the reasons why LS-Emacs normally uses 8-bit characters for delimiting fill-ins.

of a fill-in consists of just the description-property for a terminal fill-in or just the expansion-property for a non-terminal fill-in. Each and every fill-in definition should contain a description property unless it is **never** directly seen by the user of LS-Emacs.

There are five groups of fill-in properties:

- Description-properties provide information for interactive help about the fill-in.
- Expansion-properties define the type, and control expansion, of the fill-in.
- Killing-properties control the behavior of `lse-kill-fill-in`.
- Replacement-properties specify how LS-Emacs behaves before and after — manual or automatic — replacement of the fill-in.
- Token-properties define tokens associated with the fill-in.

The following subsections will detail the properties you can define.

4.4.1 Description-Properties

Description-properties provide interactive help for the fill-in. A description property is a list starting with the symbol `description` followed by strings. The second element of the list is displayed by the command `lse-describe-fill-in` and in menus. The other elements of the list are displayed by `lse-help-fill-in` separated from each other by new-lines.

4.4.2 Expansion-Properties

Expansion-properties define the type of the fill-in and control expansion. The definition of a terminal fill-in contains no expansion-property, all other fill-in definitions contain exactly one expansion-property. The expansion-property is a list. The first element determines the type of the fill-in — replacement, menu, or function — (see Section 3.3.6 [Fill-In-Expansion], page 20), the rest of the expansion-property specifies the value of the expansion.

Expansion of a replacement or function fill-in directly leads to an action — LS-Emacs will interpret the value of the expansion-property. By way of contrast, expansion of a menu fill-in results in the first instance in a dialogue between the user and LS-Emacs; only after selection of one alternative will LS-Emacs do something — either perform the expansion of another fill-in or replace the menu fill-in by the text chosen by the user.

4.4.2.1 Replacement-Expansion-Property

Fill-ins with an expansion-property of the replacement type are the most common kind of fill-in. In this case, the expansion-property is a list of objects which specify the replacement text. On expansion, LS-Emacs inserts this replacement text into the buffer. When you define the expansion-property, you can use different types of objects: strings, lisp-symbols, special symbols like `@` and `&`, and valid lisp-forms. We will consider the use of these different types shortly.

When LS-Emacs expands an expansion-property of the replacement type, it iterates over the elements of the list defined as replacement. LS-Emacs will interpret each element in turn and take the appropriate action. Except after some special lisp-functions and the special symbol `&`, LS-Emacs will start the insertion of the value of each element on a new line at the proper level of indentation. You can change the indentation level by including the appropriate lisp-function in the definition of the expansion-property.

strings The most common type of replacement object is a string. LS-Emacs will simply insert that string into the buffer. For Emacs, a string is any sequence of characters enclosed in a pair of double quotes. You can include a double quote in a string by preceding it by a backslash.[4] Do not use new-lines inside a string — the result will disappoint both you and the user of your templates.

If the expansion-property contains two strings in a row, LS-Emacs will insert the first string into the buffer followed by a new-line and the appropriate indentation followed by the second string. To avoid the new-line, use the special symbol `&`.

lisp-symbols

When expanding the fill-in, LS-Emacs will first check if there is a function defined for the symbol. If so, LS-Emacs will call the function without parameters.[5] If there is no function defined, LS-Emacs will insert the symbol-value into the buffer if it is a string; otherwise an error will occur.

Normally, the function will either control the indentation or insert some text into the buffer, e.g., the current date, the name of the buffer, or maybe the name of the user. The function should not change the position of the point — otherwise the next element of the replacement will be inserted at the wrong place.

LS-Emacs provides a number of functions for controlling indentation.[6] See Section 4.6 [Indentation], page 49.

`delete-horizontal-space`
: Deletes all white space around the point.

`fixup-whitespace`
: Replaces white space between the objects at either side of the point with either one space or no space as appropriate.

`just-one-space`
: Deletes white space around the point except for one space.

---

[4] A backslash character in a string has special meaning for LS-Emacs. To include a backslash in a string you have to use two backslashes in a row.

[5] To use a function which needs parameters you have to use a lisp-form.

[6] Use of these functions will inhibit the new-line(s) which LS-Emacs normally inserts between elements of the expansion-property.

`lse-environment-indent`
: Sets the indentation level of the current and the following lines to that of the line containing the flat fill-in LS-Emacs currently expands.

`lse-expansion-indent`
: Sets the indentation level of the current and the following lines to the position of the flat fill-in LS-Emacs currently expands.

`lse-indent+1`
: Increase the indentation level of the current and the following lines by `lse-language:tab-increment` compared to the previous line.

`lse-indent-1`
: Decrease the indentation level of the current and the following lines by `lse-language:tab-increment` compared to the previous line.

`lse-newline`
: Insert a new-line without indentation.

`lse-newline-and-indent`
: Insert a new-line and indent to the current indentation level.

`lse-no-indent`
: Remove indentation from the current line and the following lines.

`lse-outer-environment-indent`
: Corresponds to the sequence `lse-environment-indent` followed by `lse-indent-1`.

`lse-tabulator`
: Aligns to the next alignment position of the previous line, or inserts `lse-language:tab-increment` spaces if there is none.

`@` Instructs LS-Emacs to interpret the next element of the expansion-property as name of a fill-in to be expanded as part of the current fill-in. You can specify a string or a symbol for this name.

When expanding the fill-in, LS-Emacs will look-up the expansion-property of the fill-in named by the `@` symbol and expand it. This corresponds to a subroutine-call and allows the sharing of common expansion-properties between any number of fill-ins. Whenever you notice commonality in the expansion-properties of two fill-ins you should consider sharing by means of the `@` symbol!

`&` This symbol inhibits the implicit new-line between the previous and next element of the expansion-property. You need this symbol when you want to combine a literal string and a lisp-function in a single line.

lisp-forms You can specify any valid lisp-form in parentheses. LS-Emacs will evaluate the form by means of `eval`. Use lisp-forms when you want to call a lisp-function which needs

parameters. Normally the form will insert text into the buffer or control the indentation. It should not change the position of the point.

One use of lisp-forms are calls to the function `lse-indent`. The argument specifies the number of indentation levels by which the indentation of the current and the following lines are changed. `(lse-indent -1)` corresponds to a call of `lse-indent-1`.

4.4.2.2 Menu-Expansion-Property

Menu fill-ins allow the user to select one of a number of alternative expansions. The expansion-property is a list of menu-entries. For each menu-entry you can use different types of objects: strings, symbols, lists comprising two strings, and the special symbol `@`. Unless you specify the fill-in property `'(sort)`, LS-Emacs will display the menu-entries in the sequence given by the expansion-property. Using the `sort` property is particularly useful for menus containing the special symbol `@`.

symbols LS-Emacs interprets a menu-entry of type symbol as reference to another fill-in. If the user selects this entry, LS-Emacs will expand that fill-in according to its type. While it makes no sense to refer to a terminal fill-in, any type of non-terminal fill-in — replacement, menu, or function — will be fine. The referenced fill-in should provide a description property that can be used as description of the entry in the menu.

lists A menu-entry of type list must contain two strings. When the user selects the entry, LS-Emacs uses the first string as replacement. The second string is used as description in the menu. Normally, this type should be used rather than a string without description.

strings A menu-entry of type string specifies the replacement to be used when the user selects the entry. In most cases, it will be better to use a list containing both the replacement and a description. You should use strings only in trivial cases.

`@` Instructs LS-Emacs to interpret the next menu-entry as name of another menu-fill-in. You can specify a string or a symbol for this name. LS-Emacs will include the menu-entries of the other fill-in in the menu presented to the user. This corresponds to a macro-expansion. Use `@` when two or more of menu-fill-ins share a number of menu-entries.

You should consider using the `sort`-property when using `@` in a menu.

4.4.2.3 Function-Expansion-Property

Fill-ins with an expansion-property of type function are the least common kind of fill-in. The expansion-property is a list of lisp-functions and lisp-forms.

When expanding a function fill-in, LS-Emacs does not remove the flat state of the fill-in. Therefore, no element of the expansion-property should insert text into the buffer — at least not at the current position, which is inside the flat fill-in. Function fill-ins are useful to display information

provided by a lisp-function, to manipulate the buffer in some way, e.g., call a compiler or another utility to process (part of) the buffer's contents, or to switch to another buffer related to the first one, e.g., from the header file of a `C` module to the source file or vice versa.

With function fill-ins you can ease the cognitive burden of having to remember too many function-names for the user.

4.4.3 Killing-Properties

Killing-Properties control how much text is removed from the buffer by `lse-kill-fill-in` and allow the specification of an action to be performed after killing.

In any case, `lse-kill-fill-in` removes the flat fill-in plus adjacent white space. By using the properties `leading` and `trailer` you can force LS-Emacs to remove additional text. This is for instance useful to remove the separator between duplicates when the user kills the duplicate.

Each of the killing-properties `leading` and `trailer` is a list of two elements, the first naming the property — either `leading` or `trailer` — the second a string specifying the text to remove (without whitespace!). If you use dotted-pair notation for the list, e.g., `'(leading . "<text>")`, the string will be interpreted as a regular expression. If you use normal list notation, e.g., `'(trailer "<text>")`, the string will be interpreted as a literal string.[7]

If you specify the replacement-property `separator`, you should also specify a corresponding killing-property for `leading` (but without whitespace!).

> **Note for DEC LSE Users:** `leading` and `trailing` are much more important in DEC LSE than in LS-Emacs. This is due to the fact that LS-Emacs provides the possibility to define leading and trailing text added automatically to each replacement of a fill-in (properties `replacement-leading` and `replacement-trailer`). You should rely on these replacement-properties rather than on the killing-properties wherever possible.

4.4.3.1 kill-action

This property specifies an action to be performed after the fill-in was killed. The point is at the position of the killed fill-in. One use of this property allows to insert indentation in place of the killed fill-in — this is handy for table-editing. Another use could be the automatic killing of flat fill-ins down in the buffer by using the LS-Emacs function `lse-kill-future-fill-in`.

This property is a list starting with the symbol `kill-action` followed by the body of the kill-action. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21). Normally, the elements of the body will be lisp-functions or lisp-forms, but you can also specify literal text to be inserted into the buffer.

---

[7] A regular expression is much more powerful, but you have to take care to quote characters which have special meaning in regular expressions.

4.4.4 Replacement-Properties

Replacement-Properties control the behavior of LS-Emacs before and after the replacement of a fill-in which may have been caused by either `lse-replace-fill-in` (for any kind of replaceable fill-in) or by `lse-expand` (for a fill-in of type replacement).

LS-Emacs looks at the following properties before it starts replacement: `replacement-leading`, `replacement-trailer`, `replacement-vanguard`, and `separator`.

LS-Emacs looks at the following properties after replacement: `auto-replicate`, `max-line-move`, `no-history`, `rcompletion-action`, `rcompletion-leading`, and `rcompletion-trailer`.

4.4.4.1 auto-replicate

This replacement-property specifies how many following instances of the fill-in should be replaced using the deep-state contents of the current fill-in.

After finishing replacement of the fill-in, LS-Emacs takes the value of `auto-replicate` and tries to find as many fill-ins with the same name between the point and the end of the buffer. Each of those found is replicated using the value of the replacement just finished.

Use this replacement-property wherever a document requires exactly the same value in more than one place when that value can be determined only at run-time. A typical example for the use of `auto-replicate` is provided by programming languages like `Ada` which recommend that the `END` statement of a sub-program be followed by its name.[8]

To define this replacement-property use a list with two elements `'(auto-replicate <number>)`, where `<number>` specifies how many instances are to be replicated automatically. If you don't define the property, LS-Emacs will not auto-replicate the fill-in.

4.4.4.2 max-line-move

After the successful expansion of a fill-in, LS-Emacs tries to position the point on the next flat fill-in. This replacement-property specifies how far LS-Emacs looks down in the buffer. It is a list with two elements `'(max-line-move <number>)`, where number specifies the number of lines to be looked at. If you don't define this property, LS-Emacs will use a default value of one, i.e., look only at the current line. If you want the point to stay at the end of the expansion specify a value of zero for `<number>`.

4.4.4.3 no-history

If you specify this property, LS-Emacs won't record the expansion on the expansion-history, i.e., no un-expansion will be possible.

[8] I would strongly recommend to use this practice for all programming languages!

4.4.4.4 rcompletion-action

This property is similar to `rcompletion-leading` and `rcompletion-trailer` and specifies an action to be performed after successful completion of a replacement. The point is at the position where the replacement was finished when the completion-action is performed.[9]

This property is a list starting with the symbol `rcompletion-action` followed by the body of the completion-action. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21). Normally, the elements of the body will be lisp-functions or lisp-forms, but you can also specify literal text to be inserted into the buffer.

4.4.4.5 rcompletion-leading

This property is similar to `rcompletion-action` and `rcompletion-trailer` and specifies an action to be performed after successful completion of a replacement. The point is at the position where replacement started when this property is interpreted.

This property is a list starting with the symbol `rcompletion-leading` followed by the body of the completion-action. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21). Normally, the elements of the body will be lisp-functions or lisp-forms, but you can also specify literal text to be inserted into the buffer. The conventional use for this property is to adjust indentation in front of the replacement.

4.4.4.6 rcompletion-trailer

This property is similar to `rcompletion-action` and `rcompletion-leading` and specifies an action to be performed after successful completion of a replacement. The point is at the end of the range of the replacement when this property is interpreted.

This property is a list starting with the symbol `rcompletion-trailer` followed by the body of the completion-action. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21). Normally, the elements of the body will be lisp-functions or lisp-forms, but you can also specify literal text to be inserted into the buffer. The conventional use for this property is to adjust indentation behind the replacement.

> **Note:** Normally, the positions where `rcompletion-trailer` and `rcompletion-action` are interpreted are equal. Yet, this is not necessarily always the case. If the replacement contains function calls, the positions may well differ. So you have to think carefully about the effect to be achieved before you decide which property to use.

[9] The difference between `rcompletion-action`, `rcompletion-leading`, and `rcompletion-trailer` is the position of the point during interpretation of the corresponding property by LS-Emacs.

4.4.4.7 replacement-leading

This property specifies text to be automatically inserted in front of the replacement, however it is started. That is done before the replacement is begun.

This property is a list starting with the symbol `replacement-leading` followed by the body of the replacement-leading. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21).

A typical use for this property are fill-ins which need to be enclosed in quotes or brackets. Do not use this property to specify leading indentation, use `rcompletion-leading` instead (the replacement itself may change the indentation again, which is probably not what you intended).

4.4.4.8 replacement-trailer

This property specifies text to be automatically inserted behind the replacement, however it is started. That is done before the replacement is begun.

This property is a list starting with the symbol `replacement-trailer` followed by the body of the replacement-trailer. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21).

A typical use for this property are fill-ins which need to be enclosed in quotes or brackets. Do not use this property to specify trailing indentation, use `rcompletion-trailer` instead (the replacement itself will change the indentation again, which is certainly not what you intended).

4.4.4.9 replacement-vanguard

This property is similar to `replacement-leading` and `replacement-trailer`, but the difference in its handling is subtle. In contrast to the latter properties, `replacement-vanguard` specifies an action to be performed before removing the flat fill-in. When the value of `replacement-vanguard` is expanded, the position of the point is just before the leading delimiter of the flat fill-in.

Use this property for defining things like leading indentation, which should not be considered to be part of the replacement — the result of the expansion of `replacement-leading` and friends is considered to be part of the replacement.

4.4.4.10 separator

This property defines how a duplicate will be separated from its original. By default, LS-Emacs uses the fill-in `$$default$$separator` as separator (the default value of this fill-in is `lse-newline-and-indent`).

This property is a list starting with the symbol `separator` followed by the body of the separator. The semantics of the body is the same as for an expansion-property of type replacement (see Section 3.3.6.1 [Replacement-Expansion], page 21). If you specify a value for this property

which expands into more than pure whitespace, you have to define the killing-property `leading` appropriately.

You can change the default of the separator by redefining `$$default$$separator`, but it should expand to whitespace only (otherwise you would have to implement a corresponding default for `leading`!).

4.4.5 Token-Properties

A token-property defines a token associated to the fill-in. In contrast to all other properties, you can define as many token-properties as you think necessary. Quite a number of fill-ins are associated with more than one token.

A token-property is a list with the first element `token`. The second element is an optional string and specifies the name of the token. If no name is given the token-name is equal to the name of the fill-in.

4.5 Definition of Tokens

The large majority of tokens is associated to fill-ins, so-called fill-in-tokens. Normally, they are defined alongside the associated fill-in. Yet, sometimes you may want to define a fill-in-token independently from the fill-in definition. This chapter explains how to do that and how to define so-called simple tokens, which are not associated to a fill-in at all.

lse-define-fill-in-token *token-name fill-in* Function

This function defines a token named *token-name* for the *fill-in*. *fill-in* can be a string naming the fill-in or the lisp-symbol defined for the fill-in. For explicit calls you will probably prefer to pass a string for *fill-in*.

An explicit call to this function is equivalent to using the token-property inside the fill-in concerned with the parameter *token-name*. Unless you have good reasons not to, you should always define the token with the token-property rather than explicitly.

lse-define-simple-token *token-name expansion* Function

This function defines a simple token with the name *token-name*. The expansion is either a string or a cons cell which can be evaluated. On expansion, LS-Emacs inserts the string into the buffer or evaluates the cons cell — whichever is bound to the token.

Simple tokens are useful to provide an easy way to include the 8-bit characters used as fill-in delimiters into the buffer (see '`lse-templates-generic.lse`'). The expansion of a simple token should be very simple, otherwise you'd better define a fill-in with an associated token.

4.6 Indentation

This chapter explains how LS-Emacs manages indentation.

First, we will look at the basics. Consider a flat fill-in somewhere in a line. When you instruct LS-Emacs to expand that fill-in, there are two particularly important columns:

`environment-indent`
: This is the position of the first non-blank character in the line.

`expansion-indent`
: This is the start position of the flat fill-in to be expanded.

If the fill-in is the first element of the line, then these two columns coincide.

If you don't use functions changing the indentation, LS-Emacs will indent all lines of the expansion to the column given by `expansion-indent`. The expansion of many fill-ins contains lines which should be further indented by one indentation-level.[10] **Do not ever use explicit blanks to cause that indentation!** Instead, enclose the lines to be further indented inside a pair of calls to `lse-indent+1` and `lse-indent-1`!

Sometimes, you will want to indent part of the expansion of a fill-in relative to the `environment-indent`. The function `lse-environment-indent` lets you do just that. To return to an indentation relative to the position of the expanded fill-in, use `lse-expansion-indent`. A typical example for the use of `lse-environment-indent` is the definition of a record-type in a programming language like `Pascal` or `Ada`. Let's consider the situation in `Ada`: The type-definition starts with something of the form

```
type foo is <type-definition>;
```

If you expand `<type-definition>`, LS-Emacs will display a menu containing the menu-entry `record-type-definition` as one alternative. If you had defined the fill-in `record-type-definition` without using `lse-environment-indent`, the expansion would be indented like this:

```
type foo is record
                <record-component>···;
            end record;
```

When you use `lse-environment-indent` for the definition of `record-type-definition`, the expansion looks much better and does not waste valuable space:

```
type foo is record
    <record-component>···;
end record;
```

For a complete list of LS-Emacs functions controlling indentation, see Section 4.4.2.1 [Replacement-Expansion-Property], page 40.

[10] The depth of an indentation-level is defined by the language property `lse-language:tab-increment`.

4.7 Expansion of Comments

If a flat fill-in is located inside a comment, LS-Emacs will put the entire replacement inside a comment. This means, that for each new-line started, LS-Emacs will first add a comment terminator to the previous line and then insert a comment starter at the new line. It will also use the correct indentation. LS-Emacs will use the language properties `lse_comment_head_delim_pattern` and `lse_comment_tail_delim_pattern` to determine the comment delimiters used for starting and terminating comments according to the context.

If you want to define fill-ins for things like the header-comment of a document, you should define the fill-in itself without comment delimiters and use it in flat form inside a comment. The expansion will provide all comment delimiters necessary. This allows the use of the same fill-in for languages with different comment delimiters and offers therefore all benefits of reuse. A typical example is the definition of the fill-in `header-comment` in the template file '`lse-templates-generic.lse`'.

5 Programming with LS-Emacs

This chapter is planned to explain how to use LS-Emacs lisp functions for programming in Emacs lisp. Until it is written[1], you have to explore the elisp source code to do that.[2]

The lisp source of LS-Emacs comprises a number of Emacs libraries. The names of these all start with '`lse-`', except for the root file which is named '`ls-emacs.el`'. The best starting point for exploration is probably '`lse-interactive.el`' which defines most of the interactive, language-sensitive commands provided by LS-Emacs. Other files of interest are:

- '`lse-flat-fill-in.el`'
- '`lse-define.el`'
- '`lse-indent.el`'
- '`lse-language.el`'
- '`lse-tpu.el`'
- '`lse-editing.el`'

[1] Any volunteers?

[2] Did you ever play Adventure?

6 Reference Manual for Interactive Commands

This chapter explains all interactive commands provided by LS-Emacs.[1]

lse-align-and-down Command

This command inserts or removes as many spaces as necessary to align to the corresponding position of the same character as the current one on the previous line. Afterwards, it moves point one line down.

By default, this command is bound to the key `blue /`.

lse-align-and-up Command

This command inserts or removes as many spaces as necessary to align to the corresponding position of the same character as the current one on the next line. Afterwards, it moves point one line up.

By default, this command is bound to the key `gold /`.

lse-align-to-next-word *num* Command

This command inserts as many spaces as necessary to align to the position of the corresponding word of the next line, i.e. in forward direction. The prefix argument *num* specifies how many lines to move for finding the alignment position.

By default, this command is bound to the key `gold SPC`.

lse-align-to-next-word-and-up *num* Command

This command inserts as many spaces as necessary to align to the position of the corresponding word of the next line, i.e. in forward direction; after aligning, `lse-align-to-next-word-and-up` moves point one line up. The prefix argument *num* specifies how many lines to move for finding the alignment position.

By default, this command is bound to the key `gold v`.

lse-align-to-previous-word *num* Command

This command inserts as many spaces as necessary to align to the position of the corresponding word of the previous line, i.e. in reverse direction. The prefix argument *num* specifies how many lines to move for finding the alignment position.

By default, this command is bound to the key `blue SPC`.

[1] Alas, it describes an ancient version of LS-Emacs (Summer 1994). I didn't have enough time to update this chapter yet. Sorry.

lse-align-to-previous-word-and-down *num* Command

This command inserts as many spaces as necessary to align to the position of the corresponding word of the previous line, i.e. in reverse direction; after aligning, `lse-align-to-next-word-and-up` moves point one line down. The prefix argument *num* specifies how many lines to move for finding the alignment position.

By default, this command is bound to the key `blue v`.

lse-balance-windows Command

This command resizes all windows so that all have roughly the same height.

lse-blink-select-mark Command

This command positions point for 2 seconds on the other boundary of the selection.

By default, this command is bound to the key `blue gold enter`.

lse-change-output-file *buf* &optional *new-name silent* Command

This command changes the name of the output file associated to buffer *buf* to the name *new-name*. Unless *silent* is specified, the command will display an informational message in the message window.

By default, this command is bound to the key `blue o`.

lse-command:do &optional *command* Command

This command pops up a completion menu with the most commonly used commands. You can add commands to the menu with the function `lse-command:add`.

By default, this command is bound to the key `gold kp7`.

lse-compilation:goto-last-mark Command

This command returns to the position prior to the last execution of `lse-compilation:next-error`. It comes in handy if there were no more errors left, in which case Emacs switches to the compilation buffer.

In Emacs compilation buffers, this command is bound to the key `gold b`.

lse-compilation:next-error &optional *prefix* Command

This command positions to the next compiler error or grep match. You can use it after compiling a buffer with the command `lse-compile` (or the standard Emacs command `compile`) or after searching with the command `lse-grep` (or its Emacs analogon `grep`).

Strange things happen, if you issue this command when the last error or grep match was already visited.

By default, this command is bound to the key `A-f`.

lse-compile Command

This command compiles the current buffer. The effect depends on the kind of buffer.

- If the buffer contains LS-Emacs template definitions `lse-compile` calls `lse-language:compile`.

  LS-Emacs looks at the extension of the buffer to decide if it contains LS-Emacs templates.
- If the buffer uses the major mode Emacs-Lisp `lse-compile` calls `eval-current-buffer` which evaluates the buffer as lisp code.
- For all other kinds of buffers `lse-compile` calls `compile` with the variable *compile-command* as parameter.

By default, this command is bound to the key `blue y`.

lse-compile-defun Command

This command evaluates the lisp function containing or before point and prints the value in the echo area. It is equivalent to executing the Emacs commands `beginning-of-defun` and `eval-defun` in sequence (inside a `save-excursion` block).

By default, this command is bound to the key `red y`.

lse-count-matches Command

This command counts the number of matches of a regular expression in the entire current buffer.

By default, this command is bound to the key `blue #`.

lse-delete-other-windows &optional *wdw* Command

This command deletes all windows but the current window. The optional argument *wdw* instructs the command to all windows but *wdw*.

By default, this command is bound to the key `blue gold -`.

lse-delete-window &optional *wdw* Command

This command deletes the current window. The optional argument *wdw* instructs the command to delete another than the current window.

By default, this command is bound to the key `blue gold =`.

lse-describe-fill-in Command

This command displays a description of the current fill-in in the message window. Point must be inside a flat fill-in.

By default, this command is bound to the key `A-o`.

lse-expand Command

If point is inside a flat fill-in, `lse-expand` will expand that fill-in. Otherwise, this command will look for a token and expands that token if it finds one. `lse-expand` will look at the words preceding point to find a token and always will try to find the longest match.

A token-name cannot cross line boundaries, and `lse-expand` will consider at most 5 words when looking for the token-name.

By default, this command is bound to the key `A-e`.

lse-expand-or-goto-next Command

This command expands the current fill-in, if any, or else it moves point to the next flat fill-in.

lse-expand-or-tabulator Command

This command expands the current fill-in, if any, or else it calls `lse-tabulator`.

By default, this command is bound to the key `TAB`.

lse-expand-token &optional *quiet* Command

Expands the token preceding point. If `lse-expand-token` does not find a valid token-name, it will display a warning message, unless the optional argument *quiet* has a non-nil value. You can use a prefix argument for *quiet*.

This command is called by `lse-expand`.

lse-fill-range Command

This command fills the current selection or the current paragraph, if no selection is active.

By default, this command is bound to the keys `gold |` and `blue gold ~`.

lse-flush-replacement Command

This commands terminates a pending replacement. Sometimes, LS-Emacs becomes confused when a replacement is pending. This happens for instance, when you delete the range of the pending replacement by normal deletion commands. This confusion may result in a hanging of the editor. If that happens, use `C-g` to regain control and use `lse-flush-replacement` to exorcize any trace of the pending replacement.

By default, this command is bound to the key `blue gold A-r`.

lse-goto-buffer *buf may-create temporary no-mark default* Command

This command switches to buffer *buf*. If *may-create* is supplied, a new buffer is created, if it does not already exist. *temporary* and *no-mark* have the same meaning as for `lse-goto-buffer+maybe-create`.

By default, this command is bound to the key `gold f`.

lse-goto-buffer+maybe-create &optional *buf temporary no-mark* Command

This command switches to the buffer *buf*. It that buffer does not exist, it will be created. If *temporary* is supplied, the new buffer won't be considered to be a user buffer. If *no-mark* is supplied, the command won't change the mark-stacks.

By default, this command is bound to the key `blue f`.

lse-goto-buffer-other-window &optional *buf may-create temporary no-mark default* Command

This command switches to buffer *buf* in another window. If *may-create* is supplied, a new buffer is created, if it does not already exist. *temporary* and *no-mark* have the same meaning as for `lse-goto-buffer+maybe-create`.

By default, this command is bound to the key `blue gold f`.

lse-goto-home-mark-buffer Command

This command switches to the buffer and position specified by the home-mark of the mark-stack of the current buffer. The position before the command will be stacked on the appropriate mark-stacks.

lse-goto-home-mark-global Command

This command switches to the buffer and position specified by the home-mark of the global mark-stack. The position before the command will be stacked on the appropriate mark-stacks.

By default, this command is bound to the key `gold h`.

lse-goto-home-mark-window Command

This command switches to the buffer and position specified by the home-mark of the mark-stack of the current window. The position before the command will be stacked on the appropriate mark-stacks.

By default, this command is bound to the key `blue h`.

lse-goto-last-mark-buffer Command

This command switches to the buffer and position specified by the last-mark of the mark-stack of the current buffer. The position before the command will be stacked on the appropriate mark-stacks.

By default, this command is bound to the key `blue gold b`.

lse-goto-last-mark-global Command

This command switches to the buffer and position specified by the last-mark of the global mark-stack. The position before the command will be stacked on the appropriate mark-stacks.

By default, this command is bound to the key `gold b`.

lse-goto-last-mark-window Command

This command switches to the buffer and position specified by the last-mark of the mark-stack of the current window. The position before the command will be stacked on the appropriate mark-stacks.

By default, this command is bound to the key `blue b`.

lse-goto-last-position Command

This commands positions point to the position before the last execution of either `lse-goto-next-fill-in` or `lse-goto-prev-fill-in`.

By default, this command is bound to the keys `gold A-n` and `gold A-p`.

lse-goto-mark-and-pop-buffer Command

This command removes the top-mark from the mark-stack of the current buffer and positions point to that mark.

By default, this command is bound to the key `blue gold g`.

lse-goto-mark-and-pop-global Command

This command removes the top-mark from the global mark-stack and positions point to that mark.

By default, this command is bound to the key `gold g`.

lse-goto-mark-and-pop-window Command

This command removes the top-mark from the mark-stack of the current window and positions point to that mark.

By default, this command is bound to the key `blue g`.

lse-goto-next-buffer Command

This command switches to the next (younger) user buffer. LS-Emacs orders the user buffers by age. This command does not change the mark-stacks.

By default, this command is bound to the key `gold n`.

lse-goto-next-fill-in &optional *quiet name* Command

This command moves point to the next flat fill-in in forward direction. It displays a warning message if none is found unless *quiet* has a non-nil value. You can use a prefix argument for *quiet*.

If *name* is specified — this can be done only by a call from a lisp function — `lse-goto-next-fill-in` will move position to the next occurrence of the flat fill-in *name*, if any.

By default, this command is bound to the key `A-n`.

lse-goto-prev-buffer Command

This command switches to the previous (older) user buffer. LS-Emacs orders the user buffers by age. This command does not change the mark-stacks.

lse-goto-prev-fill-in &optional *quiet name* Command

This command moves point to the previous fill-in in backward direction, if any. It displays a warning message if none is found unless *quiet* has a non-nil value. You can use a prefix argument for *quiet*.

If *name* is specified — this can be done only by a call from a lisp function — `lse-goto-prev-fill-in` will move position to the previous occurrence of the flat fill-in *name*, if any.

By default, this command is bound to the key `A-p`.

lse-grep *grep-args* Command

This command runs agrep with the arguments grep-args. If used interactively, it will prompt for *grep-args*. You can visit all matches found by grep by executing the command `lse-compilation:next-error`.

The default for the command is taken from the variable *lse-compilation:grep-command*. You can change this variable in your '`.emacs`' file if you want to use another version of grep.

By default, this command is bound to the key `blue gold find`.

lse-help-fill-in Command

This command displays a description of the current fill-in in a popped-up window. Point must be inside a flat fill-in.

You can remove the popped-up window by using the key `BLUE A-f`.

By default, this command is bound to the key `gold A-o`.

lse-indent-line Command

This command indents the current line — no matter where in the line point is located — corresponding to the indentation of the previous line. If point is currently located on any kind of closing parenthesis (as defined by the syntax table), `lse-indent-line` will indent to the indentation level of the corresponding opening parenthesis.

By default, this command is bound to the key `gold TAB`.

lse-indent-rigidly &optional *amount* Command

This command indents the current selection or the current line rigidly by *amount* blanks (default 2).

By default, this command is bound (with different arguments) to the keys `gold left`, `gold right`, `blue left`, and `blue right`.

lse-insert-buffer &optional *buffer* Command

This command inserts the contents of the buffer named *buffer* into the current buffer. In contrast to the corresponding Emacs command, no mark is set.

By default, this command is bound to the key `blue p`.

lse-insert-buffer-name &optional *buf* Command

This command inserts the name of buffer *buf* into the current buffer.

lse-insert-dd-mm-yyyy Command

This command inserts the current date formatted as `dd-mm-yyyy`, e.g., 1.7.1994, into the buffer.

lse-insert-dd-mm-yyyy+blank Command

This command inserts the current date formatted as `dd-mm-yyyy`, e.g., 1.7.1994, into the buffer. It will add a blank unless the next character already is a blank.

By default, this command is bound to the key `blue d`.

lse-insert-dd-mmm-yyyy Command

This command inserts the current date formatted as `dd-mmm-yyyy`, e.g., 1-Jul-1994, into the buffer.

lse-insert-dd-mmm-yyyy+blank Command

This command inserts the current date formatted as `dd-mmm-yyyy`, e.g., 1-Jul-1994, into the buffer. It will add a blank unless the next character already is a blank.

By default, this command is bound to the key `gold d`.

lse-insert-file &optional *file* Command

This command inserts the contents of file *file* at the current position of the current buffer.

By default, this command is bound to the key `blue gold i`.

lse-insert-time Command

This command inserts the current date and time into the current buffer.

lse-insert-time+blank Command

This command inserts the current date and time into the current buffer. It will add a blank unless the next character already is a blank.

lse-insert-user-full-name Command

This command inserts the full name of the user into the current buffer.

By default, this command is bound to the key `blue z`.

lse-insert-user-full-name+blank Command

This command inserts the full name of the user into the current buffer. It will add a blank unless the next character already is a blank.

lse-insert-user-initials Command

This command inserts the initials of the user into the current buffer.

By default, this command is bound to the key `gold z`.

lse-insert-user-initials-tex Command

This command inserts the initials of the user as TEX-command into the current buffer.

By default, this command is bound to the key `gold z`.

lse-insert-user-initials+blank Command

This command inserts the initials of the user into the current buffer. It will add a blank unless the next character already is a blank.

lse-insert-user-initials-tex+blank Command

This command inserts the initials of the user as TEX-command into the current buffer. It will add a blank unless the next character already is a blank.

lse-insert-user-name Command

This command inserts the user name, i.e., your login name, into the current buffer.

By default, this command is bound to the key `blue gold z`.

lse-insert-user-name+blank Command

This command inserts the user name, i.e., your login name, into the current buffer. It will add a blank unless the next character already is a blank.

lse-insert-year Command

This command inserts the current year into the current buffer.

lse-kill-all-optional-fill-ins Command

This command kills all optional flat fill-ins between point and the end of the buffer. Required fill-ins are not affected.

By default, this command is bound to the key `blue gold A-k`.

lse-kill-buffer &optional *buf* Command

This command deletes the buffer *buf* (default: current buffer). Emacs will ask for confirmation if the buffer was changed since the last saving to a file.

By default, this command is bound to the key `blue gold e`.

lse-kill-fill-in &optional *dont-move* Command

This command kills the current or the next fill-in in the buffer. If that fill-in is required, `lse-kill-fill-in` will ask for confirmation. If the fill-in is non-replaceable, an error will occur.

Afterwards it will position point to the next fill-in unless *dont-move* has a non-nil value. You can use a prefix argument for *dont-move*.

By default, this command is bound to the key `A-k`.

lse-language:compile &optional *name* Command

This command compiles the LS-Emacs language named *name*.

`lse-language:compile` will prompt for *name*, if none was provided in the call. The prompting is done in two stages. First, `lse-language:compile` offers a completion mode, where you can select any of the pre-loaded languages. If you exit the completion mode without selection, lse-language:compile will then prompt without completion, which allows you to compile a language not yet known to LS-Emacs.

lse-language:reload &optional *name* Command

This command reloads the LS-Emacs language named *name*. You can use this command to load a new version of a language without leaving LS-Emacs and starting it afresh.

`lse-language:reload` uses the same completion mode for *name* as `lse-language:compile`.

lse-language:use &optional *name* Command

This commands selects a specific LS-Emacs language for the current buffer. Normally LS-Emacs will select the language automatically, but you can change the language used whenever you want.

`lse-language:use` uses the same completion mode for *name* as `lse-language:compile`.

lse-learn-key *key* Command

This command starts the learning of a keyboard macro to be associated to the key *key*. From now on, all LS-Emacs will record all keystrokes you type, whether they are commands or literal text. To end the definition, type the key *key* once again. While the recording is active, `Learn-'`*key*`'` will appear in the status line. Any error during the recording will immediately terminate the definition.

By default, this command is bound to the key `blue l`.

lse-learn-named-key *key name* Command

This command starts the learning of a named keyboard macro to be associated to the key *key*. From now on, all LS-Emacs will record all keystrokes you type, whether they are commands or literal text. To end the definition, type the key *key* once again. While the recording is active, `Learn-'`*key*`'` will appear in the status line. Any error during the recording will immediately terminate the definition.

By default, this command is bound to the key `blue gold l`.

lse-next-screen &optional *arg* Command

This command scrolls in forward direction, i.e., moves point down by *arg* scroll-units. One scroll-unit equals twenty percent of the current window's height.

By default, this command is bound with different arguments to the keys `next` and `gold next`.

lse-next-window Command

This command selects the next window below the current window on the screen. If the current window is at the bottom of the screen, `lse-next-window` will select the top-most window.

By default, this command is bound to the key `gold down`.

lse-open-line Command

This command opens a new line above the current line. If point is inside a fill-in, no replacement will be started and point will remain in the fill-in.

By default, this command is bound to the key `gold kp0`.

lse-pop+restore-window-configuration Command

This command restores the window configuration which is on the top of LS-Emacs window configuration stack.

By default, this command is bound to the key `blue gold ^`.

lse-previous-screen &optional *arg* Command

This command scrolls in reverse direction, i.e., moves point up by *arg* scroll-units. One scroll-unit equals twenty percent of the current window's height.

By default, this command is bound with different arguments to the keys `prior` and `gold prior`.

lse-previous-window Command

This command selects the next window above the current window on the screen. If the current window is at the top of the screen, `lse-previous-window` will select the bottom-most window.

By default, this command is bound to the key `gold up`.

lse-push-mark-buffer Command

This command pushes the current position of point onto the mark-stack of the current buffer.

By default, this command is bound to the key `blue gold m`.

lse-push-mark-global Command

This command pushes the current position of point onto the global mark-stack.

By default, this command is bound to the key `gold m`.

lse-push-mark-window Command

This command pushes the current position of point onto the mark-stack of the current window.

By default, this command is bound to the key `blue m`.

lse-push-window-configuration Command

This command pushes the current window configuration onto a stack of window configurations. You can later restore that configuration by using the command `lse-pop+restore-window-configuration`.

By default, this command is bound to the key `blue ^`.

lse-reexpand-fill-in Command

This command re-expands the last un-expanded fill-in, if any, i.e., it undoes the effect of the last `lse-unexpand-fill-in`.

By default, this command is bound to the key `blue A-e`.

lse-rename-buffer *buf* &optional *new-name* Command

This command renames the buffer *buf* to *new-name*.

lse-replace-fill-in Command

This command either closes the currently active replacement or — if point is inside a flat fill-in — starts the replacement of that fill-in.

By default, this command is bound to the key `A-r`.

lse-replicate-fill-in Command

If point is inside a flat fill-in, this command will replace the fill-in by the value of the last expansion of the same fill-in. If the fill-in was replicated and LS-Emacs is still in replacement mode, this command will replace the fill-in by the value of the next to last expansion of the same fill-in. By using this command repeatedly, you can try all the different previous expansions of the fill-in.

By default, this command is bound to the keys `A-s` and `gold A-s`.

lse-rereplace-fill-in Command

This command re-replaces the last un-replaced fill-in, if any, i.e., it undoes the effect of the last `lse-unreplace-fill-in`.

`lse-rereplace-fill-in` re-opens the replacement of the fill-in, i.e., you can extend the replacement by using this command.

By default, this command is bound to the key `blue A-r`.

lse-revert-buffer Command

This command reverts the current buffer to the contents of the associated file.

lse-ring-bell Command

This commands rings the terminal's bell.

By default, this command is bound to the key `blue blue`.

lse-scroll-other-window-back Command

This command scrolls the other window backward by a third of its height, i.e., moves point up.

By default, this command is bound to the key `blue prior`.

lse-scroll-other-window-forw Command

This command scrolls the other window forward by a third of its height, i.e., moves point down.

By default, this command is bound to the key `blue next`.

lse-select-angle-range Command

This command selects the next range which is enclosed in angle brackets (`<` ... `>`) and puts point to the end of the selection.

By default, this command is bound to the key `blue <`.

lse-select-brace-range Command

This command selects the next range which is enclosed in braces and puts point to the end of the selection.

By default, this command is bound to the key `blue {`.

lse-select-bracket-range Command

This command selects the next range which is enclosed in brackets and puts point to the end of the selection.

By default, this command is bound to the key `blue [`.

lse-select-current-bs-word *num* Command

This command selects the current (*num*) blank-separated word(s) and puts point to the end of the selection. If point is inside a word, that word is current; if it is between words, the previous word is considered current.

lse-select-current-word *num* Command

This command selects the current (*num*) word(s) and puts point to the end of the selection. If point is inside a word, that word is current; if it is between words, the previous word is considered current.

lse-select-guillemot-range Command

This command selects the next range which is enclosed in guillemots (< ... >)and puts point to the end of the selection.

By default, this command is bound to the key `blue gold <`.

lse-select-paren-range Command

This command selects the next range which is enclosed in parentheses and puts point to the end of the selection.

By default, this command is bound to the key `blue (`.

lse-set-buffer-nowrite *buf* Command

This command disables writing of the buffer *buf*. As Emacs won't let you change a buffer which is set to no-write, this command removes the association between the buffer and its output file. In addition, it inhibits auto-saving.

By default, this command is bound to the key `blue A-w`.

lse-set-buffer-write *buf* Command

This command enables writing of the buffer *buf*. In addition, it enables auto-saving.

By default, this command is bound to the key `blue gold A-w`.

lse-set-home-mark-buffer &optional *to-mark* Command

This command sets the home-mark of the current buffer's mark-stack to *to-mark* or to the current position of point.

lse-set-home-mark-global &optional *to-mark* Command

This command sets the home-mark of the global mark-stack to *to-mark* or to the current position of point.

lse-set-home-mark-window &optional *to-mark* Command

This command sets the home-mark of the current window's mark-stack to *to-mark* or to the current position of point.

By default, this command is bound to the key `blue gold h`.

lse-set-last-mark-buffer &optional *to-mark* Command

This command sets the last-mark of the current buffer's mark-stack to *to-mark* or to the current position of point.

lse-set-last-mark-global &optional *to-mark* Command

This command sets the last-mark of the global mark-stack to *to-mark* or to the current position of point.

lse-set-last-mark-window &optional *to-mark* Command

This command sets the last-mark of the current window's mark-stack to *to-mark* or to the current position of point.

lse-set-tab-increment *inc* Command

This command sets the value of tab increment to *inc*. The tab increment controls the depth of indentation per indentation level.

lse-shell-command *command* &optional *flag* Command

This command executes the unix command *command* in an shell. If you specify the prefix argument flag, LS-Emacs will insert the output from the shell into the current buffer.

By default, this command is bound to the key `blue x`.

lse-shift-select-mark *by* Command

This command shifts the position of the selection mark by *by* characters to the right.

lse-show-position Command

This command displays the current position in the message window: current line number, total line number, current column, current line length, position of point and buffer size in bytes.

By default, this command is bound to the key `blue kp7`.

lse-split-line Command

This command splits the current line. It handles comments properly, i.e., if point is inside a comment, `lse-split-line` will insert a comment terminator at the end of the current line, insert a new-line and insert a comment starter in the new line.

By default, this command is bound to the key `blue RET`.

lse-split-window &optional *wdw horizontally* Command

This command splits the current window into two and balances the sizes of all windows so that they are roughly equal. It leaves point in the lower or right window. The optional argument *wdw* instructs the command to split another than the current window. If a value for *horizontally* is given, `lse-split-window` will split the current window horizontally instead of vertically.

By default, this command is bound to the keys `gold =` and `blue =`.

lse-split-window-horizontally &optional *wdw* Command

This command splits the current window horizontally. It leaves point in the lower or right window. The optional argument *wdw* instructs the command to split another than the current window.

lse-tabulator Command

If point currently is at the beginning of a line, this command indents the line corresponding to the next non-blank line in forward direction; otherwise it calls `lse-align-to-previous-word`.

lse-toggle-mark-buffer Command

This command interchanges the current position and the position of the top-mark of the mark-stack of the current buffer.

By default, this command is bound to the key `blue gold t`.

lse-toggle-mark-global Command

This command interchanges the current position and the position of the top-mark of the global mark-stack.

By default, this command is bound to the key `gold t`.

lse-toggle-mark-window Command

This command interchanges the current position and the position of the top-mark of the mark-stack of the current window.

By default, this command is bound to the key `blue t`.

lse-tpu:add-at-bol *text* Command

This command adds *text* to the beginning of each line in the selected region or in the entire buffer, if no selection is active.

lse-tpu:add-at-eol *text* Command

This command adds *text* to the end of each line in the selected region or in the entire buffer, if no selection is active.

lse-tpu:advance-direction Command

Set direction to forward. This influences the behavior of many LS-Emacs commands.

By default, this command is bound to the key `kp4`.

lse-tpu:backup-direction Command

Set direction to reverse or backward. This influences the behavior of many LS-Emacs commands.

By default, this command is bound to the key `kp5`.

lse-tpu:backward-char *num* Command

This command moves *num* characters in the reverse direction (forward, if *num* is negative).

lse-tpu:backward-line *num* Command

This command moves to the beginning of the next line in reverse direction. The prefix argument *num* specifies how many lines to move.

lse-tpu:capitalize-strongly Command

This command capitalizes the selection or search-match, if any, or else the current word. Strongly means that this command first converts all characters to lower case before capitalizing the initial letters of the words in the range.

By default, this command is bound to the key `blue c`.

lse-tpu:capitalize-weakly Command

This command capitalizes the selection or search-match, if any, or else the current word. Weakly means that this command does not convert any characters to lower case before capitalizing the initial letters of the words in the range.

By default, this command is bound to the key `gold c`.

lse-tpu:change-case &optional *num* Command

This command inverts the case of the selected region, if any, or of the character under the cursor. Using a prefix argument *num* you can specify how many characters to change.

By default, this command is bound to the key `gold kp1`.

lse-tpu:change-case-lower Command

This command changes all characters in the selection or search-range, if any, or else of the current word to lower-case.

By default, this command is bound to the key `gold l`.

lse-tpu:change-case-upper Command

This command changes all characters in the selection or search-range, if any, or else of the current word to upper-case.

By default, this command is bound to the key `gold u`.

lse-tpu:char *num* Command

This command moves *num* characters in the current direction.

lse-tpu:copy-append-region Command

This command appends the selected region to the paste-buffer without deleting it.

By default, this command is bound to the key `blue kp9`.

lse-tpu:copy-region Command

Copy the selected region to the paste-buffer without deleting it. The previous contents of the paste-buffer is lost.

By default, this command is bound to the key `kp9`.

lse-tpu:current-end-of-line Command

This command moves point to the end of the current line. It will not go to another line.

lse-tpu:cut-append-region Command

This command deletes the selected region and appends it to the paste-buffer.

By default, this command is bound to the key `kp9`.

lse-tpu:cut-region Command

Delete the selected region and put it into the paste-buffer. The previous contents of the paste-buffer is lost.

By default, this command is bound to the key `kp6`.

lse-tpu:delete-head-of-line *num* &optional *append* Command

This command deletes the beginnings of the next *num* lines and stores them for the command `lse-tpu:undelete-line`. Unless you use the optional argument *append*, the old value of the line-deletion-history is lost.

By default, this command is bound to the key `A-u`.

lse-tpu:delete-head-of-line-append *num* Command

This command deletes the beginnings of the next *num* lines and appends them to the line-deletion-history for later use by the command `lse-tpu:undelete-line`.

lse-tpu:delete-next-char *num* &optional *append* Command

This command deletes the next *num* characters and stores them for the command `lse-tpu:undelete-char`. Unless you use the optional argument *append*, the old value of the character-deletion-history is lost.

By default, this command is bound to the key `kp,`.

lse-tpu:delete-next-char-append *num* Command

This command deletes the next *num* characters and appends them to the character-deletion-history for use by the command `lse-tpu:undelete-char`.

By default, this command is bound to the key `gold kp,`.

lse-tpu:delete-next-line *num* &optional *append* Command

This command deletes the next *num* lines and stores them for the command `lse-tpu:undelete-line`. Unless you use the optional argument *append*, the old value of the line-deletion-history is lost.

By default, this command is bound to the key `pf4`.

lse-tpu:delete-next-line-append *num* Command

This command deletes the next *num* lines and appends them to the line-deletion-history for later use by the command `lse-tpu:undelete-line`.

By default, this command is bound to the key `blue gold pf4`.

lse-tpu:delete-next-word *num* &optional *append* Command

This command deletes the next *num* words and stores them for the command `lse-tpu:undelete-word`. Unless you use the optional argument *append*, the old value of the word-deletion-history is lost.

By default, this command is bound to the key `kp-`.

lse-tpu:delete-next-word-append *num* Command

This command deletes the next word and appends it to the word-deletion-history for later use by the command `lse-tpu:undelete-word`.

By default, this command is bound to the key `blue kp-`.

lse-tpu:delete-prev-char *num* &optional *append* Command

This command deletes the previous *num* characters and stores them for the command `lse-tpu:undelete-char`. Unless you use the optional argument *append*, the old value of the character-deletion-history is lost.

By default, this command is bound to the key `DEL`.

lse-tpu:delete-prev-char-append *num* Command

This command deletes the previous *num* characters and appends them to the character-deletion-history for later use by the command `lse-tpu:undelete-char`.

By default, this command is bound to the key `blue DEL`.

lse-tpu:delete-prev-word *num* &optional *append* Command

This command deletes the previous word and stores it for the command `lse-tpu:undelete-word`. Unless you use the optional argument *append*, the old value of the word-deletion-history is lost.

By default, this command is bound to the key `LFD`.

lse-tpu:delete-prev-word-append *num* Command

This command deletes the previous word and appends it to the word-deletion-history for later use by the command `lse-tpu:undelete-word`.

By default, this command is bound to the key `blue LFD`.

lse-tpu:delete-tail-of-line *num* &optional *append* Command

This command deletes the ends of the next *num* lines and stores them for the command `lse-tpu:undelete-line`. Unless you use the optional argument *append*, the old value of the line-deletion-history is lost.

By default, this command is bound to the key `gold kp2`.

lse-tpu:delete-tail-of-line-append *num* Command

This command deletes the ends of the next *num* lines and appends them to the line-deletion-history for later use by the command `lse-tpu:undelete-line`.

lse-tpu:duplicate-previous-bs-word *num* Command

This command duplicates the previous *num* blank-separated words.

By default, this command is bound to the key `blue remove`.

lse-tpu:duplicate-previous-char *num* Command

This command duplicates the previous char *num* times.

By default, this command is bound to the key `blue kp,`.

lse-tpu:duplicate-previous-line *num* Command

This command inserts a duplicate of the previous line into the current line. If point is at the beginning of the current line lse-tpu:duplicate-previous-line will duplicate the entire previous line, otherwise it will duplicate the end of the previous line.

By specifying the prefix argument *num* you can instruct the command to duplicate an earlier line in the buffer.

By default, this command is bound to the key `blue pf4`.

lse-tpu:duplicate-previous-word *num* Command

This command duplicates the previous *num* words.

By default, this command is bound to the key `blue kp-`.

lse-tpu:duplicate-word-in-previous-line *num* Command

This command duplicates the word of the previous line starting in the current column.

By specifying the prefix argument *num* you can instruct the command to duplicate a word from an earlier line in the buffer.

By default, this command is bound to the key `blue gold blue kp-`.

lse-tpu:end-of-line *num* Command

This command moves to the end of the next line in the current direction. The prefix argument *num* specifies how many lines to move.

By default, this command is bound to the key `kp3`.

lse-tpu:exchange-point-and-mark Command

This command interchanges the position of point and the other boundary of the selection.

By default, this command is bound to the key `blue kp.`.

lse-tpu:exit Command

This command saves the current buffer to its associated file, if necessary, asks about all other buffers, and terminates LS-Emacs.

lse-tpu:forward-char *num* Command

This command moves *num* characters in the forward direction (reverse, if *num* is negative).

lse-tpu:forward-line *num* Command

This command moves to the beginning of the next line in forward direction. The prefix argument *num* specifies how many lines to move.

lse-tpu:goto-bs-word-tail *num* &optional *limit* Command

This command moves to the end of the next blank-separated word in the current direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

By default, this command is bound to the key `kp2`.

lse-tpu:goto-next-bs-word-head *num* &optional *limit* Command

This command moves to the beginning of the next blank-separated word in forward direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-next-bs-word-tail *num* &optional *limit* Command

This command moves to the end of the next blank-separated word in forward direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-next-word-head *num* &optional *limit* Command

This command moves to the beginning of the next word in forward direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-next-word-tail *num* &optional *limit* Command

This command moves to the end of the next word in forward direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-prev-bs-word-head *num* &optional *limit* Command

This command moves to the beginning of the previous blank-separated word in reverse direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

By default, this command is bound to the key `remove`.

lse-tpu:goto-prev-bs-word-tail *num* &optional *limit* Command

This command moves to the end of the previous blank-separated word in reverse direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-prev-word-head *num* &optional *limit* Command

This command moves to the beginning of the previous word in reverse direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-prev-word-tail *num* &optional *limit* Command

This command moves to the end of the previous word in reverse direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:goto-word-head *num* &optional *limit* Command

This command moves to the beginning of the next word in the current direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

By default, this command is bound to the key `kp1`.

lse-tpu:goto-word-tail *num* &optional *limit* Command

This command moves to the end of the next word in the current direction. By using a numeric prefix *num* you can specify how many words to move.

The optional argument *limit* restricts how far the command will move.

lse-tpu:insert-formfeed Command

This command inserts a form-feed character into the current buffer.

By default, this command is bound to the key `A-l`.

lse-tpu:invert-case Command

This command inverts the case of all characters in the selected region or the search-range or the current word.

By default, this command is bound to the key `blue kp1`.

lse-tpu:line *num* Command

This command moves to the beginning of the next line in the current direction. The prefix argument *num* specifies how many lines to move.

By default, this command is bound to the key `kp0`.

lse-tpu:move-to-beginning Command

This command moves point to the beginning of the buffer. In contrast to the analogous function of standard Emacs, this command does not set the mark.

By default, this command is bound to the key `gold kp5`.

lse-tpu:move-to-end Command

This command moves point to the end of the buffer. In contrast to the analogous function of standard Emacs, this command does not set the mark.

By default, this command is bound to the key `gold kp4`.

lse-tpu:next-beginning-of-line *num* Command

This command moves to the beginning of the current line; if already at beginning, move to the beginning of the previous line, i.e., in reverse direction. The prefix argument *num* œ[Aœ[Bœ[Bœ[specifies how many lines to move (a negative argument or zero moves in forward direction).

By default, this command is bound to the key `A-h`.

lse-tpu:next-end-of-line *num* Command

This command moves to the end of the current line; if already at end, move to the end of the next line. The prefix argument *num* specifies how many lines to move (a negative argument or zero moves in reverse direction).

lse-tpu:next-line *num* Command

This command moves point down by *num* lines (default: 1 line).

By default, this command is bound to the key `down`.

lse-tpu:next-paragraph *num* Command

This command moves to the beginning of the next paragraph in forward direction. The prefix argument *num* specifies how many paragraphs to move.

lse-tpu:page *num* Command

This command moves to the next page in the current direction. The prefix argument *num* specifies how many pages to move.

By default, this command is bound to the key `kp7`.

lse-tpu:pan-left *num* Command

This command scrolls the current window horizontally to the left. You can specify a numeric prefix *num* to scroll that times as much. The user option `lse-tpu:pan-columns` specifies how many columns are scrolled.

By default, this command is bound to the key `gold <`.

lse-tpu:pan-right *num* Command

This command scrolls the current window horizontally to the right. You can specify a numeric prefix *num* to scroll that times as much. The user option `lse-tpu:pan-columns` specifies how many columns are scrolled.

By default, this command is bound to the key `gold >`.

lse-tpu:paragraph *num* Command

This command moves to the next paragraph in current direction. The prefix argument *num* specifies how many paragraphs to move.

lse-tpu:paste-region *num* Command

This command inserts the contents of the paste-buffer *num* times into the current buffer. The contents of the paste-buffer is not changed.

By default, this command is bound to the key `gold kp6`.

lse-tpu:previous-end-of-line *num* Command

This command moves to the end of the previous line; if already at end, move to the end of the next line. The prefix argument *num* specifies how many lines to move (a negative argument or zero moves in forward direction).

lse-tpu:previous-line *num* Command

This command moves point up by *num* lines (default: 1 line).

By default, this command is bound to the key `up`.

lse-tpu:previous-paragraph *num* Command

This command moves to the beginning of the previous paragraph in reverse direction. The prefix argument *num* specifies how many paragraphs to move.

lse-tpu:quit Command

This command terminates LS-Emacs. If any buffer would need saving, `lse-tpu:quit` asks for confirmation. It will not save any buffer.

lse-tpu:quoted-insert *num* Command

This command inserts the character with decimal ASCII code *num* into the buffer. If used interactively, it will prompt for *num* — press the key you want to insert.

By default, this command is bound to the key `A-v`.

lse-tpu:replace *from to* &option *head-limit tail-limit* Command

This command will look at all occurrences of *from* by *to* in the range between *head-limit* and *to-limit* (default is to replace in the whole buffer) and ask you if you want to replace each one of them.

By default, this command is bound to the key `gold kp8`.

lse-tpu:replace-all *from to* &option *head-limit tail-limit* Command

This command replaces all occurrences of *from* by *to* in the range between *head-limit* and *to-limit* (default is to replace in the whole buffer). The command will prompt for *from* and *to*.

By default, this command is bound to the key `gold enter`.

lse-tpu:scroll-window *num* Command

This command scrolls the current window to the next section in the current direction. The prefix argument *num* specifies a repeat count.

By default, this command is bound to the key `kp8`.

lse-tpu:scroll-window-down *num* Command

This command scrolls the current window down to the next section, i.e., in reverse direction. The prefix argument *num* specifies a repeat count.

lse-tpu:scroll-window-up *num* Command

This command scrolls the current window up to the next section, i.e., in forward direction. The prefix argument *num* specifies a repeat count

lse-tpu:search Command

This command searches for a string or regular expression in the current direction. It prompts for the search-string.

By default, this command is bound to the key `gold pf3`.

lse-tpu:search-again Command

This command searches for the next occurrence of a string or regular expression in the current direction. It uses the same search-string as the last search command.

By default, this command is bound to the key `pf3`.

lse-tpu:search-forward Command

This command searches for a string or regular expression in the forward direction. It prompts for the search-string.

lse-tpu:search-reverse Command

This command searches for a string or regular expression in the reverse direction. It prompts for the search-string.

lse-tpu:select &optional *quiet* Command

This command starts a selection. The current position of point becomes one of the boundaries of the selection. By moving the cursor, you can change the outer boundary.

By default, this command is bound to the keys `kp.` and `select`.

lse-tpu:show-match-markers Command

This command shows the value of the match markers in the message window.

lse-tpu:special-insert *num* Command

Insert a character according to its decimal ASCII value *num*. Interactively, you specify *num* as prefix argument.

By default, this command is bound to the key `gold kp3`.

lse-tpu:spell-check Command

Checks the spelling of the selected region, or of the entire buffer if no selection is active.

lse-tpu:toggle-newline-and-indent Command

This command toggles the binding of the `RET` between 'newline and indent' and 'simple newline'.

By default, this command is bound to the key `blue gold RET`.

lse-tpu:toggle-overwrite-mode Command

This command toggles between overwrite and insert mode.

By default, this command is bound to the key `A-a`.

lse-tpu:toggle-regexp Command

This command toggles between string search and regular expression search mode. This mode applies to all LS-Emacs functions dealing with searching and replacing.

lse-tpu:toggle-search-direction Command

This command inverts the search direction. By default, this command is bound to the key `blue kp4`.

lse-tpu:trim-line-end Command

This command removes white space from the end of the current line.

By default, this command is bound to the key `blue TAB kp3`.

lse-tpu:trim-line-ends Command

This command removes white space from the end of all lines in the buffer.

By default, this command is bound to the key `blue ~`.

lse-tpu:show-version Command

This command displays the version of LS-Emacs you're currently using.

lse-tpu:toggle-rectangle Command

This command toggles rectangular mode for cutting and pasting of selections.

By default, this command is bound to the key `gold r`.

lse-tpu:undelete-char *num* Command

This command undeletes the contents of the character-deletion-history *num* times.

By default, this command is bound to the key `gold kp,`.

lse-tpu:undelete-line *num* Command

This command undeletes the contents of the line-deletion-history *num* times.

By default, this command is bound to the key `gold pf4`.

lse-tpu:undelete-word *num* Command

This command undeletes the contents of the word-deletion-history *num* times.

By default, this command is bound to the key `gold kp-`.

lse-tpu:unselect &optional *quiet* Command

This command cancels an active selection.

By default, this command is bound to the keys `gold kp.` and `gold select`.

lse-unexpand-fill-in Command

This command un-expands the last expansion of a fill-in. You can un-expand any number of expansions. You may later re-expand the un-expansion by using the command `lse-reexpand-fill-in`.

By default, this command is bound to the key `gold A-e`.

lse-unkill-fill-in Command

This command un-kills the last fill-in that was killed. You cannot un-kill more than one fill-in.

By default, this command is bound to the key `gold A-k`.

lse-unreplace-fill-in Command

This command un-replaces the last replacement of a fill-in. You may un-replace any number of replacements.

By default, this command is bound to the key `gold A-r`.

lse-untabify-buffer Command

This command replaces all tabulator characters in the current buffer by the appropriate number of spaces.

lse-untabify-line Command

This command replaces all tabulator characters in the current line by the appropriate number of spaces.

lse-visit-alternate-file &optional *file* Command

This command replaces the contents of the current buffer by the file *file*.

By default, this command is bound to the key `C-x C-v`.

lse-visit-file &optional *must-not-exist file* Command

This command reads the file *file* into a new buffer and switches to that buffer. If the prefx argument *must-not-exist* is specified, a new file is created if it does not already exit.

When you specify a prefix argument, you can use the shell-wildcard '`*`' as part of the file name. LS-Emacs will read all files matching the wildcarded name. Alternatively, you can enclose in back-quotes any shell command which writes a list of file names to standard output. For instance, you can visit all files containing a regular expression by using the shell command `grep`.

By default, this command is bound to the key `gold i`.

lse-visit-file-other-window &optional *must-not-exist file* Command

This command reads the file *file* into a new buffer and switches to that buffer in another window. If *must-not-exist* is specified, a new file is created if it does not already exit.

By default, this command is bound to the key `blue gold i`.

lse-window:restore-temp-hidden Command

This command removes a window temporarily thrown up by Emacs, e.g., a help window, and restores the buffer which was shown previously in that place.

By default, this command is bound to the key `blue A-f`.

lse-write-current-buffer Command

This command writes the buffer to its associated file, no matter if it was changed or not.

By default, this command is bound to the key `blue w`.

lse-write-buffer *buf* Command

This command writes buffer *buf* to its associated file, no matter if it was changed or not.

By default, this command is bound to the key `gold w`.

Appendix A Installation of LS-Emacs

This appendix describes how to install LS-Emacs.

A.1 Software required

To install LS-Emacs, you need a running Emacs system, version 19.28 or higher (see below). If you want to use wildcards as arguments to `lse-visit-file`, you also need Perl. The shell scripts accompanying LS-Emacs are written for the bash shell. To process the documentation you need a working `texinfo` system and for printing it the proper `DVI`-driver.

A.1.1 Emacs Version

I started the development of LS-Emacs on Emacs version 18.58. However, somewhere in between, I changed to Emacs version 19.22.

LS-Emacs uses a number of features of Emacs version 19, e.g., text properties and some of the new hook functions. If you're still using Emacs version 18.58, the wish to use of LS-Emacs might be a good reason to move to Emacs 19.xx.

In the meanwhile, I migrated LS-Emacs to Emacs version 20.2. LS-Emacs should still work with Emacs versions more recent than 19.28, but I'm not able to guarantee compatibility to versions older than the next to the last version of Emacs.

A.2 Installing the LS-Emacs Files

First, you have to decide where to put the files. LS-Emacs expects to find different species of files in different directories:

- Emacs lisp source files (all files with extension '`.el`'). You might put these in the directory where all other elisp sources reside on your system. Or you could choose a separate directory. In this case you have to define the Emacs load path accordingly, e.g., by adding the directory to the value of the environment variable `EMACSLOADPATH`.
- LS-Emacs template source files (all files with extension '`.lse`'). You should put these in a subdirectory of their own. You have to define the environment variable `EMACSLSESRC` to point to that directory.

  The standard distribution of LS-Emacs contains at least the languages Awk, Bash, `C++`, Elisp, Generic, Latex, Lse, Perl, Sed, and Texinfo.
- Compiled LS-Emacs template files (extension '`.lsc`'). You could either put these in the same directory as the template source files or you may choose to put them in a separate directory (as these files are derived they don't need to be backuped). You have to define the environment variable `EMACSLSEDIR` to point to the directory. Users who may add or change LS-Emacs languages need write access to that directory.

- Documentation files (extensions '`.texi`' and '`.ps`'). You can put these files anywhere you want. LS-Emacs does not need to know where they reside.
- Shell scripts and other auxiliary files (all other files). Again, you have to choose a directory for these files and define an environment variable: `EMACSLSESCRIPTS`.

  The standard distribution of LS-Emacs provides the scripts '`expand_wildcard`', '`ls-emacs-ini`', and '`lse_compile_language`'.

You then have to modify the shell script '`ls-emacs-ini`' to define the environment variables according to the directory structure you selected. Two different versions of '`ls-emacs-ini`' are supplied: one for users of `bash`, the other for users of `csh`. As '`lse_compile_language`' uses '`ls-emacs-ini.bash`', you have to modify that script even if you use the `csh`.

You may want to perform some customizations (see Appendix B [Customization], page 85), although that is not strictly necessary at this time. The most important customization would probably be that of '`lse-session.el`' and '`lse-templates-generic.lse`', which define some site-dependent lisp functions and fill-ins.

A.3 Language-Installation

To install the LS-Emacs languages provided with LS-Emacs, use the shell script `lse_compile_language`, when all files are in their proper places. The easiest way to do this is to use the command

```
lse_compile_language '*'
```

which will compile all languages available.

A.4 Activating LS-Emacs

To use LS-Emacs, add the line

```
(require 'ls-emacs)
```

at the beginning of your '`.emacs`' initialization file.

Appendix B Customization of LS-Emacs

This appendix describes some customizations of LS-Emacs, specifically:

- Customization of '`lse-session.el`'.
- Customization of '`lse-templates-generic.lse`'.
- Customization of key bindings.
- Customization of fill-in delimiters.
- Customization of automatic selection of LS-Emacs languages.
- Customization of the command menu.

B.1 Customization of '`lse-session.el`'

'`lse-session.el`' defines functions dealing with the user name and the current date. Most of these are site and operating system independent. However, there are some site-specific functions, which you might want to adapt. These are preceded by the comment '`*** site-specific ***`'.

B.2 Customization of '`lse-templates-generic.lse`'

'`lse-templates-generic.lse`' defines templates used by all LS-Emacs languages. Some of this templates are site-specific — these are preceded by the comment '`*** site-specific ***`'.

You might also want to redefine the style of some fill-ins, for instance, the format of the comment-header fill-in.

B.3 Customization of Key Bindings

The key bindings of LS-Emacs are defined by functions provided by the elisp libraries '`lse-tpu-keys-v19.el`' and '`lse-keys-v19.el`'. These functions are called by the elisp library '`lse-emacs.el`'. You have two possibilities of customizing the key bindings: either redefine some of the lisp-functions or change the function calls in '`ls-emacs.el`'.

'`lse-keys-v19.el`' defines the keys related to language sensitive functions in the lisp function `lse-define-std-keys`. If you don't like the bindings you should redefine this function in your '`.emacs`' file before loading '`ls-emacs.el`'.

'`lse-tpu-keys-v19.el`' defines the key bindings not directly related to the language sensitive functionality of LS-Emacs. You may rebind these or even do not use them at all. The following paragraphs will describe the functions defined by '`lse-tpu-keys-v19.el`'.

`lse-create-std-keymaps` creates the keymaps for the `GOLD` und `BLUE` prefix keys. You may rebind these keys, but you must not skip them!

`lse-replace-std-emacs-bindings` rebinds keys bound to standard Emacs functions to the equivalent functions of LS-Emacs. You should probably keep these rebindings. Otherwise features like buffer chaining will not work.

`lse-define-tpu-keys` defines bindings for unprefixed keys. As it defines the keypad bindings, you probably wouldn't like to skip it completely.

`lse-tpu-define-gold-keys` and `lse-tpu-define-blue-keys` define the bindings for keys prefixed by `GOLD` and `BLUE`, respectively. These do not collide with standard Emacs bindings. However, if you are used to DEC editors, you might want to change some of the bindings.

LS-Emacs assumes that you're using a terminal compatible to DEC's VT100 or VT200 series. The elisp library '`lse-keys-v19.el`' defines the mapping from the terminal's escape sequences to the symbolic key names. Use of a different terminal requires that you adapt those mappings.[1]

B.4 Customization of Fill-In Delimiters

The characters used as fill-in delimiters are defined in the elisp library '`lse-fill-in--delimiters.el`'. If you are working in an environment without support for the ISO-Latin-1 character set, you will have to change these definitions.[2]

The choice of fill-in delimiters is quite difficult. You have to reconcile a number of requirements:

- The delimiters should be short, but conspicuous.
- The delimiters have to be compatible with every language you are going to use. Compatibility with a language means that the strings used as opening and closing delimiters must not have any meaning for the language. Otherwise, an expressions meant to be part of a document might be interpreted as a fill-in. For instance, the choice of `[` and `]` as delimiters collides with many programming languages — LS-Emacs might interpret an array index as fill-in.
- The delimiters should be unambiguous. Otherwise, it will be very hard to change the delimiters later. For instance, `<` and `>` are a bad choice for this reason: a much better choice would be `:<` and `>:`.
- The delimiters should be the same for all languages. Although it is possible to define language specific delimiters, this would prohibit the sharing of templates between languages. In addition, language specific delimiters increase the cognitive load on the user.

'`lse-fill-in--delimiters.el`' contains a sample definition of 7-bit fill-in delimiters as comment.

---

[1] The mappings from escape sequences to key names should be handled differently. I've implemented a quick and dirty solution for that. If you want to change those mappings maybe you design a more elegant solution.

[2] Of course, you'll have to change all template definition files, too.

B.5 Customization of Automatic Language Selection

There are two ways to provide automatic language selection. You can either tie the language to a major mode of Emacs or you can add a language-selecting function to the `auto-mode-alist`.

You tie a language to a major mode by adding a hook to the appropriate Lisp variable. This is normally done in the language master file. You can find examples in the language master files of Awk, Elisp, LaTeX, Perl, and others.

For languages without a natural relation to a major mode, add an appropriate function to the `auto-mode-alist`. The elisp library '`lse-mode-alist.el`' defines the standard value of that variable.

B.6 Customization of the Command Menu

The command menu is defined by the elisp library '`lse-command.el`'. You can add commands to the command menu by calling the lisp function `lse-command:add`. The best place for these calls is your `.emacs` file — after loading '`ls-emacs.el`'.

Appendix C Limitations of LS-Emacs and Possible Future Extensions

This appendix describes some limitations of, and possible future extensions to, LS-Emacs.

- I started work on LS-Emacs under version 18 of Emacs. Therefore, LS-Emacs does not consistently use the features provided by version 19, most notably text properties and some advanced hook-functions.
- LS-Emacs is implemented entirely in Emacs Lisp — no change of the C sources was done. Obviously, this has a lot of advantages. And, by and large, it works satisfactorily. But of course, an implementation in C would have a better performance.
- Removing text from the buffer can result in LS-Emacs losing information about fill-in expansion. In some such cases, LS-Emacs becomes seriously confused.

  In part, usage of text properties and hook-functions could probably diminish information lossage. However, I fear that the way Emacs currently handles markers prohibits a complete solution of this issue.
- LS-Emacs still contains too many bugs. Most of them aren't really serious, yet they are annoying to encounter.
- The template definitions for some of the supported languages are not yet optimized and debugged.
- It would be nice to have more functions for manipulation of the expansion history. Some ideas:
  - Currently, the expansion history is lost when you terminate your editing session. A mechanism for recovery would be very nice.
  - Functions for navigating through the expansion history would be nice to have.
  - LS-Emacs currently does not really recognize the hierarchical structure of the expansion history, i.e., document structure. Support for this would make possible some very powerful editing mechanisms.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

Short Contents

Table of Contents