The Editor exaEdit

User's Manual

Version 02.1
17. February 2009

Peter Preus

This manual was created by means of LaTeX.

Version 02.1 — 17. February 2009

peter.preus@web.de
http://exaedit.de/en/

Preface

I fear this manual will have no better fate than most of the others. It is read much too seldom because it is only consulted – if at all – when online help, trying out and asking around does not lead anywhere. I have chosen to use a direct form of address, as well as I could, so that the manual will not feel too alien to you, dear reader, at the few times you will turn to it. This form of address is not only meant to be a stylistic element but it is also intended to indicate that, while programming, I was always aware of the fact that my products will be used by human beings. In addition to this, I hope my use of language will also encourage you to contact me concerning questions, hints or suggestions about *exaEdit*. This is rather important to me because it is a very effective way for me to improve and extend *exaEdit*.

The manual on hand consists of 5 chapters.

Chapter 1, *Survey*, provides a rough survey of the editor's features and abilities, leaving out the details.

Chapter 2, *First Steps*, is a tutorial to learn how to use the editor. It is particularly suitable for private studies because of its numerous pictures and examples. In this chapter, you will only find descriptions of the most important elements of the editor. After you have read this chapter, you will find more comprehensive information in chapter 3 to 4, if needed.

Chapter 3, *The Editor and Its Commands*, contains a complete description of all the editor's features. This chapter is relevant in all doubtful cases in usage. You should read it through, at least once, if you wish to be able to use the editor supremely well.

Chapter 4, *exaEdit Synopsis*, is intended for quick reference. Its key words represent functions of the editor (e.g. 'delete one line') with descriptions of one or more solutions to them.

Chapter 5, *The exaEdit Messages*, provides a quite comprehensive list of *exaEdit* messages, together with page reference(s) to closer explanations of the individual message.

This manual describes *exaEdit* in version 02.

Table of Contents

Chapter 1

Survey

In this chapter, you will first find some information on how to obtain *exaEdit* for your usage and then a summarized version of the main features of *exaEdit*. Please note that, in order to understand this chapter, you should know what an editor is and how editors work in general. If you do not have this knowledge yet, you should skip this chapter and immediately start with the introductory chapter, *First Steps*, instead.

1.1 How to Get *exaEdit*.

exaEdit is an editor which is available for many operating systems. Current versions exist for the Unix operating systems AIX and Linux and for the PC operating system Windows (all 32 bit versions). For the systems HP–UX, IRIX, OSF1, OS/2, and SunOS *exaEdit* exists only in older versions because the author of *exaEdit* has currently no access to those systems. If you want other versions (including those for operating systems not mentioned here) please feel free to address yourself at any time to the author. In most cases it is relatively easy to produce *exaEdit* for new operating systems.

The program number consists of a two digit version number and an update letter, for example 02B. If only errors are corrected in *exaEdit* then only the update letter will be changed in the program number. If, on the other side, new features or function are added to *exaEdit* then the version number will go up and the update letter starts again from A.

The current program number of *exaEdit* is 02B. All further information on downloading and installation of *exaEdit* you will find on the page

`http://exaedit.de/`

1.2 The History

The editor *exaEdit* – as it is available at present – has not been designed and put into effect in one attempt but it has grown through the years.

The earliest precursor which I, the present author, know of was called *XEDIT* and turned up at the computing center of the university of Heidelberg in 1975. That *XEDIT* had nothing to do with the editors nowadays known under that name and which are more or less spread all over the world. It was only a local appearance in the computing center of the university of Heidelberg in Germany. The further parts of this paragraph deal also only with the situation in Heidelberg. Fairly soon *XEDIT* had become a fullscreen editor for the timesharing system TSO of IBM's operating system MVS. As the user interface of *XEDIT* was quite pleasant, it was nearly completely imitated in the construction of the interface of the editor HADES (formerly called AMOS) which was an essential component of the timesharing system of the same name (operating under MVS as well).

When the current *exaEdit* author was confronted with the necessity to turn to the operating system Unix, he got a culture shock – not only but chiefly in the field of editors (key words: `vi`, `emacs`). He came out of this shock with every intention to keep alive the – in his view – tried and tested parts of *XEDIT* (e.g. concept and interface) in the Unix version. Since it is not possible to transfer a large and complex program which has been developed by means of software methods in the 70ies, the rewriting of the editor was inescapable. As programming language for this venture C was chosen, not out of sympathy but as the result of the choice of the least evil.

At first the name *xed* had been chosen for the program in order to make a distinction between this editor and others called `xedit` in the operating system VM of IBM and in the operating system Unix. But *xed* turned out to be a frequently used short form (e.g. in directories) for editors called `xedit` (in Unix). For this reason, the editor which will be described in this manual was renamed to *pedit* in March 1996.

Since 1993 the editor has been usable in the operating system AIX, although it has experienced large improvements and enhancements since that time.

A stable version (10A) was offered in March 1994 at the Computer Centre of the University of Heidelberg for usage under the operating system AIX, and it was offered as public domain software as well.

In the following time versions for the Unix operating systems HP-UX, IRIX, Linux, OSF1, and SunOS, for OS/2, and for all 32 bit Windows systems have been developed.

The renaming from *xed* to *pedit* was not very practical as could be seen afterwards, because *pedit* was also a name for various editors worldwide. Therefore the editor was renamed again in the year 2004 to

`exaEdit`

As the editor is connected with an internet domain name (if only for the suffix `.de`) there is some hope the name will stay unambiguously for some time.

The current version is 02.

1.3 The Concepts

In this section you will learn something about the main connecting thread which led the authors of *exaEdit* and its precursors during the development of the editor. Some other *exaEdit* features for which the term concept would be too much will be described in section 1.4, *Additional Features*.

1.3.1 The workfile

exaEdit is a file editor but it does not directly process the files which you may know e.g. from a hard disk. At the beginning of each *exaEdit* session, the data of a file are read record by record from the storage medium and then put into the main memory of the computer. Afterwards you can make your changes, which only have an effect on the copy in the main memory. At the end of your *exaEdit* session, you can write the (changed) version from the main memory back to the data storage medium. The copy of a file in the main memory is called

workfile

This concept offers the advantage that mistakes which might happen during your editing the file do not automatically concern the original file. On the other hand, any changes are lost if the computer or the operating system fail and you did not save your changes before. But this case will not occur too often and there are some ways to limit the damage.

exaEdit offers the option to have several workfiles at the same time. They may contain the same file or different files. But, always, one of these workfiles is the current one, which is visible in the window and to which the commands refer.

1.3.2 Window Mode and Line Mode

exaEdit is a whole window editor, i.e. it uses the whole window for its output and receives your input from (nearly) the whole window. A typical view in a window of 24 × 80 characters looks like this:

```
059900      j = wakt -> lwwidth;
060000    memcpy (zeile, &(*lauf).datn [n], j);
060100    pz = zeile;
060200    for (k = 1; k <= j; k ++) {
060300      if (*pz < 32 || *pz >= 127 && *pz <= 159) {
060400        *pz = '.';
060500        lauf -> flag = '.';
060600      }
060700      pz++;
060800    }
060900    *(pb + wakt -> skey) = (*lauf).flag;
061000    memcpy (pb + wakt -> skey + 1, zeile, j);
061100    m -= wakt -> lwwidth;
061200    if (m > 0) {
061300      i ++;
       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...
po60600
_




    MAIN            xed06a.c                                         6193 18/ 1
```

The upper part of the image shows the 'data zone' in which the workfile (normally only a part of it, of course) is displayed. The part below the ruler is called the 'dialogue zone' where (but not only there) you can enter your commands and where *exaEdit* gives answers or asks questions. The last line is the status line, it contains information on certain states of *exaEdit*.

You can choose nearly any window size, and you can change it during an *exaEdit* session as you like it (prerequisite your operating system allows this, e.g. X-Window), *exaEdit* will always adapt to it immediately.

The next section describes which vertical part of your workfile *exaEdit* displays.

In the horizontal dimension the beginnings of the records are shown (as a standard). If your records are too long, the surplus data will be put in additional subsequent lines below the actual line. To cope with this, one possibility you could choose is to define a 'logical' windowwidth as you like it. Thus you can determine the data which should be displayed in subsequent lines. The second possibility is to move the horizontal section over the logical windowwidth according to your preferences (not yet implemented).

If *exaEdit* cannot use the whole window, it works in the line mode, which you can ask of *exaEdit* explicitly.

1.3.3 The Current Line

One of the most important features of *exaEdit* is the current line, which follows your actions automatically.

The line in the middle of the data zone is called the current line, which normally is optically emphasized. It is the current line that commands refer to. For example, the command

```
copy
```

means "copy the current line (beneath the current line)", which results in the doubling of the current line.

Another example: the command

```
change /abc/xyz/ 6
```

means "change the character string '`abc`' to '`xyz`', do this change within 6 lines starting with the current line".

The content of the current line is not fixed but follows your actions automatically. Roughly, this means that the line you have changed last will become the current line (after you have pressed the return key). Of course, the notion of a current line exists only in your workfile.

The underlying concept is the following one: *exaEdit* assumes that you wish to start making changes on the top of your document and then work through it – change by change – until you have finished, (in the same way in which you may correct a letter, for example). So, your last change presumably is in the centre of your interest. For this reason it is positioned in the middle of your window.

1.3.4 Input

There are three ways to enter data in *exaEdit*:

- direct data change
- (line) commands
- prefix commands

Direct data change means that you move the cursor into the data zone and overwrite, insert, or delete characters.

Line commands (often simply called commands) are instructions to change, insert, and delete characters or records (see above `copy` or `change`), to position the current line, to load and save workfiles, and to apply a variety of other functions. Actually, you can enter (line) commands anywhere in your window but usually you will use the dialogue zone.

Prefix commands are commands to insert, duplicate, delete, copy, and move lines. Prefix commands are written in the data zone in the prefix area (see picture in 1.3.2) at the beginning of each line (where the line number is located).

The main principle of feeding data in *exaEdit* is that anything you have typed in remains without consequences and is still changeable until you press the `return` key. Only then will your input be processed.

For your input, which you hand over to *exaEdit* by hitting the `return` key, you are allowed to combine the three options discussed above without any restrictions – although not every combination does make sense. But frequently it is an advantage to enter direct changes and prefix commands simultaneously by pressing the `return` key only once.

1.3.5 Keyboard Usage

exaEdit uses the keyboard in a conservative way. This implies that there are no key combinations (which have to be learnt by heart) in order to call certain functions of the editor. Actually, you have to keep the commands in mind as well but this should be easier for you since the commands consist of simple words taken from the English vocabulary (see section 3.2 *Commands* as well).

For reasonable usage of *exaEdit* you will need, besides the character keys (including those modified with `Alt` etc.) and the `return` key, the following keys as well:

- the cursor moving keys (↑, ↓, →, ←),
- the delete key (`DEL`, `Entf`, ~~a~~) or the backspace key,
- the insert key (`INS`, `Einfg`, â) or an *exaEdit* function.

In many cases you will need the keys `F1` to `F12` and the `Pos1` key (`Home`) for comfortable usage, but those keys are not essential. If you wish so, you can define the `F` keys with the functions of the delete, insert and cursor keys (among others).

exaEdit recognizes additional keys as well.

1.3.6 Command Syntax

(Line) commands in *exaEdit* look like this:

```
command parameter parameter ...
```

The commands and the parameters with fixed values usually are taken from English vocabulary, so that you can keep them in mind easily.

You may abbreviate *exaEdit* commands as you like (starting at the right end of the word) as long as the command remains unambiguous. Note that there are predefined minimal abbreviations for the commands. For example, the minimal abbreviation of `copy` is `co` while the abbreviation of `change` can be as short as `c` (for `change` is in general used more frequently than `copy`).

It will be very useful for you as well that you can leave out spaces if this does not affect the distinction between the command and its parameters. For example you can write

```
COPY 1000 B
```

as

```
COPY1000B
```

1.4 Additional Features

1.4.1 Editing Directories

exaEdit allows you to edit all data sets of a directory simultaneously. For that reason all files will each get an unambiguous heading and are loaded together into a single workfile. When written back to the data storage they will be separated again if you wish so. In addition you may choose the files to be loaded by specifying their names or part of their names. Details you will find in section 3.1.3.5, *Loading all Files of a Directory*.

1.4.2 Programmability

exaEdit has some features which allow you to execute a series of editing steps in a programmed manner. *exaEdit* could be looked at rather as a kind of script language. Details you will find in section 3.1.21 *Programming the Editor* and following.

1.4.3 Profile Files

A profile file for *exaEdit* is a file that contains *exaEdit* commands carried out at the beginning of the *exaEdit* session or when a new workfile is started. With the help of these commands you can change the editor's adjustment once and for all.

There could be an installation profile file, which is the same for everybody who uses the same installed editor (e.g. when working with a workstation). But you can also have private profile files and determine their content by yourself. In section 3.1.26 *Profile Files* you will find more information on this subject.

1.4.4 Fail Behaviour

If *exaEdit* fails due to a program error, it saves every workfile as a new file on your disk – as far as possible. Thus, the danger of losing data is effectively limited.

Chapter 2

First Steps

2.1 Prerequisites

This chapter is structured in a way which should enable you to work through it without any other assistance. If you get stuck, however, it will be not necessarily your fault; there could also be a shortcoming of the text. I am quite willing to answer each of your questions and I welcome hints at mistakes and suggestions for improvement:

Peter Preus
`peter.preus@web.de`
`http://exaedit.de/en/`

If it is possible for you, you should work through this chapter on a data terminal and really key in the commands as explained below.

2.1.1 For Unix Systems

You should be able to use a suitable terminal and its keyboard, and you should be able to carry out a `login` successfully.

After the `login`, you usually get a window of the size 24×80 (24 lines with 80 columns, each). The editor is enabled to use windows of any size but it will be useful for you to use this standardized window size for a while since it is used in the following example pictures.

2.1.2 For Windows Systems

You should start the command prompt. This could be found by clicking "Start" down left, then "All Programs", then "Accessories", and then "Command Prompt".

The window you will get has normally 24 lines with 80 columns each. For reasonable working you will probably want to use a larger window, but for the beginning it is surely appropriate to use the standard extents, because these extents are used in the following figures.

2.2 Some Editor Logic

The smallest information unit which is of interest to us when we talk about editors is the byte or character.

Several characters together can form a unit, the so-called record. Such a unit can be labelled with a particular end-of-record character or it may be distinguished in some other way from the rest of the other bytes in the computer system; this is not important to us at the moment.

It is more important to know what a file and an editor are, and how you can change or create a file with the help of an editor. I will explain this to you in the next few sentences on a very basic level.

A file (sometimes also called a data set) is a sequence of records on a data medium (fixed disk, hard disk, CD–ROM, etc.).

As a common rule, a file is displayed in the window in such a way that each record of the file takes one line.

An editor is a program to create or change files. Most of the editors work – as *exaEdit* does as well – in such a way that they keep a copy of the file in the main memory (central memory, internal memory, working storage, general storage) of the computer.

If you want to e.g. change a file, you call an editor and tell it the file name. Then the editor copies the file from the data medium into the main memory. Usually, the editor displays the data it has stored in the main memory in sections in the window. After that the editor carries out the changes you will specify. The final step is to write all the data from the main memory back on the data medium. Only then the changing of the file is finished.

The copy of a file which is kept in the main memory by *exaEdit* is called

workfile

If you want to create a new file whose data still have to be put in, you must start the editor with an *empty* workfile. Then you type your data and at the end of it you give the command to make a new file out of this workfile, which now contains your data. This example is described more detailed in the section 2.4 *How to Create a File* after the next section.

2.3 For Your Orientation

As explained above, your window should be as large as 24 $\times$ 80 characters. It has to be in the basic state of your Unix session or your command prompt, i.e. accept Unix commands or line commands as input. Then please type

```
exaedit
```

as a command. As the result of the execution of this command, you receive a picture similar to the following one:

```
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
  +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|                                                                                |
 6|                                                                                |
 7|                                                                                |
 8|MAIN     exaEdit 02B TOP LINE                                                   |
 9|                                                                                |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
17|                                                                                |
18|exaEdit                                                                         |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN                                                                0 19/ 1 |
  +--------------------------------------------------------------------------------+
```

Strictly speaking, you only get the inner part of the broken line box in your window. In this manual there is additionally shown a frame with the numbering 1 - 80 on top and 1 - 24 on the left side in order to be able to refer to the distinctive parts of each picture.

In line 8, which is (in the real editor window) optically emphasized, *exaEdit* identifies itself with its name in columns 10 to 16 and its version number (consisting of two digits and one letter) in columns 18 to 20.

Behind the version number there are the words `TOP LINE`. This means that this is the first line of the workfile. The rest of the workfile is empty since you did not tell the editor a file to edit. For *exaEdit* a workfile always consists of the top line and the actual data. The top line only exists in the workfile, it is never written in a file together with the data.

In the columns 1 to 8 of line 8 you can read the word

    MAIN

This is the name *exaEdit* gave the workfile. This name implies that there can be more than one workfile; but you will learn more about this in one of the following sections.

In line 16 of the picture above, there is a ruler which is intended to help you to horizontally orientate yourself in the window in the data the editor displays.

In line 18 of the picture, there is the display of the word

    exaEdit

which is issued as an invitation to enter commands if *exaEdit* has nothing else to say (this will be commented on later).

The last line of the picture is the status line, which informs you on different states and gives some other pieces of information.

In the status line in column 5, the name of the workfile is repeated.

In column 73, you can find the number of records of the workfile. Since this workfile is empty, there is the number `0` issued.

In columns 75 to 79, the cursor position is indicated. As the cursor is in line 19, column 1 at the moment, you can read the output '`19/1`'.

The lines above the ruler, in this case lines 1 to 15, are called the

data zone

The lines between ruler and status line, here lines 17 to 23, are called the

dialogue zone

After so much dry information, you are allowed, now, to do something: Please, move the cursor with the arrow keys across the window and observe, while doing this, how the display in the status line changes. And, please, note the cursor behaviour when the cursor is moved out of the window. Keep in mind this behaviour and use it when you have to move the cursor while editing. Since not a small amount of your work with the editor will consist of cursor movement, it will be useful for the efficiency of your work to do this with a minimum of effort.

Now, move the cursor back to its initial position (so that there is '`19/ 1`' displayed at the lower right corner again).

2.4 Creating a File

As it was mentioned briefly in the previous section, 2.2, *Some Editor Logic*, you can create a file by starting the editor with an empty workfile first, then filling it with your data and writing the data into a new file at the end.

At the moment you see the empty workfile in front of you. There are several methods to fill a workfile with some data but in this section you will only learn something about the input mode. Initially you have to enter the command

`input`

which means that you first have to type the word `input` (in line 19) and then press the `return` key. As the result of this, the window should look like this:

```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|                                                                                |
 6|                                                                                |
 7|                                                                                |
 8|MAIN     exaEdit 02B TOP LINE                                                   |
 9|                                                                                |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|....;....1....;....2....;....3....;....4....;....5....;....6....;....7....;....8|
17|input                                                                           |
18|Input                                                                           |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN      I                                                         0 19/ 1 |
 +--------------------------------------------------------------------------------+
```

What has been changed compared with the previous picture?

- The command you have just given is repeated at the beginning of the dialogue zone, i.e. in line 17.
- In the following line the word

  `Input`

  has appeared as a request for the input of data.
- The cursor is at he beginning on the next line (19), ready for the input of data.
- In the status line an

  `I`

  has appeared in column 15 in order to show that *exaEdit* is in the input mode now.
- The ruler appears now at the very left of your window since you will start writing your data in column 1.

Please, now enter the first line of your input, for example

```
This is the first line.
```

You have to type the sentence and then press the `return` key. After this your window looks as follows:

```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|                                                                                |
 6|                                                                                |
 7|MAIN     exaEdit 02B TOP LINE                                                   |
 8|       This is the first line.                                                  |
 9|                                                                                |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|....;....1....;....2....;....3....;....4....;....5....;....6....;....7....;....8|
17|This is the first line.                                                         |
18|_                                                                               |
19|                                                                                |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN     I                                                         1 18/ 1 |
 +--------------------------------------------------------------------------------+
```

These things have changed this time:

- Your input is repeated in the first line of the dialogue zone (in line 17) – as it happened with the command `INPUT` before.
- The line you have just entered has been transferred into the workfile, which can be seen from the fact that it appears in line 8 of the data zone.
- In the status line, in column 73, the display of the counter of records in the workfile has jumped to 1.
- The cursor is positioned in the next free line in the dialogue zone (in line 18), ready for your input.

Now, please, enter the second line, e.g.

```
The second line
```

by typing the characters and pressing the `return` key, again.

This line, as well, is repeated in line 17 and transferred into the workfile, which is mirrored by its appearing in line 8. The first line you have entered has slipped upward by one line.

After you have entered the third line

```
line three
```

you will get the following picture:

```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|MAIN     exaEdit 02B TOP LINE                                                   |
 6|       This is the first line.                                                  |
 7|       The second line                                                          |
 8|       line three                                                               |
 9|                                                                                |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|....;....1....;....2....;....3....;....4....;....5....;....6....;....7....;....8|
17|line three                                                                      |
18|_                                                                               |
19|                                                                                |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN      I                                                       3 18/ 1 |
 +--------------------------------------------------------------------------------+
```

Now you have entered enough, and you wish to quit the input mode. This happens if you press the `return` key without having typed another character before your last pressing the `return` key. After this, the window looks like this:

```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
+--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|MAIN     exaEdit 02B TOP LINE                                                   |
 6|                                                                                |
 7|000100 This is the first line.                                                 |
 8|000200 The second line                                                         |
 9|000300 line three                                                              |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|                                                                                |
17|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
18|line three                                                                      |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN                                                              3 18/ 1 |
+--------------------------------------------------------------------------------+
```

The character `I` in the status line has disappeared since *exaEdit* is not in the input mode any more.

The three lines in the workfile each received a number.

The next thing I assume you like to do is to write the workfile onto the disk. You give rise to this by entering the command `FILE`. You have to specify the file name you desire. If you wish that the file name is `exafil1`, you now have to enter the command

```
file exafil1
```

(do not forget the `return` key!). This command is repeated in the first line of the dialogue zone as well. In the next line there will be the output:

```
New data set, press J or Y to create it:
```

This means that *exaEdit* has detected that there exists no file called `exafil1`, yet. If this file already existed, the message would be:

```
Old data set, press J or Y to replace it:
```

This behaviour of *exaEdit* is intended to protect you from mistakes as, e.g, typing errors which could cause another file name, different from the one you actually wanted.

The letters `j` or `y` stand for the words German 'ja' or English 'yes'. *exaEdit* recognizes them immediately when you press one of the keys `j` or `y`. So it is not necessary to additionally press the `return` key and, to a certain extent, this would be even damaging since the reaction of *exaEdit* to the key you have pressed would not be visible, then.

After so much explanation, please, press the key `j` or `y`. As the reaction to this, *exaEdit* writes the contents of your workfile in the file with the name you have specified before (and the file is now created in order to do this). To show you that the action was successful, you receive the message

```
Data saved
```

Now you are allowed to finish this lesson and quit *exaEdit* by entering one of the following commands:

`end` or `quit`

2.5 Upper and Lower Case, Abbreviations

As you have seen in the previous section, you can write your data as usual in capital or small letters, that reach the workfile in the same way you have typed them.

When you enter commands in *exaEdit* (e. g. `INPUT`, `FILE`, `END`, `QUIT`) or when you give answers (`j`, `y`), small and capital letters are interchangeable, they are never distinguished by *exaEdit*. This is, of course, only true for command names and operands which have a fixed meaning in *exaEdit*, whereas operands chosen by yourself such as file names or character strings are case-sensitive as a matter of course.

In this manual *exaEdit* commands are written in the text in upper case so that they are better recognizable as commands. But in the examples, commands are usually written in lower case. On the keyboard, of course, you will only type small letters.

You can abbreviate the very most *exaEdit* commands to reduce the amount of typing; for example, you abbreviate `INPUT` with `I`, `FILE` with `FIL`, `END` with `E` and `QUIT` with `Q`. Later there will be some explanation on the minimal abbreviations that are possible. This manual always uses the full commands in its texts to make it easier for you to understand and recall them (it is quite clear what ‘`END`’ does; but what does ‘`E`’ do?). In the examples, newly introduced commands are fully written but later on, when you are supposed to know them, they are often abbreviated. When you use the editor more intensively, you will learn the minimal abbreviations by heart – since it will help you to save a lot of time.

2.6 Keys to Delete and Insert Characters

When you are typing, mistakes can happen easily. You can correct them early and easily if you have not yet pressed the return key with which you finish your input in *exaEdit*.

If you have typed

```
Thisz is the ...
```

for example, you move the cursor back to the superfluous character and press the

```
Del
```

key, which should be on your keyboard. (Sometimes it is labelled with `Entf` or with something else.)

An alternative to this procedure offers the `backspace` key. It is above the `return` key and it has an arrow pointing to the left on it (please do not mix it up with the ‘cursor to the left’ key that looks very similar to the `backspace` key). The backspace key deletes the character on the left of the cursor. Anything on the right of that character moves to the left by one position. For comparison, the `Del` key deletes the character on the cursor position while the `backspace` key deletes the character before; in both cases anything on the right moves to the left by one position.

But if you have typed

```
Thi is the ...
```

– and forgotten the `s`, as you can see – you move the cursor back to the place where you have to insert the character (i.e. the space after ‘`Thi`’) and press the

```
Ins
```

key, that should be on your keyboard (and which is sometimes labelled with ‘`Einfg`’ or something else). Finally, you press the key with the missing character.

When you have pressed the `Ins` key, you have put *exaEdit* into the insert mode (please, note the difference between insert mode and input mode, the term input mode is explained in the section 2.4). ‘Insert mode’ means that any additional characters are inserted at the cursor position. The sign $\wedge$ that is displayed in column 14 in the status line helps you to recognize that *exaEdit* is in the insert mode. You leave the insert mode by pressing the `Ins` key again.

The insert mode is also turned off automatically each time you press the `Enter` key. It is also possible to induce *exaEdit* to use the insert mode permanently (see chapter 3).

Later, in the lessons 2.8 and 2.16, you will learn how you can remove typing errors that you only recognize after you have entered them into the workfile with the `return` key.

2.7 Editing a File That Already Exists

The next thing you might do is to edit the file `exafil1`, that you have just created, again. To do this, you type

```
exaedit exafil1
```

and you receive the following picture:

```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|                                                                                |
 6|                                                                                |
 7|                                                                                |
 8|MAIN     exaEdit 02B TOP LINE exafil1                                            |
 9|000100 This is the first line.                                                  |
10|000200 The 2nd line                                                              |
11|000300 line three                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
17|                                                                                |
18|exaEdit                                                                         |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN         exafil1                                              3 19/ 1 |
 +--------------------------------------------------------------------------------+
```

Compared to the last picture in the lesson 2.4, *Creating a File*, the following has changed: Both in the top line and in the status line there is the file name

```
exafil1
```

put down. If you now make some changes in the workfile, you only need to enter the command

```
file
```

to save them at the end; *exaEdit* will write the workfile in the file `exafil1`.

But you could also write the workfile into another file by indicating its name in the `FILE` command:

```
file filnew
```

Please, enter this command. After this, you receive the following request, again:

```
New data set, press J or Y to create it:
```

In contrast to the lesson 2.4, *Creating a File*, I now suggest that you press any (character) key except J or Y. As this could happen involuntarily (you actually wanted to type Y but you missed the key), *exaEdit* warns you by giving a message and a sound that the FILE command has not been finished.

```
ATTENTION: Data not saved!
```

But now, in order to have a second file for later lessons, you repeat the FILE command, please, and answer the question with J or Y. Then finish this *exaEdit* session with the command QUIT or END.

2.8 Changing Data Directly

In this lesson you will learn how you can change data that already exists.

Please, enter the command

```
exaedit exafil1
```

as in the previous lesson. exafil1 is the file you created in the lesson *How to Create a File*. Now the window should look the same way it did in the previous lesson.

The changes you might like to do could be, e.g., to replace 'first' by '1st' and 'three' by '3'. To do this you only have to move the cursor with the arrow keys to the correct position in the data zone and simply overwrite the data there. When you overwrite the word 'first' with '1st', you have some characters left. You can delete them with the Del key (compare section 2.6, *Keys to Delete and Insert Characters*). If you had to insert one or more characters, you would use the Ins key.

If you want the changes to reach the workfile, you have to hit the return key. But if it suddenly occurs to you that you actually do not like these changes, you can press the Pos 1 (or Home) key instead of the return key. The result of this is the state before you started making changes.

After you have pressed the return key, you have no chance to undo your changes at once. Of course, you can do without saving your workfile but in this case you lose any other changes you might have done previously in your *exaEdit* session.

2.9 How to Quit *exaEdit*

This lesson directly follows the previous one, in which you made some changes but did not give the command FILE to save them.

Please, try now to end this *exaEdit* session with the command

```
quit    or    end
```

Since *exaEdit* knows that the changed workfile has not been written back onto the disk so far, it tries to protect you from an ill-considered step. For this reason, *exaEdit* answers with the following hint and request:

```
Changes not saved
Press J oder Y to stop:
```

When you now press any other (character) key – please, do so – *exaEdit* will cancel the processing of the QUIT or END command, put down exaEdit in the dialogue zone and wait for the commands you give next.

If you had, in contrast to this, pressed one of the keys J oder Y, *exaEdit* would have finished its working without saving your changes.

2.10 Current Line, Positioning

As you may have noticed in the previous lesson, the three lines in your file have changed their position after you have changed your text. Before your changes, there was the top line (that is the line with '`MAIN      exaEdit...`') emphasized (white letters on black background) and positioned in the middle of the data zone. Now, after your changes, the third line of your text takes this highlighted position.

The emphasized line in the data zone is called

current line

In *exaEdit* it has two special features:

- it is the starting point of commands which need the indication of a certain line,
- it is automatically readjusted.

Because of the first characteristic of the current line, it has to be possible to move it by giving commands. The next thing I would like you to do is to exercise the so-called positioning of the current line (please, try everything I suggest below).

With the command

```
top
```

the top line becomes the current line. With the command

```
bottom
```

the last line becomes the current line. Since they are both frequently used commands, they have the minimal abbreviations `T` and `B`.

With the commands

```
+ n
down n
next n
```

– where you have to insert a number for '`n`' – the current line is moved downward by `n` lines, i.e. closer to the end of the file.

With the commands

```
- n
up n
back n
```

the current line is moved upward, i.e. closer to the top of your file.

If you specify more lines than there are in your file, the current line remains unchanged and *exaEdit* gives the message `End of data` or `Begin of data`.

But if you like to move the current line only by one line, you can leave out the line number `n` and only enter `+`, for example.

In many cases you will wish that the current line exactly moves by the whole window height or only by the half of one page on your window. There are special keys for this case:

`F7`	one page back
`F8`	one page forward
`F10`	half a page back
`F11`	half a page forward

Instead of F7 and F8 you can use the keys PgUp↑ and PgDn↓, as well, prerequisite they are on your keyboard. Since your file exafil1 does not contain enough lines, you will no really see the effect of those keys if you try them now. But you will see how these special keys are translated into *exaEdit* commands, e.g.

```
-14
+14
-7
+7
```

if your window has 24 lines.

When you jump one page forward, *exaEdit* works in such a way that the last line of the previous window becomes the first line of the next display. This is meant to provide a certain coherence when you move down a text.

Now, in this context, it only remains to explain how the automatic positioning of the current line works.

When you make your changes directly, in the data zone (compare section 2.8), the lowest line of those you have changed will become the current line – after you have pressed the return key. The reason for this behaviour is that *exaEdit* assumes that the change in the lowest line is the last change you have done and that the last change should be placed in the middle of the window (in the centre of your interest).

The current line can also move as a result of certain commands. There are predefined rules for this and these rules can differ from command to command. This is the reason why these rules are explained in detail together with the explanation of each command in the next chapter.

2.11 Inserting Lines

There are several possibilities to do this. The first solution is the input mode, which you learnt to know in the section 2.4. If you would like to insert lines between the first and the second line of your data, for example, you have to position your workfile (compare chapter 2.10, *Current Line, Positioning*) in such a way, that the first line is the current line. Then you put *exaEdit* into the input mode with the command

```
input
```

Anything else should be familiar to you from the section 2.4.

A second method to insert lines uses the

Number command

To apply it, you have to type a line (not in the input mode but in the *exaEdit* home position). This line contains not only your data but also the number of a line in front of it. In other words, it is something which looks like a line in the data zone:

```
222 text
```

From this input *exaEdit* creates the corresponding line in the data zone. Your choice of the line number determines the position of the new line. It is up to you whether or not you write the number with leading zeros.

By the way, it does not matter whether or not you put a space after the number:

`222text` or `222 text`

Both spellings are treated as the same. But, in contrast to this, additional spaces make a difference.

If the line number you have specified does already exist, there will not be a new line inserted but the existing line will be overwritten.

Further methods to insert (special) lines will be described in later sections.

2.12 Deleting Lines

There are several methods to delete one or more lines. The command

```
delete n
```

deletes `n` lines beginning with the current line. When you leave out the specification `n`, there will be only one line deleted (the current line). The command DELETE has the minimal abbreviation DE.

Another method to delete lines offers the command

```
dl  from  to
```

in which you have to specify for `from` and `to` the first and the last number of the lines to be deleted. DL stands for delete lines.

The third possibility to delete lines allows you to type '`d`' into the number area of the line you want to be deleted:

```
0003d0
```

After you have pressed the `return` key, any line you have marked in this way is deleted. You can write the label `d` wherever you like in the number area.

You can apply this method to delete several lines in a row. The only thing you have to do is to type the number of lines to delete as well:

```
0003d2
```

This deletes the marked line and the following one. If you type the character `d` at the beginning of the number, you have to use a space after the specification of the number of lines to delete.

```
00d300
```

this deletes only the marked line, while the following example deletes three lines:

```
00d3 0
```

Additional methods to delete lines are described in other sections.

2.13 Copying Lines

You will frequently face the fact that you need one line twice. Of course, you do not need to type it in twice. The command

```
copy  from  to
```

copies the lines you have specified (`from` and `to` are line numbers) beneath the current line. If you wish to copy one line, you only need to specify the number of this line. If you leave out both line specifications, *exaEdit* copies (doubles) the current line.

Besides the real line numbers, *exaEdit* recognizes symbolic line numbers, i.e.

`*`	the current line,
`p`	the line above the current line ('previous'),
`n`	the line beneath the current line ('next'),
`f`	the first data line of your workfile ('first'),
`l`	the last line of your workfile ('last'),
`b`	the last line of your workfile ('bottom'),
`t`	the first line of your workfile ('top'), where *exaEdit* refers to the top line or the first line containing data depending on the context,
`s`	the line which has been marked by SET.

Examples for the `COPY` command:

`copy 100 200`	copies the lines from `100` to `200` beneath the current line.
`co`	doubles the current line.
`co *`	doubles the current line.
`co p`	doubles the line above the current line.
`co p *`	copies the line before the current line and the current line beneath the current line, i.e. it changes the two lines `a b` to the 4 lines `a b a b`, if `b` was the current line, before.
`co f l`	copies all lines beneath the current line (only possible if the current line is the last line or the top line.

An additional method to copy lines you will find in section 3.3, *Prefix Commands*.

2.14 Moving Lines

For moving lines there is the command

```
move  from  to
```

It works in the same way as `COPY` – with the single exception that the lines you specify are not copied but moved. Examples for this are:

`move 100 200`	moves the lines from `100` to `200` beneath the current line,
`mo p`	exchanges the current line and the previous one.

An additional method to move lines will be implemented in future versions of *exaEdit*.

2.15 Searching Data

When you are looking for a certain passage in your data, you can page through your file. But you can also find a particular location if you precisely know what is written there. The command

`locate /`*`character string`*`/`

starts its search from the current line and searches for the character string you have just entered. If it is found, the line that contains this character string becomes the current line, e.g.

```
locate /the/
```

The character string you are looking for has to be embedded in two delimiting characters (where you can leave out the end delimiter sometimes). The character to delimit the string (in the example above it is a '`/`') can be any character except a numeral, the character '`&`', or the command separator. The following command, for example, would search for the character string '`he`' ('`t`' delimits the beginning while the end marker is missing):

```
locate the
```

It will be easiest for you, probably, when you get used to one character as delimiter of character strings. I would suggest the ♯ character on the right on your keyboard (in the special character field). But, if this character occurs within the character string itself, you have to choose another delimiter for this string.

When the search for your character string has reached the end of your workfile, the search will be continued from the beginning of the data. *exaEdit* makes you aware of this procedure by writing the following message on your window when it passes the end of the file:

```
Search from begin (wrap)
```

(As an alternative, you can stop the search at the end of the workfile; compare the explanation on the command `WRAP` in the next chapter.)

If you have to look several times for one and the same character string, you only need to type in

```
locate
```

alone, the following times. Since the minimal abbreviation of LOCATE is L, you do not even need to type in more than

```
l
```

Besides the search from top to bottom, *exaEdit* can also search from bottom to top. This is possible with the command

rlocate /*character string*/

(rlocate = 'reverse locate'). If in this case the search passes the beginning of the workfile and continues at the end of it, then *exaEdit* writes

```
Search from end (wrap)
```

in the window.

The memory for the search is the same for LOCATE and RLOCATE.

2.16 Changing Data

In section 2.8 you learnt to manage the direct change of data. Another possibility to do changes was the use of a number command with an existing number. But with the help of the number command you could only change the whole lines.

One more method to change data offers the command CHANGE; its minimal abbreviation is C:

```
change /old/new/
```

In this example, the character string

```
old
```

will be replaced by the character string

```
new
```

in the current line. The character '/' again functions as delimiter of the character string. As you have just learnt in the previous section, you can also use other characters as delimiters.

You will like to make the same change in several lines frequently. For this purpose, you can add a definite number of lines to your change command. For example, the following command

```
c // /5
```

– beginning in the current line – moves the data by one position to the right; this change is done within five lines. The 'old' character string is empty (two delimiters follow each other directly). This character string occurs at the beginning of each line. The 'new' character string contains one space.

More about the use of the CHANGE command can be found in chapter 3.

2.17 Help

exaEdit includes rather condensed help texts. They are very useful if you know a certain command and the function you need and if you cannot recall the precise syntax. The command

```
help
```

– minimal abbreviation H – provides you with a list of all *exaEdit* commands. The capitalized initials of the command words you find there denote the minimal abbreviations; the whole help list of commands is sorted according to the abbreviations.

With the command

```
help command
```

you receive information about the command you specified. For example, the result of `HELP CHANGE` looks as follows:

```
help change
Change [col1 [col2]] /string1/string2/ [n] [A] [D] [H] [I]         change data
 string1 is searched in n lines (default 1) and replaced by string2. Without
 string2 string1 is removed. 'A' changes all occurrences in the line, D displays
 changed lines. H interprets hexadecimal. 'I' searches case-insensitive.
 Search is restricted by ZONE or the specified columns (which come first).
```

This information is structured as follows:

In the first line there is the syntax on the left, and a very brief characterization of the function of the command on the right. Beneath, there is an explanation of the function and the parameters.

For the syntax, square brackets [ ] are used to mark specifications that can be left out.

Please, try to get used to this manner of representing information and look at the help texts of some commands you have learnt to know so far.

Usually, the full *exaEdit* user manual can be read on–screen (via WWW–Browser) by using the command `MANUAL`.

2.18 An Important Command

Please pardon my misleading you a bit: There are no important or unimportant commands. In this chapter, *First Steps*, I have tried to show you anything you need if *exaEdit* should become a useful instrument for you.

But it is important to note that the things you have learnt will not be enough for a number of special applications and, particularly, for a highly efficient usage of the editor. So, there is no other choice for you but to read through the following chapter *The Editor and its Commands* at least once patiently and with concentration. This should enable you to remember the things you have read and find them again if you need them.

Chapter 3

The Editor and Its Commands

This chapter contains a complete and comprehensive description of any feature of the editor. In any case of doubt or whenever the impression of discrepancies between this chapter and other chapters or other sources of information on *exaEdit* should rise, this chapter is relevant. Please keep in mind that in the following first subchapter on different functions of *exaEdit* not all commands are mentioned. Therefore you are strongly recommended to study the subchapter on all commands as well.

3.1 Functions

This chapter describes the functions *exaEdit* has from your point of view. These functions may sometimes be realized in different ways, and sometimes their realization requires different *exaEdit* commands.

3.1.1 Starting a *exaEdit* Session

Usually, the start of an *exaEdit* session is the input of the line command

```
exaedit [filename]
```

The specification `[filename]` in square brackets means that you may or may not specify the name of a file.

If you call *exaEdit* with more than one parameter, only the first one will be honoured and any additional parameter is ignored without further notice. In the section 3.1.3, *Loading a File* you will find an explanation of what the parameter means in detail.

First, *exaEdit* reads the profile files if one or more exist. This is explicitly explained in section 3.1.26, *The Profile Files*.

Second, *exaEdit* has to inform itself on the properties of the screen or window used and it has to activate the ways from and to the window. *exaEdit* is a whole-window editor, i.e. *exaEdit* produces the data for the whole window and – the other way round – the editor can receive changes from anywhere in the window.

In order to fulfill the whole window function, *exaEdit* uses a complement of the operating system which is called

Curses

This stands for a collection of program functions which can be used to begin, perform and end the usage of the whole window. Since you will prefer in normal cases the whole window option, *exaEdit* tests at this stage whether you can use the whole window or not. If this is the case, Curses is initialized.

The remainder of this section refers to Unix systems only.

The first prerequisite is the availability of the environment variable

```
TERM
```

Environment variables are provided by the operating system. You can ask for a list of them with the Unix command

```
set
```

If `TERM` does not exist, against any expectation, *exaEdit* displays the following two messages

```
TERM not defined
exaEdit in line mode
```

in the window. The first message explains itself, the second one means that *exaEdit* cannot work in the whole window mode (i.e. the initialization of Curses is not possible) but only line by line. More detailed information on the line mode is available in 3.1.20, *The Line Mode*.

If the environment variable `TERM` exists, *exaEdit* checks the value of the parameter and whether it is appropriate for the window mode. At the moment,

```
TERM=NETWORK         and
TERM=IBM3278-x
```

(with any `x` value) are recognized as inappropriate. In this case *exaEdit* displays those two messages

```
Terminaltype is ...
exaEdit in line mode
```

in the window and continues in the line mode (compare section 3.1.20, *The Line Mode*).

Unfortunately, at the moment it is not possible for *exaEdit* to recognize all `TERM` values that are inappropriate for the window mode. So, it may occur that *exaEdit* tries the initialization of Curses but Curses aborts *exaEdit* with an error message. Messages of this type look as follows:

```
Sorry, I don't know how to deal with your '...' terminal.
Sorry, I need to know a more specific terminal type than ''.
```

But what can you do if *exaEdit* switches to the line mode – against your will – or if one of the `Sorry` messages appears? As a solution, you could define the environment variable `TERM` with the Unix command

```
export TERM=...
```

– where you replace the dots by the terminal type you like. If you decide, after you received one of the `Sorry` messages, to continue work at least in the line mode, you can enter, for example, `export TERM=NETWORK`.

3.1.2 Workfiles

exaEdit – as well as many other editors – follows the common procedure, which is to load a file from the data medium into the main memory, to have the file changed there, and eventually to write it back onto the data medium. The copy of the file in the main memory is called the

workfile

exaEdit can have several workfiles at the same time, the number of them is not restricted (except, of course, the case that the main memory is not large enough).

Every workfile has its own name, which consists of eight letters or digits and begins with a letter.

The first (or only) workfile in a *exaEdit* session is called

```
MAIN
```

You may spell workfile names in capital or small letters, which is not distinguished by the program. *exaEdit* always displays workfile names in capital letters.

The workfile name is displayed in the status line (i.e. the last line on your window) in columns 5 to 11 and in the top line of the workfile (top line, compare next paragraph).

The workfile does not only contain the actual data but also the so-called

    top line

You could imagine this as a fictitious 0 record which only exists to designate the beginning of your workfile. The top line is displayed in the window but it does not exist in the file on the disk since it is only created when the data are loaded and it is not written when the data is saved to the data medium.

Some commands treat the top line as a normal data line, which is not changeable, however.

To create and delete workfiles, and to switch from one workfile to another, you use the command

```
workfile
```

There can only be one current workfile at one time, which is the only visible one in the window and to which any given command refers. But it is possible to copy data from one workfile to another.

If you prefer to see two or more workfiles at the same time, you cannot do this with one single *exaEdit* session; you need two or more windows or screens.

3.1.3 Loading a File

3.1.3.1 Loading in the Normal Case

This section describes the usual routine to get a file into the *exaEdit*–workfile. The following sections deal with possible special cases.

There are two different ways to load a file, which means to read from the data medium and bring the file into the workfile. The first method is to specify a file name when calling *exaEdit*:

```
exaedit file
```

If the file name contains special characters, you have to make special arrangements, for example, put the name in quotation marks ("). But this does not help in every case. How you actually have to deal with this problem is, for Unix, determined by your user interface (shell) or has to be looked up for Windows.

If the loading was successful, the file name is assigned to the workfile, which means that it appears in the top line and in the status line. As a result of this, when the command `FILE` is given without the specification of a file name, the workfile will be written in the file with this name.

The second method to load a file is to use the command `LOAD` with a file name during a running *exaEdit* session:

```
load file
```

The file name, either the one which is assigned to the workfile or the one you specified in the `LOAD` command, can be an absolute or a relative file name.

Now we have to differentiate between Unix and Windows systems.

In Unix systems:

The absolute file name begins with a slash, the relative one does not. If you specify a relative file name, it will be completed to an absolute file name by putting the current directory in front of your relative specification. If you want to find out the name of your current working directory, you may use the *exaEdit* command

`call pwd`   or   `_pwd`

which both display the result of the Unix command `pwd` (pathname of the working directory). Example: If

```
/u/fmath/ppreus/exaEdit
```

is your current working directory, the commands

`load etc/one` or `load /etc/one`

will read the files

`/u/fmath/ppreus/exaEdit/etc/one` or `/etc/one`

You may also want to refer to the parent directory and use the common spelling convention to achieve this. In the example above you would use the command

```
load ../abc/two
```

to read the file

```
/u/fmath/ppreus/abc/two
```

and be successful with this although your current working directory is

```
/u/fmath/ppreus/exaEdit
```

Additionally, you may also use the spelling with the tilde (~) for file names. In the next example,

```
~/...
```

the tilde stands for your HOME directory, in the following example,

```
~uid/...
```

`~uid` stands for the HOME directory of `uid`.

In Windows systems:

The absolute file name begins with a backslash (‘\’) or drive letter, the relative one does not. If you specify a relative file name, it will be completed to an absolute file name by putting the current directory in front of your relative specification. If you want to find out the name of your current working directory, you may use the *exaEdit* command

`call cd` or `_cd`

which both display the result of the DOS command `cd`. Example: If

```
d:\pe\dok
```

is your current working directory, the commands

`load winnt\win.ini` or `load \winnt\win.ini`

will read the files

`d:\pe\dok\winnt\win.ini` or `d:\winnt\win.ini`

You may also refer to the parent directory and use the common spelling convention to achieve this. In the example above you would use the command

```
load ..\abc\two
```

to read the file

```
d:\pe\abc\two
```

and be successful with this although your working directory is

```
d:\pe\dok
```

From here on it is about **Unix and Windows systems** both.

If the file name contains special characters, you have to apply the following rule: Put the entire file name in single apostrophes (`'`); replace an apostrophe within your file name by two apostrophes.

When loading a file with the LOAD command, it is important to know that loading a file is also possible for workfiles that are not empty. In this case, the file will be loaded behind the record of the current line. In order to join two files in this way, you have to load one of the two first (by means of *exaEdit* or LOAD), then you have to give the command BOTTOM and then you can load the other file with LOAD.

If the loading with the LOAD command was successful, the name of the loaded file will be assigned to the workfile (in other words, the file name becomes the file name of the workfile) if there has not been a file name for the workfile before. Contrastingly, if there already exists a file name for the workfile, it will not be changed through the execution of the LOAD command.

If you decide to transfer the result of your editing into another file, you can do this with the FILE command, compare section 3.1.4, *Saving a File*.

Now, a few words about possible errors that could occur in the process of loading a file.

If the file you specified cannot be found, maybe because you made a spelling mistake or the file is in another directory, you will receive the following message:

```
File not found
```

Despite of this, the specified file name will be assigned to the workfile if there has not been a name for the workfile so far and if you have asked for the loading of the file by specifying a file name when you called *exaEdit*. This behaviour allows you to start the creation of a not yet existing file with the definition of its name which you then need not repeat later with the FILE command.

'Common' mistakes you may make when you load a file are the following ones:

```
Parameter missing
```

This message occurs if you use the LOAD command without a file name.

```
Ending ' missing
```

This message occurs if you begin the file name in the LOAD command with an opening apostrophe (') and *exaEdit* cannot find the corresponding closing apostrophe, and therefore does not know which file name is actually meant. The latter is the case when you use the opening apostrophe and your command line contains a space somewhere behind the opening apostrophe. Vice versa this means: If the file name which you want to enclose in apostrophes does not contain spaces, you may leave out the closing apostrophe.

```
Access not allowed
```

This message occurs when you do not have the rights to access the file, for example, because the file belongs to somebody else.

```
A directory cannot be edited
```

This message occurs if you specified a directory name instead of a file name. *exaEdit* only manages to edit files (also see chapter 3.1.3.5, *Loading all Files of a Directory* for further information about this).

```
No file and no directory
```

This message occurs when the object you tried to load neither was a file nor a directory. It cannot be edited.

```
No connection to another computer
```

This message occurs if in networking computers the searching for a file or the check for access requires the service of another computer ('server') and the operating system cannot get connected to this server.

```
Part of the name is no directory
```

This message occurs when you specify a qualified file name (i.e. several subnames that are connected with '/' or '\' resp.) and not every part except the last one is a directory.

```
Too many symbolic links, refer to itself?
```

This message occurs if the operating system tries to solve those parts of the file name that point to other names and the maximal number (which is provided for the operating system) is exceeded. The most frequent reason for this error is a file name which points to itself (direct or indirect).

```
access errno = ...
getcwd errno = ...
stat errno = ...
```

These messages should never occur. They appear if certain errors happen, for which *exaEdit* provides no special message. In such a case, you should record the complete message together with the circumstances of its appearance and send this information to the author of *exaEdit*.

```
Data set not opened (does not exist?)
```

This message appears if all the preliminary checks *exaEdit* performs are positive but the file would not open in spite of this.

3.1.3.2 Loading a File via DD–Names

As an alternative to specifying a file name with the `LOAD` command, you can use an environment variable. Before doing so, you have to connect the environment variable to a file name by specific means.

For example, you could define the environment variable `DD_ABC` as follows in a Unix system:

```
export DD_ABC=/u/fmath/ppreus/qwer
```

Or in a Windows system:

```
set DD_ABC=h:\dok\qwer
```

After having done that, you can load the file '`qwer`' by using the command:

```
LOAD (abc)
```

The parentheses serve to distinguish this special case from the standard case of a file name.

The `LOAD` command does not distinguish between upper case or lower case letters in the DD–name, but in Unix the environment variable must be written in upper case letters only. The name of the environment variable must contain the three characters `DD_`. This is a precaution against collisions with other user–defined variables.

The command `FILE` does not yet accept DD–names. It is of course possible to save a file that has been loaded via `load (abc)` using `file`, because *exaEdit* knows which file it is. But if you try something like `file (abc)`, *exaEdit* will create a file called '`(abc)`'.

3.1.3.3 Files with Special Record Formats

exaEdit usually operates on conventional text files consisting of records that are separated by a Linefeed character (`x0A`) or by the two characters Carriage Return (`x0D`) and Linefeed. (Such characters are customary but not needed for reading the file. *exaEdit* writes them nonetheless.)

There are other ways of formatting records, especially in operating systems aside from Unix or Windows. *exaEdit* is able to read one of those: Records of a variable length, which is encoded in a 2–byte–field at the beginning of the record. The length field itself is not included in the length count.

When reading such files, *exaEdit* analyses the length field and assigns the read data to records according to the length specifications. A workfile thus created can only be saved as a conventional text file (see above) at the moment.

The command

```
WIDTH
```

is another alternative to reading data by separating via x0A. For further information see the description of that command.

3.1.3.4 Parameters for Large Files

Especially when dealing with very large files, it may be beneficial to load only a part of the file instead of all of it. LOAD can take three parameters for this purpose:

```
COUNT
```

counts how many records the file contains, but does not load it at all. At the same time, it is calculated how large in bytes the workfile would have to be. This number is larger than the file size on data medium, because *exaEdit* needs several bytes of control information for each workfile. The calculated number is, on the other hand, smaller than the number of bytes the program *exaEdit* with the loaded data would need in the computer's main memory. COUNT generates this message:

```
Records counted: ..., size of workfile: ...
```

The size is measured in B, KB, MB or GB. The message also shows which unit is used.

If you also use the parameter MULTIPLE, the parameter COUNT must appear prior to MULTIPLE, unless you use MULTIPLE with the sub–parameter /string/.

If you want to load the first n records of a file only, you can use the parameter

```
RECORDS n
```

For skipping the first n records of file, you can use the parameter

```
IGNORE n
```

Of course you can use these two parameters together, also in conjunction with COUNT. It is not possible, however, to use IGNORE and RECORDS together with MULTIPLE.

To get the help text for LOAD that deals with COUNT, IGNORE and RECORDS, you can use

```
HELP LOADX
```

3.1.3.5 Loading All Files of a Directory

exaEdit offers the possibility to load all files of a directory at once into one workfile, to make changes there and to write all files back into the directory.

If you wish to use this multiple loading, it is only possible with the *exaEdit* command LOAD, not with the calling of *exaEdit*. The parameter needed is

```
MULTIPLE
```

Its minimal abbreviation is M.

It is not possible to say anything about the sequence of the individual files when they are loaded, anyway, it is not necessarily an alphabetical order.

If the directory contains subdirectories, they are ignored.

In order to have the information which records belong to which file in the workfile also, a certain separator record is written as the first record of each file. A separator record normally has the form

```
$$$DDD$$$
```

where the file name replaces the characters DDD.

You have the option to use separator records of your own choice when you specify the parameter

```
MULTIPLE /string/
```

at the LOAD command. The choice of the delimiters of the separator string is free, as usual in *exaEdit*. If the character string DDD occurs in the separator string, it will be replaced by the file name. Working without any separator string is possible if you specify an empty character string (//).

If LOAD ... MULTIPLE ... is executed in an empty workfile, *exaEdit* stores the information that the workfile was generated in this way and notes this with the letter M in the status line. Besides, the top line and the status line display the name of the directory instead of a file name. This has an effect on the execution of the command FILE.

The specification of a directory name may be done in the usual way, for example is . (a dot) the name of the current directory.

You might receive one of the following error messages (in addition to those at LOAD):

```
Directory not found
```

That means that you specified MULTIPLE and no object with that name could be found.

```
Object is no directory
```

This means that you specified MULTIPLE, that the object exists (contrasting to the message above) but it is not a directory but probably a file.

```
Directory not opened
```

That means that the directory cannot be processed despite all the previous successful checks. The precise cause cannot be specified.

If data sets of the directory were loaded, you receive the message

```
... file[s] loaded
```

If the directory also contains subdirectories, you will receive the message:

```
... subdirectory/-ies skipped
```

You also have the possibility to load only files with specified names or to exclude files with specified names. The required parameters are

```
SELECT ...      resp.      EXCLUDE ...
```

Both parameters have to appear prior to MULTIPLE, unless MULTIPLE is used with the sub–parameter /string/. In that case, the positioning of the parameters is variable. If one of the two parameters is specified, MULTIPLE is always assumed, too.

How do you specify the file names? The first way is to specify a workfile–name. This workfile then has to contain file names, line by line. The alternative to this is specifying a list of file names, separated by spaces and enclosed in parentheses. An example:

```
SELECT LISTE EXCLUDE (ABC XY)
```

This would load all files of a directory that are mentioned in the workfile LISTE, without the two files ABC and XY. You can, of course, use the parameters SELECT and EXCLUDE separately.

It is not necessary to specify full names for the file names. A question mark (?) designates any single character, an asterisk (*) designates any sequence of characters (including empty sequences), and of any number of characters enclosed in brackets ([]) one has to appear at the given position. It is also possible to specify a range of characters within the brackets: Two characters connected with a minus sign (-) mean any character within the lexical range of the two characters. If you want to use either the minus sign or the closing bracket as a specifying character within the brackets, they have to appear as first or last character within the brackets. You can exclude (a range of) characters by preceding them with an exclamation mark (!), i.e. the file names chosen must not have the specified characters at the given position.

The help text for the command LOAD that deals with EXCLUDE and SELECT is available with

```
HELP LOADX      and
HELP LOADY
```

3.1.4 Saving a File

The saving of a file does not happen automatically; it requires that you give the command

```
file [filename]
```

If you do not specify a file name, the contents of your workfile (without the top line) will be written in the file your workfile is assigned to. You find the name of the respective file in the status line from column 19 onwards or in the top line after the words TOP LINE. If the file name corresponding to your workfile is longer than 41 characters, there will be only its beginning displayed in the status line. In this case, you may have to look it up in the top line to be able to read the full name of the file.

If there is no file name assigned to your workfile, you receive this message:

```
Parameter missing
```

Contrasting, if you specify a file name in the FILE command, the workfile will be written in this file. If the file assigned to your workfile has a different name, it will not be changed.

The file name, both the one that is assigned to the workfile and the one you specified in the FILE command, could be either an absolute or a relative file name.

Now we have to differentiate between Unix and Windows systems.

In Unix systems:

An absolute file name begins with a slash, a relative one does not. If you specify a relative file name, it will be completed by *exaEdit* putting your current working directory in front of it. If you would like to know the name of your current working directory, you may use the following *exaEdit* command:

```
call pwd     or     _pwd
```

This gives you the result of the Unix command pwd (pathname of the working directory). For example: If

```
/u/fmath/ppreus/exaEdit
```

is your current working directory, the following two commands

```
file etc/one     or     file /etc/one
```

write into the following two files:

```
/u/fmath/ppreus/exaEdit/etc/one     or     /etc/one
```

You may also refer to the parent directory by using the common spelling conventions. In the example above, you would use

```
file ../abc/two
```

to write in the file

```
/u/fmath/ppreus/abc/two
```

although your working directory is

```
/u/fmath/ppreus/exaEdit
```

at the moment.

Finally, you could also use the spelling with the tilde (˜) for the file name. In the following example

```
~/...
```

the tilde stands for your HOME directory and

```
~uid/...
```

stands for the HOME directory of `uid`.

In Windows systems:

An absolute file name begins with a backslash ('\') or a drive letter, a relative one does not. If you specify a relative file name, it will be completed to an absolute file name by *exaEdit* putting the name of your current working directory in front of it. If you would like to know the name of your current working directory, you may use the following *exaEdit* command:

`call cd` or `_cd`

This gives you the result of the DOS command `cd`. For example: If

```
d:\pe\dok
```

is your current working directory, the following two commands

`file winnt\win.ini` or `file \winnt\win.ini`

will write the following two files:

`d:\pe\dok\winnt\win.ini` or `d:\winnt\win.ini`

You may also refer to the parent directory by using the common spelling conventions. In the example above, you would use

```
file ..\abc\two
```

to write in the file

```
d:\pe\abc\two
```

although your working directory is

```
d:\pe\dok
```

at the moment.

From here on, it is about **Unix and Windows systems** both.

If the file name you specified in the `FILE` command contains spaces, apostrophes or the command separator, you have to include the file name in apostrophes. In the example

```
file 'a ;'
```

the file name consists in an `a`, one space, and the semicolon (which is your command separator). An apostrophe as part of the file name has to be spelt as two apostrophes in a row.

Before the workfile is actually written in the file, *exaEdit* finds out whether the file already exists or whether it is a new file. Then you receive one of the two following messages:

```
Old data set, press J or Y to replace it:
New data set, press J or Y to create it:
```

These questions are used to reduce the danger of typing errors and file names you did not want to specify.

If you want to continue the saving process as you specified it, you only have to press the key '`J`' or '`Y`' (standing for 'ja' or 'yes'); small letters are sufficient, the `return` key is not necessary. If you press – involuntarily or voluntarily – another key, you receive this message:

```
ATTENTION: Data not saved!
```

and the alarm message beeps if it exists. But if you allowed the saving process to continue and everything has worked without problems, you receive the message

```
Data saved
```

It is important to note that you will only see the latter message if you respond to the request '`press J or Y ...`' with either the `J` or the `Y` key without pressing the `return` key afterwards.

When you save a workfile you may receive one of the following error messages:

```
Data set may be read only
```

The access rights for the file to write in are such that only read is possible.

```
File not found
```

This message appears if *exaEdit* cannot find the file in question. Maybe you made a typing error or the file is in another directory.

```
File system may be read only
```

The access setting for the directory to write in says that you only have admission to read.

```
No Home-directory found for ...
```

This message appears when you have used `~uid` (see above) and the operating system cannot find a home directory for `uid`.

```
No file and no directory
```

This message appears if the object you wanted to save into is neither a file nor a directory. It cannot be edited.

```
No connection to another computer
```

In order to find out whether you have access to the file you specified, the operating system has to ask for access at a computer different from the one you are just working at. But if the other computer or the connection to it do not work properly, you receive the message cited above. It is highly probable, then, that you cannot read or write any file at the moment.

```
Ending ' missing
```

This message occurs if a file name in the `FILE` command begins with an apostrophe (`'`) but *exaEdit* cannot find the corresponding closing apostrophe, and so *exaEdit* does not know which file name is meant. The latter is the case if there is a space in the line anywhere after the opening apostrophe which is followed by still other characters. In other words: If the file name (which you want to enclose in apostrophes) contains no spaces, you may leave out the closing apostrophe.

```
Part of the name is no directory
```

A component of the absolute file name which is not the last one was found not to be a directory but, for example, a file.

```
A directory cannot be edited
```

This message appears if you tried to save in a directory instead in a file. *exaEdit* can only handle files.

```
Directory not found
```

You receive this message if a directory in your file name could not be found.

```
Too many symbolic links, refer to itself?
```

This message appears if the operating system cannot solve the file name, which contains links to other files. The cause is in most cases a circular definition, i.e. a direct or indirect link to itself.

```
Access not allowed
```

This message appears if you are denied access to a file because it belongs to somebody else, for example.

```
access errno = ...
getcwd errno = ...
stat errno = ...
```

and similar messages. These messages should never occur. They appear if certain errors happen, for which *exaEdit* has no special message stored. If you receive one of these messages, you should fix the complete message together with the circumstances of its appearance and send this information to the author of *exaEdit*.

Finally, it may be interesting for you to know that in writing the workfiles back onto the disk any space at the end of each record (including the last one) is taken away and a newline character (`\n`, `x0a`) is added there.

3.1.5 Leaving *exaEdit*

You leave *exaEdit* either with the command `QUIT` or `END`. The two commands are equivalent.

To save you from leaving *exaEdit* involuntarily, *exaEdit* checks whether there are some workfiles left, which have been changed but not saved yet, before the program stops the session. If there is only one workfile in your *exaEdit* session (that will be the `MAIN`), *exaEdit* writes the following message in the window:

```
Changes not saved
Press J or Y to stop:
```

In line mode, the second line reads as follows

```
Enter J or Y to stop:
```

If there is more than one workfile in your *exaEdit* session, the message will read as follows instead:

```
Workfiles not saved: ...
```

– where the three dots will be replaced by the names of the workfiles which have been changed but not saved.

If you think you will do without saving your workfile(s), you only have to hit one of the keys `J` (for 'ja') or `Y` (for 'yes') – and *exaEdit* immediately finishes the session. Please note that you do not need to press the `return` key; one of the keys `J` or `Y` is sufficient. Of course, you do not have to use the upper case letters `J` or `Y`, the single keys are sufficient.

Contrastingly, if you want to save your workfile(s) instead or do not want to quit *exaEdit* straight away, you only have to hit any other key instead of `J` or `Y`. As a result of this, the leaving process is aborted and you find yourself in the normal *exaEdit* session, as usual.

You should also note that a workfile is even regarded as changed if you have done as little change as replacing a character by itself.

In very rare cases, and probably only in Unix systems, *exaEdit* may write a message in the window after the *exaEdit* session has been finished:

```
exaEdit: Escape sequences instead of keys:
```

The facts of the matter are these: The information that one of the special keys, e.g. *right arrow*, has been pressed, sometimes reaches a certain stage in the operating system in the form of a sequence of other keys. So, it may occur, for example, that the *right arrow* key is translated into the sequence of the three keys `Esc[C`. If they reach the operating system with very short intervals, the operating system manages to translate them into the information *right arrow* key. If the time intervals between the individual keys of the sequence are too long – as it may occur in case of highly charged lines between terminal and computer – the keys are passed on as separate keys, which are useless for *exaEdit* in this form. This is the reason why *exaEdit* has a mechanism to complete the single characters to the correct sequence *right arrow* key. All the instances where this mechanism is used are counted and the number of these events is displayed in the window after the actual *exaEdit* session has been finished.

If the displayed number is small, you may ignore the message. But if there appear larger numbers after the end of your session, there might be some problem. *exaEdit* completes any recognized sequence correctly but there might have been some cases that *exaEdit* did not recognize.

3.1.6 Structure and Input of Commands

Commands are always words or compounds taken from the English language or individual special characters. The usage of words as commands is intended to enable you to recall the commands more easily. The use of special characters is reduced to the minimum for the same reason; there are no key combinations where you have to press two keys at once.

On the other hand, there are abbreviations of the commands to let the amount of things to type in not become too overwhelming. So you can use `C` for `CHANGE`, `CO` for `COPY`, `DE` for `DELETE` etc. Besides, you may leave out any space that is not necessary for syntactic reasons.

Between the admitted minimal abbreviation and the full command word any transgressive grades are allowed, so that you may start working with the complete commands as a beginner and then work through to the minimal abbreviations a professional may use. Example: `change chang chan cha ch c` – all these spellings are allowed for the `CHANGE` command.

Commands may have operands (parameters), which are normally separated from the command word and from each other by spaces. As mentioned above, you can leave out the separating spaces if the resulting creation is not ambiguous.

For example, the command `COPY` copies the specified lines (starting with the first specified one to the second specified one, includingly) behind the current line. The `COPY` command will be discussed in detail at another place in the text, the example here is only supposed to demonstrate the varying possibilities for spelling.

```
copy 100 200
```

The minimal abbreviation of `COPY` is `CO`, the space before the specification `100` is not necessary and therefore the shortest spelling is:

```
co100 200
```

As a third operand the name of a workfile can be specified, from which the lines will be copied. For such a name (e.g. `abc`) has to begin with a letter, it is possible to write it directly behind the operand `200` without a space:

```
co100 200abc
```

If the area you want to copy includes the first or the last line of the workfile, you do not have to specify the numbers of those lines. In this case, it will be enough to denote them with their symbolic values `f` (for first line) and `l` (for last line), respectively. So, in this example, you have to write at least:

```
co f l abc
```

here, all spaces are necessary.

It is also possible to enter several commands at once as you will see in the next section.

3.1.7 Concatenating Commands, Command Separator

When editing, you will often have to give a sequence of commands which you know in advance. For example:

- Search for the next record containing the character string `'abc'`,
- starting there, go two records back,

- in that record, change `xyz` to `12`.

The commands that are necessary to do this look like that:

```
locate /abc/
up 2
change /xyz/12/
```

These commands can be concatenated by writing them in one line, separated with the command separator '`;`', and entering them at once with `Enter`:

```
l/abc/;-2;c/xyz/12
```

(Of course, you may also write the full command words – in the same manner as they are listed above.)

One of the advantages of concatenation is that some of the actions *exaEdit* has to execute only have to be made once instead of three times, which results in saving time. Another application of command concatenation can be found in section 3.1.22, *Command Storage*.

The command separator is predefined as the character '`;`'. But this is not the only character possible for this purpose; you may use any other character except the question mark and digits. The command to change or display the character which functions as command separator is `CMDSEP`. Please note, that the command to change the command separator has to be the last command in an input line. Compare section 3.2, *The Commands*, for more information.

On some keyboards, the character '`;`' can be only reached with the `Shift` key. For such a frequently used character, which should serve the efficiency of your working, this may be a bad position. In this case, it would probably be better to use another character, which is reachable directly. Often the comma '`,`' will provide a solution. To avoid the trouble of entering the command

```
cmdsep ,
```

every time you start a *exaEdit* session, you can solve the problem once and for all by putting the command mentioned above in the *exaEdit* profile file. Compare also section 3.1.26, *The Profile Files*.

Please note that you do not have to change the command separator if the character '`;`' is part of a character string in a command. For example, `l/ab` searches for the character string '`ab`'. If you want to concatenate the `LOCATE` command with another command, you only need to write `l/ab/;change ...` The command `l/ab;` would search for the character string '`ab;`'. To concatenate this command with another one, you could write the following version: `l/ab;/;change ...` Delimitating character strings has priority over recognition of the command separator.

3.1.8 Presentation in the Window, Current Line

The editor *exaEdit* works on the records in a file and treats them as lines at the same time (this is an example of a file with three records):

```
*** 1st record of the file ***
2nd record
This is the last record
```

exaEdit uses the whole window at disposal for its work. If you change the window size while you work with *exaEdit* (which unfortunately is not possible in all operating systems), *exaEdit* recognizes this change and adjusts to the new size automatically. If the window is too small or if the operating system cannot use the window mode (i.e. writing in the whole window), *exaEdit* will work in the line mode. You can explicitly ask for the line mode (*exaEdit* command `scope off`), compare section 3.1.20.

exaEdit displays the content or (in most cases) parts of the content of the workfile in the upper part of the window, the so-called data zone. The lower part of the window mainly serves the input of commands (details discussed later) and the output of answers of the editor; it is called dialogue zone. So, a window showing the workfile presented above would look as follows (prerequisite a window with 24 lines).

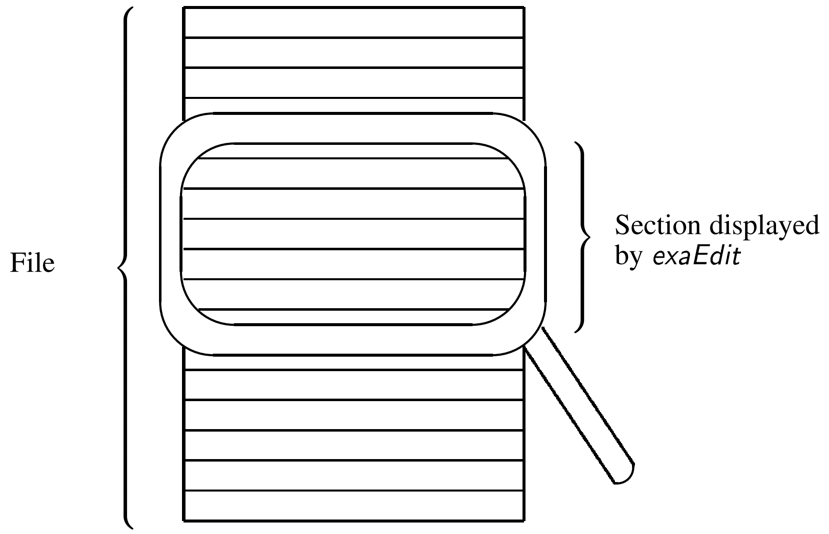
```
          1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
+--------------------------------------------------------------------------------+
 1|                                                                               |
 2|                                                                               |
 3|                                                                               |
 4|                                                                               |
 5|                                                                               |
 6|                                                                               |
 7|MAIN     exaEdit 02B TOP LINE file                                             |
 8|000100 *** 1st record of the file ***                                          |
 9|000200 2nd record                                                              |
10|000300 This is the last record                                                 |
11|                                                                               |
12|                                                                               |
13|                                                                               |
14|                                                                               |
15|                                                                               |
16|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
17|                                                                               |
18|exaEdit                                                                        |
19|_                                                                              |
20|                                                                               |
21|                                                                               |
22|                                                                               |
23|                                                                               |
24|    MAIN          file                                                3 19/ 1 |
+--------------------------------------------------------------------------------+
```

The numbering above (1 – 80) and on the left (1 – 24) and the frame are not visible in the window, they are only used as points of reference in this manual. The data zone, where the records of the workfile (or parts of them) are displayed, contains the lines 1 to 15. In line 16, there is a ruler, which is supposed to make it easier for you to count the columns. Lines 17 to 23 contain the dialogue zone. Line 24 is the status line, in which the so-called workfile name (`MAIN`) is displayed, the name of the file currently worked on (`file`), on the very right border the number of records of the workfile (`3`), and the line and column number of the current cursor position (`19/ 1`).

The following considerations require that each record in the workfile takes only one line in the data zone if the record is visible there. It does not have to be like that necessarily, as you will see later, but this requirement is useful here since it allows us to use the terms line and record interchangeably.

Line 7 contains (as record number zero, so to speak) an additional record of the workfile with the workfile name (`MAIN`), the name of the editor `exaEdit`, the version number of *exaEdit* (`02B`), the identification `TOP LINE` and the name of the file currently worked on (`file`). The top line is generated when a file is loaded into a workfile and it is eliminated when the workfile is written into a file.

If a file has more records than the number of records that would fit into the window (which is normal), *exaEdit* only displays a certain section in the window.



This section moves over the file according to the *exaEdit* commands used. Please try to keep this image in mind: The section moves over the file as a reading magnifier moves over a piece of paper. In contrast to this, the perception that the section remains fixed like a window and the file is pulled behind it is definitely wrong. The *exaEdit* command `+5`, for example, pushes the section five records onwards; this means five records downward in the direction of the end of the file.

The section contains an odd number of lines. This implies that there is a middle line, which is emphasized optically and logically. This line is called 'current line' (to be more precise: the line of the current record). Any command which contains no line specification refers to the current line. For example, you use the command `delete 3` ('delete 3 lines') to delete the record of the current line and the two following records in the workfile.

It is true that the current line is fixed in the window but it is not on a fixed place in the workfile. Every time when the workfile is changed, the current line is moved to the line that was changed last – after the changes are done. The section of the workfile displayed is always positioned in such a way that the place which has probably just been in the centre of your interest is in the middle of the window. In the explanations of each individual command notes are made whether and in which manner the current line is moved. With some exercise you will know where the current line is positioned after the execution of the most of the commands. Above that, you will be able to use this information to your advantage when you are concatenating commands (compare section 3.1.7).

Each record of the workfile has a record number. This number is determined when the file is loaded into the workfile. The numbers are consecutively counted up, the standard counting is `100`, `200`, `300`, ...

The lines in the data zone, that contain the records from the workfile, are structured as a standard as follows:

```
                 11111   77778
column  12345678901234...67890
content nnnnnnfddddddd...ddddd
```

- Column 1 – 6 (`nnnnnn`) is the number area.
- Column 7 (`f`) is the flag field, in which there are marks of certain qualities of the line (details later; in any normal case the field is empty).
- Column 8 – 80 (`dd...dd`) is the data zone.

The width of the number area (default: 6) may be changed (0 to 8 characters); the data zone adjusts its width automatically.

With the values mentioned, there can be records up to 73 characters displayed in the data zone. But often there will be longer records than those fitting in 73 columns. As long as you do not give any other specification, the rests of the records will be displayed in subsequent lines. You recognize these subsequent lines from the empty number area:

```
000100 -------TEN----TWENTY----THIRTY-----FORTY...---SEVENTY---
       -EIGHTY----NINETY---HUNDRED
000200 One record that fits in one line.
```

In this example, there are two records: The record with the number 200, the content of which fits in one line in the window, and the record with the number 100, which has 100 characters and takes two lines in the window.

Since this kind of presentation is not always liked, there is the possibility to define a logical window width (command `LWWIDTH`). If the window width is defined as 110, the example from above will transmute to the following picture:

```
000100 -------TEN----TWENTY----THIRTY-----FORTY...---SEVENTY---
000200 One record that fits in one line.
```

With this method (but not yet in the current version of *exaEdit*) you have the possibility to move the visible part of the window (command `szone`). If you define `szone` as 30, you see the following in the window:

```
000100 -----FORTY...---SEVENTY----EIGHTY----NINETY---HUNDRED
000200 ne.
```

– this means anything from the thirtieth character on.

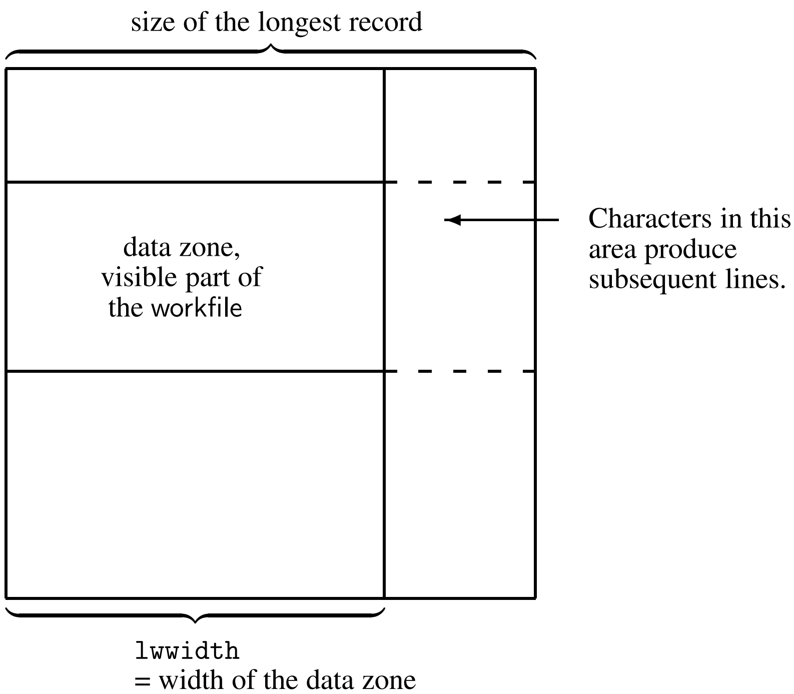
What happens if `LWWIDTH` is defined as 90 (and `szone` is set back to its normal value 1)?

```
000100 -------TEN----TWENTY----THIRTY-----FORTY...---SEVENTY---
       ---HUNDRED
000200 One record that fits in one line.
```

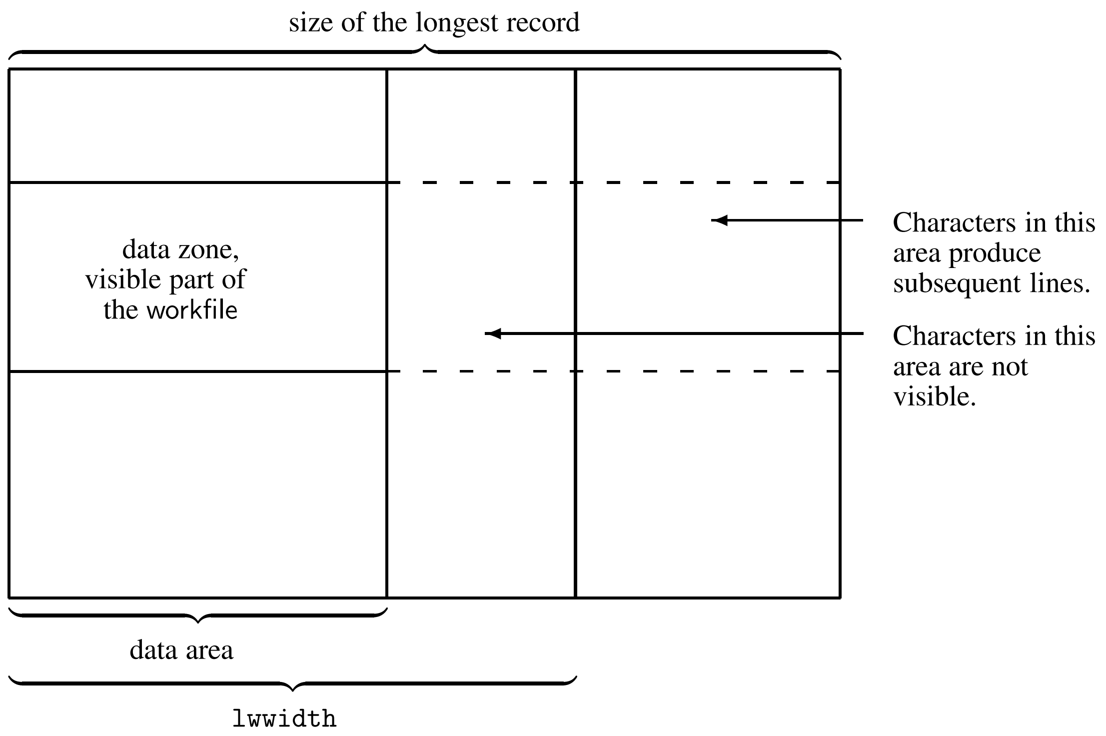
This illustrates that the part which exceeds the value defined in `LWWIDTH` is always displayed in subsequent lines. If you do not want any subsequent lines at all, the value of `LWWIDTH` has to be $\geq$ the longest record.

One more schematic presentation of these facts:

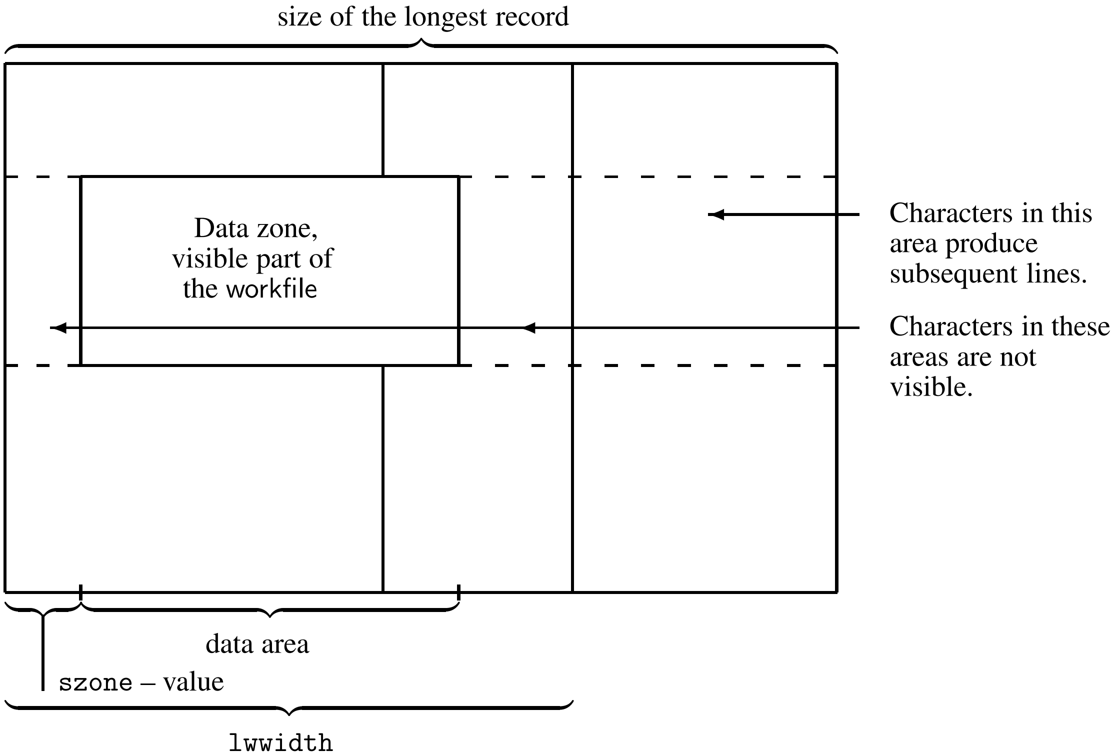
1st case: logical windowwidth (`lwwidth`) = width of the data area,
left margin = 1st character (`szone 1`):



2nd case: logical windowwidth (`lwwidth`) > width of the data area,
left margin = 1st character (`szone 1`):



3rd case: logical windowwidth (`lwwidth`) > width of the data area,
left margin = nth character (`szone n` defined with `n > 1`):
(note: This case is not possible in the current version of *exaEdit*.)



The same rule is valid for the subsequent lines, this means that the specification of `szone` may change the part of these lines which is displayed, and it means that characters beyond the column `LWWIDTH` produce subsequent lines, again.

3.1.9 Record Numbers

All records of a workfile have a number. This number is generated when the file is loaded, but omitted when saving the file. Several commands need the numbers to define a target or working area.

The record numbers run from 0 to 99999999. Normally the first record will have the number 100 and the interval between two record will be 100, too. When editing, lines might be added, deleted or moved. The record numbers will remain strictly ascending. Since the interval between record numbers is set to 100 in the beginning, changes can often be handled within those intervals. Whenever possible, *exaEdit* avoids changing existing record numbers. Only when inevitable, *exaEdit* partially renumbers the records. The interval between two numbers then is 20 (instead of 100). Using small steps like this, it's highly probable that an existing record number that can be kept is reached soon. All following record numbers remain untouched. *exaEdit* reports a renumbering with the message:

```
Renumbered
```

If you want to renumber the records yourself, you can use the command:

```
REKEY
```

For modifying the starting number or the size of the intervals see the description of the command.

Usually, only 6 of the actual 8 digits of the record numbers are displayed (`000100`, `000200` etc.). If a file consisting of more than 10000 or more than 100000 lines is loaded, 7 or 8 digits will be displayed.

You can control this display width by using the command

```
SKEY
```

and its parameters (see command description).

When the display width is not sufficient for showing all significant numbers, the required number of leading digits will be omitted on the display. The * character shows this has happened, e.g. the record number 1234500 could show as:

```
234500*data
```

In most cases you should be able to avoid this situation by choosing suitable values for the display width of the record numbers. This is recommended, because otherwise there might occur misunderstandings between you and *exaEdit*, for example with the line number 234500. If neccessary you must reduce the number intervals by means of `REKEY ....`

3.1.10 Deleting and Inserting of Characters

Generally you may delete any character in the *exaEdit* window, excepted are the top line, the status line, and the number areas in the data zone. Deleting can be usually done by one of two keys.

The `Del` key deletes the character under the cursor. All characters to the right of this place are moved 1 place to the left (sometimes the contents of the next lines also, if they belong to a multi-line displayed record which is not our concern here).

The `Backspace` key, often marked with an arrow to the left, deletes the character before the cursor. As with the other key, all characters to right of this place are moved 1 place to the left.

Inserting characters is a little bit more complicated, because it has to be separated from overwriting of an existing character. It is true that nowadays there are many programs which offer no possibility for overwriting a character but only deleting and inserting, but not so *exaEdit*.

The restrictions for inserting are the same as for deleting a character.

In order to insert a character in the editor window *exaEdit* has to be put into the insert mode (please distinguish that from the input mode). This is accomplished in general by pressing the insert key. When the insert mode holds it is shown in the status line by the character $\wedge$. As long as the insert mode is valid all characters put in will not serve overwriting at the place of the cursor but inserting at this place. All characters to the right are moved 1 place to the right.

In the default behaviour of the editor the insert mode is ended by pressing the return key (or any function key). If you want to end it before, perhaps because the next characters you want to type are meant to overwrite, you must press the insert key again. Prerequisite is that at least 1 character key has been pressed after switching on the insert mode. We call the insert mode described here the ordinary insert mode (in contrast to the insert mode described in the next paragraph).

Now comes the permanent insert mode: If you do not want the insert mode by every return key you may do the following: press the insert key twice (without any other key in-between). If you press during that permanent insert mode the insert key again it will be switched off.

In addition to pressing the insert key there is also the command

```
INSMODE
```

`INSMODE ON` switches to the permanent insert mode (just like the double pressing of the insert key), `INSMODE OFF` switches off the insert mode (just like the single pressing of the insert key). Thus you have the opportunity to insert characters with a keyboard where the insert key is missing. The command `INSMODE` and the insert key are of course usable mutually.

3.1.11 Setting Of and Going To Markers

It will occur quite often that you want to keep in mind a certain position in the file you are editing in order to be able to go back there later. To achieve this, you may use the line number but this method is awkward and it might go wrong if there was a renumbering in the meantime.

A more elegant method is to set markers by using the command

```
set
```

The processing of this command consists in *exaEdit* memorizing the record of the current line.

With the command

```
return
```

you can go back to the marked record from everywhere in your file.

If you give the command `SET` another time, the previous marker is overridden and, again, the current line is marked.

With the command

```
set ?
```

you can inform yourself which record a following `RETURN` command would go back to.

If you give the command `RETURN` without having set a marker in the same workfile with the `SET` command, you receive the message

```
SET storage unused
```

If you have deleted the record with the `SET` marker in the meantime, the commands `SET ?` and `RETURN` provide the previous record. To call your attention to this, you receive in both cases the following message:

```
SET storage changed, return to previous record
```

This message will only then stop to occur when you define a new `SET` marker.

There is an extra `SET` memory for every single workfile. Thus, it is not possible to return to another workfile with `RETURN`.

The line marked by `SET` has the symbolic line number `s`, which you may use effectively in all commands needing a line number.

3.1.12 Positioning

Positioning means the moving of the current line (compare section 3.1.8, *Presentation in the Window, Current Line*). You may position the current line directly or indirectly.

You use indirect positioning when you either change data directly in the data zone or give a command that changes the position of the current line as a side effect.

If you change data directly, the changed line farthest down becomes the current line after you have pressed the `return` key.

In commands that change data, the last line changed becomes the current line. When you delete lines, the previous line becomes the current line. When you insert lines, the last inserted line becomes the current one. The individual explanations of the commands in section 3.2.3 take account of the behaviour of the current line, i.e. whether it is moved and if yes, what the rules are.

There is a number of commands for direct positioning of the current line. Forward, i.e. towards the end of the file, you apply one of these commands:

```
next
down
+
```

– all of them are identical with regard to parameters and execution. The number of records by which you want to move the current line is either 1 (if you do not specify a parameter) or the number `n` you specified in the command. If you go too far, for example, when you give the command `NEXT 5` while the current line is 3 lines in front of the last line, you receive the error message

```
End of data
```

and the current line stays where it is.

If you want to position the current line on the last data line, you use the command

```
bottom
```

Backward, i.e. towards the beginning of the file, you position the current line with one of these commands:

```
back
up
-
```

– these three are all identical with regard to parameter and execution, as well. They work equivalently to the commands for the forward movements; the only difference is that the error message reads

```
Begin of data
```

if you try to go beyond the file.

With the command

```
top
```

you position the current line on the `top line`. If you want to move the current line on the first data line, you may enter

```
top; next
```

You can use the command `POINT` to set the current line to the record given with the command, e.g.

```
point 400
```

So you can reach the first data line very quickly with

```
po f
```

With the commands `LOCATE` and `RLOCATE` you have the option to go to specific data, and you can also position the current line directly. Compare section 3.1.14, *Searching*, for more detail.

With the command `RETURN` you can position the current line on a record you have marked before. You find more details on this subject in section 3.1.11, *Setting and Going To Markers*.

A special kind of positioning is leafing through a file, which will be discussed in the following section.

3.1.13 Leafing Through the File

Related to positioning of a file, which has been described in the section above, is leafing through a file. Basically, it is also a kind of positioning but it partially responds to other rules.

You may jump a whole page or the half of a page forward or backward. There are no *exaEdit* commands for these steps but *exaEdit* functions, instead.

```
'+page'
'+half'
'-page'
'-half'
```

These (and others, compare section 3.1.30) *exaEdit* functions are linked to defined keys on your keyboard when *exaEdit* starts or they may be linked to the F keys with the help of the command PFK.

When *exaEdit* starts, the default setting is as follows:

function	keys
'+page'	F8, PgUp ↓
'-page'	F7, PgDn ↑
'+half'	F11
'-half'	F10

Each of the functions mentioned can be linked to any F key with the command PFK (compare section 3.1.23, *Programmable Function Keys*).

When these *exaEdit* functions are displayed in the window or when they are being processed, the processes are converted to the corresponding *exaEdit* commands + and -, respectively.

Since the leafing through a file is done in such a way that the last line of the previous window becomes the first line of the whole or half window (when you go forward), the conversion results are as follows – prerequisite a window with 24 lines and, thus, 15 lines in the data zone:

function	result
'+page'	+14
'-page'	-14
'+half'	+7
'-half'	-7

This means, for example, that pressing the key F10, which is linked to '+half', has the same effect as giving the command +7.

However, there are some deviations from the equivalent treatment of the *exaEdit* functions and the positioning commands in *exaEdit*. The 1st deviation consists in the fact that when you go beyond the top line or the last data line, the current line is positioned on the top line or on the last line if the *exaEdit* function is used. In contrast to this, *exaEdit* denies the execution of the normal positioning commands and gives the messages Begin of data or End of data when you try to go beyond the data, i.e. the position of the current line is not changed.

The 2nd deviation from the common pattern has only been planned, so far. It includes that too long records which are put in subsequent lines will be considered in future versions of *exaEdit*, meaning the data will be displayed without omissions. In contrast, if the current line has to be moved with +14 by 14 records, it will be moved by 14 records even if there are some data not shown between the data displayed.

The 3rd deviation from the pattern of positioning is that the *exaEdit* command +14 is independent of the window size while the *exaEdit* function '+page' only results in '+14' if a window with 24 or 25 lines is used. This also means that the function '+page' has different values corresponding to different values of the window size.

3.1.14 Searching

'Searching' means looking for and going to records that contain certain character strings or do not contain certain character strings. Commonly, you use the command

locate /*character string*/

to search for a certain character string. The command is explained in detail in section 3.2.

From bottom to top you may search with the command RLOCATE ('reverse locate').

With the commands NLOCATE ('negative locate') and RNLOCATE ('reverse negative locate') or NRLOCATE ('negative reverse locate') you receive the nearest line that does not contain the specified character string.

As the minimal abbreviation for LOCATE is only L and as *exaEdit* memorizes the character string that was searched for last, the repetitive search for the same character string is quite simple: you only have to give the command L. Please keep in mind, that you may define one of the F keys with the LOCATE command, which allows you to find different character strings with once pressing a key (compare section 3.1.30).

The stored character string that has been searched for remains the same for any form of the LOCATE command. So, you only need to give the command RLOCATE to search backwardly for the same character string that you were searching for forwardly in advance.

3.1.15 Changing Data, Survey

There are three different methods to change data

1. direct changes
2. commands
3. prefix commands

3.1.15.1 Direct Changes

This means that the characters in the data area may be changed, deleted or inserted. All the changes only take effect when the return key has been pressed, in other words, only then will the data be changed. It is not possible to insert new lines or to delete lines with direct changes. You need commands or prefix commands to achieve this.

3.1.15.2 Commands

Commands are typed in a command line (commonly beneath the ruler line) and serve

- to change the workfile,
  examples: DELETE, CHANGE,
- to change the display,
  examples: LWWIDTH, SKEY,
- to position the current line,
  examples: NEXT, TOP, LOCATE,
- and many other things.

Commands may be entered in any line, except the status line (i.e. the lowest line), but in most of the cases it is most sensible to type the commands in the line beneath the ruler. This is the place on which the cursor is placed after the return key is pressed.

Section 3.2 provides details on the commands.

3.1.15.3 Prefix Commands

These are special commands which are typed in the number (=prefix) area of a line and which only effect this line. Examples are

d[n]	to delete n lines
dd	to delete a range of lines
i	to insert an empty line
"	to double a line

Section 3.3 provides details.

3.1.15.4 Sequence of Processing

Any changes, direct ones and those done with commands or prefix commands, only have an effect when the `return` key is hit.

You may do as many changes as you like with any kind of the three classes mentioned above. The sequence of processing is like this: The window is checked from top to bottom for lines that have been changed. A line even is regarded as changed if there only was some character replaced by itself. The kind of change is concluded from the position of the line in the window, from its previous and its current content.

A line that was changed and in which there was no part of the workfiles (i.e. beyond the data zone or within the data zone above the first line used or beneath the last line used), is always interpreted as a command.

For any other changed line the following rule is valid: If the number area has the same content as it had before, a direct change was made. If an accepted prefix command is identified in the changed number area, the change is recognized as a prefix command. In advance to this, the data area, which may have been changed, is treated as a direct change. Contrastingly, if there is no valid prefix command in the number area, the whole line is treated as a command.

If the explanations above are too abstract for you, you should carefully work through the examples in the chapter *First Steps*.

3.1.16 Deleting Lines

There are several possibilities to delete lines.

The command `DELETE`, with the minimal abbreviation `DE`, deletes the current line and possibly the following ones if you specify it (compare section 3.2.3).

The command `DELETEL`, with the minimal abbreviation `DL` deletes the lines you specified by indicating their line numbers. Of course, you may also use symbolic line numbers. For example

```
dl p n
```

deletes the current line and the lines immediately above and immediately below the current line.

You may also delete lines with prefix commands. At the moment, the commands `d` and `d`*n* are valid.

Any line, in which you type '`d`' in the number area, will be deleted when you press the `return` key.

The type `d`*n* deletes *n* lines, starting at the line with this marker. You have to mind some particularities when you use the spelling with `d`*n*. These special rules are explained in the section 3.1.15.3.

The prefix command `dd` must occur in two lines. If you then press the return key all records in the marked area will be deleted.

3.1.17 Inserting Lines

There are several possibilities to insert lines. First, those methods are explained that insert lines immediately after the current line. In these cases, you have to position the workfile appropriately in advance.

With the command `INPUT` you put *exaEdit* in the input mode. You can see this from the character '`I`' in the status line and from the ruler (between the data zone and the dialogue zone), which begins in column 1. Any line you enter from now on is inserted above, in the data zone. First, the new lines do not get a line number. They only obtain a number when you leave the input mode. You leave the input mode by pressing the `return` key without having typed something before.

As *exaEdit* recognizes the input of characters at any place of the window, you can reduce the effort of typing things in if you wish to insert similar lines in a row. For example, imagine you would like to insert the two lines

```
23-950320 New command CASE.
22-950320 New command PFK.
```

When you have typed in the first of the two lines and transferred it into the data zone with `return`, you find the following situation in a window with 24 lines:

- The first line you inserted is repeated in line 17 in the window.
- The cursor is at the beginning of line 18, ready for your input.

Instead of typing the second line completely in line 18, you move the cursor to line 17 and change it, so that it looks like the second line. This takes much less effort than typing in the whole second line. Then you enter the input with the `return` key, as usual.

Some other commands you may use to insert lines after the current line are `COPY`, `LOAD` and `MOVE`. They are explained in detail in section 3.2.3.

If you do not like to insert lines after the current line, you may apply the number command and insert lines immediately anywhere in the window, instead. From a formal point of view, the number command looks like a line from the data zone:

```
number data
```

It is interesting to note that the space between the two parts is not necessary. If you have a workfile with the line numbers `100`, `200`, `300`, ... and you give the command

```
120 qwert
```

– you insert the line '`000120 qwert`' by doing so. You would achieve the same result with '`120qwert`', while '`120   qwert`' results in the line '`000120   qwert`'.

The number command is most effective if you do not have to type it completely but have the occasion to use something that already is there in your file. Imagine, you have the workfile

```
000100 New command CASE.
000200 anything
```

and you would like the text '`New command PFK.`' to be the second line between the two other lines. The most simple way to do this is to move the cursor in the data zone to the line `100`, change the number there to `000150`, change the word `CASE` in the text to `PFK`, and enter everything with the `return` key. No need to worry, the overwritten line `100` reappears and after it the intended line `150`. With this method you use two *exaEdit* qualities at once: first, the number command and second, the fact that you are allowed to give input at any place in the window (including the data zone).

Of course, you have to be careful that you choose the right number. If you use a number that already exists, the line will be overwritten. May be that is your intention but that subject does not belong any more to the section *Inserting Lines*.

If you would like to apply the number command but the intervals of the numbers are too small, you can restore the 100 intervals with the command `REKEY`.

3.1.18 Features of the Input Mode

As you have seen in section 3.1.17, *Inserting Lines*, you can switch *exaEdit* to the input mode and thereby insert new lines after the current line of the workfile. In the input mode which you start with the command `INPUT` the editor has additional features which are beneficial especially for the input of text or programs.

3.1.18.1 Automatic Indenting

After the input of a line the cursor is positioned again for the next one. The default setting of the editor is not always the first column but that column, which has been the first non-empty column in the line before. The "line before" is the current line when starting the input mode and it is afterwards the line put in just before during the input mode. The indent column is also kept when empty lines are put in. The automatic indenting is especially beneficial when entering programs.

You may control the indenting with the command

```
INDENT
```

It specifies if automatic indenting should occur or not. Furthermore you can specify if the indent column should depend on the previous line as described or if it should have a fixed value. More details on the syntax you can find in the corresponding part of the section 3.2, *The Commands*.

3.1.18.2 Automatic Line Break

If the input of text comprises several lines it is useful when the editor completes a filled line and starts a new one all by itself. Thus you may concentrate yourself fully to the text to be entered. Starting with version 02 of *exaEdit* this is the default behaviour of the editor. Default are lines with the maximum length `LWWIDTH`. `LWWIDTH` is the visible data width in the data zone, for example 73 with an editor window of 80 columns and a number area of 6 digits.

As soon as 73 characters are entered *exaEdit* searches from left to right for the first group of blanks and adds all characters before that group to the first input line which is entered at once. All characters after that group are put at the start of the next input line which will be continued with the characters entered thereafter.

You may control the indenting with the command

```
INLENGTH
```

It specifies if automatic line break should occur or not. Furthermore you can specify if the line break column should be at the end of the visible line as described or if it should have a fixed value. More details on the syntax you can find in the corresponding part of the section 3.2, *The Commands*.

3.1.19 Editing Blocks

exaEdit knows of three commands (`CCOPY`, `CMOVE`, `CDELETE`) which can copy, move, or delete columns (`CCOPY` = column copy).

Since these commands can be used on the same columns in several consecutive lines, it is possible to copy, move or delete ‘rectangles’ of characters in a workfile with a single command. If, for example, you have the workfile:

```
000100 Here is
000200 a block
000300 of text
000400 =======
```

you can use the command

```
ccopy (f l) 1 7 column 9 line 400
```

to make it look like this:

```
000100 Here is
000200 a block
000300 of text
000400 ======= Here is
000500          a block
000600          of text
000700          =======
```

The meaning of the command is: From the first to the last line of the workfile, copy the columns 1 to 7 to the columns 9 and following of the lines 400 and following.

You can find details on the three commands at their descriptions in section 3.2.3.

3.1.20 The Line Mode

Regarding the use of the window, *exaEdit* offers two manners: the window mode and the line mode. The normal state is the window mode, in which *exaEdit* recognizes the input from the whole window and writes into the whole window for the output.

In contrast to this, in the line mode the window is used like a typewriter terminal where you and *exaEdit* can only add new lines at the end of the text you have written so far.

exaEdit uses the line mode only if the window mode is not possible or if you asked for the line mode with the command

```
scope off
```

You can go back to the window mode with the command

```
scope on
```

– while `scope` alone has the same effect as the latter.

The line mode may be of advantage if you want to display something that does not fit into the 7 or 8 lines of the dialogue zone, e.g. the complete output of the command `HELP`. In such cases you can use the commands `scope off; help` and return to the more usual window mode with `scope on` later.

Another usage is to end and restart the window mode with `scope off; scope on`. This helps when the operating system got confused with the display properties (which unfortunately does happen from time to time on some systems).

A useful command for the line mode is the command

```
display ...
```

with which you ask *exaEdit* to display the number of lines you specified, beginning with the current line.

3.1.21 Programming the Editor

Programming the editor includes all the possibilities allowing to have complicated commands or sequences of commands executed with a few hits of keys. *exaEdit* has a lot of features that make this programming possible. Other functions will be added in the course of time. But *exaEdit* will not reach the flexibility of a real programming language; that would not be reasonable because there are enough other, very useful programs for that purpose. What does *exaEdit* offer for this field?

- The programmable function keys `F1`, `F2`, ..., compare section 3.1.23.
- The command storages `X` and `Y`, compare section 3.1.22.
- The command sequence workfile `EXEC`, compare section 3.1.24.
- The parameter variables, compare section 3.1.25.

3.1.22 Command Storage

exaEdit can store commands or commands that are concatenated with the command separator (compare section 3.1.7) to allow you to fetch them conveniently whenever you need them.

exaEdit knows two kinds of command storage. First, there are the F keys explained in section 3.1.23, and, second, the two *exaEdit* commands X and Y which are explained below. The essential difference between the F keys and the commands is that you may use the command sequence saved as X or Y within another command sequence as command X and Y, again, while the F keys can only be used separately.

You set the command storage X with the command

```
X command sequence
```

For example:

```
x next 2; change /ab/xy/
```

For the execution of this command it is sufficient to give the command X. You will often wish to execute a command repeatedly. To do this, you only have to specify the number of repetitions behind the command.

```
x 17
```

executes the command sequence 17 times.

If a command cannot be executed, for example because the end of the workfile is reached, then the execution of X is aborted. This would allow you to use, for example, the predefined X from above with X 9999 even if the workfile does not contain as many records as would be necessary at first sight.

With the command

```
x ?
```

you achieve that *exaEdit* displays the command sequence stored in X in the window. This is very useful if you have defined a highly complex command sequence and you want to change it only slightly. In this case you enter X ?, change the line in the window and hit the return key. By doing so, you have redefined (and changed) X.

The command Y is identical with X. In the definition of X you may use Y and vice versa. A directly or indirectly recursive definition is recognized by *exaEdit* in the execution of the commands and results in one of the following two messages:

```
Cancelled at recursive X
Cancelled at recursive Y
```

You can find further details in section 3.2.3 at the description of X or Y.

3.1.23 Programmable Function Keys

exaEdit allows to reserve the F keys (labelled with F1, F2, ...) for commands or functions.

Usually, keyboards have the F keys F1 to F12. Regardless of how many such keys there are, pressing any of them has to be registered as an action and sent to the program *exaEdit* to make them available for *exaEdit*. Unfortunately, this is not always the case. You can test whether the F keys are available. The keys F7, F8, F10, and F11 have to result in one of the *exaEdit* commands to browse half of or a full page (e.g. -7 for F7 in a window with 24 lines). The remaining F keys result in the message:

```
F-key is not defined
```

If you do not see any of these responses after pressing an F key, that key is not available for *exaEdit*.

First, a simple example: You want to change in a workfile in every tenth line the digit 3 to a 7. For this purpose you could give the command line

```
next 10; change /3/7/
```

as often as you need it. A facilitation for this are the command storages X and Y (compare the previous section or section 3.2.3, *The Commands in Detail*). But it is even more easy to reserve one of the F keys for the command sequence, for example the key F1. To do this, you have to type the line

```
n10;c/3/7
```

and press the F1 key afterwards, instead of the return key. As a result of your success, you receive the message

```
F-key now defined
```

Every time you press the key F1, the stored command line is executed.

The contents of F keys are not particularly protected. This means that a F key receives a new content if you enter something and press the F key again.

The contents of the F keys are the same for any workfile of your *exaEdit* session. When you leave *exaEdit*, they are lost.

Besides the easy usage of the F keys described above, there are more possibilities, for which you need the command PFK ('program function key'). PFK is explained in detail in section 3.2.3, *The Commands in Detail*; here there is only the most essential information.

With

```
pfk n ?
```

you see the content of the F key n. If you omit the number n, you see the content of any F key with some content. The question mark is also dispensable.

With

```
pfk n lock
```

you protect the content of the F key n. This means that it will not be definable by entering commands and pressing this key. This protects the F key from involuntary setting. You can set it back to the unprotected state with

```
pfk n unlock
```

You can also set the F key n with the command PFK.

```
pfk n set /.../
```

Between the dashes, which may be replaced by any other separator, as usual, there is the content of the key. When you use PFK SET, the key is put into the protected mode automatically.

The defining of F keys with PFK SET is the only possibility to combine *exaEdit* functions (compare section 3.1.30) with F keys. For example, if the Entf or Del key is missing on your keyboard or it is not handed over to *exaEdit* or it has another function, then you can define its original function, namely to delete the character behind the cursor, by using

```
pfk 1 set /'del'/
```

to the key F1.

3.1.24 Command Sequences in the Workfile: EXEC

As you may already know, you can store *exaEdit* commands in the *exaEdit* profile files and then the commands are executed when *exaEdit* or a new workfile is started (compare section 3.1.26). But this execution happens only once and only before the file is loaded. If you want to have prepared sequences of *exaEdit* commands executed, you may use the command storages X and Y and the programmable function keys F1, F2, . . . , but the amount of executable commands is very limited. An expansion of this possibility works as follows:

1. Create a workfile with the name EXEC.
2. Fill this workfile with the commands you want to have executed.
3. Go back to the other workfile where you actually want to edit a text.

4. Call the command EXEC.

Then, all characters of the workfile EXEC are executed in a row in the current workfile.

You should note the following particularities:

- After the command EXEC, nothing else may appear in a *exaEdit* command line (this restriction will be abandoned in future *exaEdit* versions).
- If there is no workfile EXEC, you receive the error message

  ```
  Workfile not found
  ```

- If the workfile EXEC contains the command FILE, the usual questions of the kind

  ```
  Old data set, press J or Y to replace it:
  ```

  etc. do not appear. You should be very careful in this case and precisely check the file names you use.

You can find further details at the description of the command EXEC in section 3.2.3, *The Commands in Detail*.

3.1.25 Parameter Variables

Parameters in *exaEdit* commands are at first constants such as 6 or /abc/ in the commands NEXT 6 or LOCATE /abc/. In the place of such constants you may also use variables, they are called parameter variables.

There are three types of parameter variables:

- Type N (numerical). The value of the variable is a whole number.
- Type L (line number). The value is a line number.
- Type S (string). The value is a character string.

Some parameter variables are pre-defined. They get their value by the execution of certain commands:

- &Col is of type N and is set by the commands LOCATE and RLOCATE as the variable indicates the column in which the search argument was found.
- &Count is of type N and contains the result of the execution of the command COUNT.
- &Line is of type L and is set by the commands of the LOCATE family. The value is the line number where the search stops.
- &Loc is of type S and is set by the commands of the LOCATE family. The value is the search argument.

Besides these parameter variables which are always present you may define own variables as you want. How this is accomplished you can find in section 3.2.3, *The Commands in Detail*, under &.

The parameter variables allow you in a certain way to "calculate", for example you may add constants or build substrings. These features are also described in the named section.

3.1.26 The Profile Files

Although *exaEdit* has default settings for most of the parameters, which are most sensible in the most frequent cases, there will occur situations, in which you would prefer other default settings. To solve this problem there are profile files, i.e. files containing *exaEdit* commands that are executed when you start *exaEdit*. Strictly speaking, the profile commands are even executed before the loading of the file you wish to edit happens.

There are two kinds of profile files for *exaEdit*:

- the installation profile file
- the private profile file

The installation profile is named such because it is in effect for the installed editor, no matter by whom it is called (on multi–user–systems). The private profile on the other hand depends on the user identification which it is called from. In detail, *exaEdit* acts like this:

exaEdit finds its installation profile via environment variable, in this case called

```
EXAEDITIP
```

To test whether this environment variable exists and which value it has, you may use the Unix command

```
echo $EXAEDITIP
```

and in Windows systems you apply the command

```
set exaeditip
```

If the value of `EXAEDITIP` begins with the character '`-`', it means that there is no profile file.

But if the environment variable `EXAEDITIP` does not exist at all, *exaEdit* uses the file `.exaeditip` in the directory *exaEdit* has been called from. If this file does not exist, *exaEdit* works without the installation profile.

You create the private profile file on your own when you need it. The file is called

```
.exaeditpp
```

and is searched for by *exaEdit* in the directory from which *exaEdit* has been called. If this file does not exist there, *exaEdit* searches for it in your HOME directory. If there is no such file either, *exaEdit* works without a private profile.

exaEdit finds the home directory in Unix systems via the environment variable `HOME`, in Windows systems via the two environment variables `HOMEDRIVE` and `HOMEPATH`. The values of these you may determine as described above with the commands `echo` or `set`.

exaEdit first processes the installation profile and then the private one, which allows you to override commands defined in the installation profile you do not like with your own corresponding commands in your private profile.

A useful example for an entry in the profile file is the command

```
cmdsep ,
```

which changes the command separator for the concatenation of *exaEdit* commands from '`;`' to '`,`'. This change is an advantage if the character '`;`' can be only produced with the help of the `shift` key and, in opposition to this, the character '`,`' can be typed without the `shift` key.

When *exaEdit* gets started and looks for the profile files in different ways (as explained above), the program stores the current situation without informing you directly: the processing of the profile is supposed to happen noiseless. However, you can inform yourself whenever you want during the *exaEdit* session how the profiles work at the start of *exaEdit*. You receive this information by calling the command

```
profile ?
```

The messages *exaEdit* presents are supposed to explain themselves. A short version of the actions *exaEdit* generally tries is available with the command

```
help profilex
```

`profilex` is only a help text, not a command. The functions of the command `PROFILE` are described further down.

Some of the preferences for *exaEdit* are workfile specific, e.g. the width of the number area you set with the command `SKEY`. You can always put such a command in a *exaEdit* profile, so it is executed every time when *exaEdit* is started. But if you then open an additional workfile the command `SKEY` works with its standard preference again. To make sure the *exaEdit* profile is acknowledged in those cases, too, you can precede *exaEdit* command lines in the profile with

```
!
```

Lines marked such will be evaluated just like the other lines when *exaEdit* is started. But whenever you open a new workfile or delete a workfile completely with the command

```
delete all
```

all command lines in the profile beginning with an exclamation mark will be evaluated again.

With the command `profile` and suitable parameters you can

- display
- execute
- load into the active workfile

the command lines of the profile.

When doing so, you can specify whether you want all lines to be processed, or only the ones beginning with an exclamation mark. Details on how you have to use the command `profile` to achieve this can be found by typing `help profile` or at the extensive description of all *exaEdit* commands in section 3.2.3.

As already mentioned at the appropriate places, the commands of the profile file are executed before the loading of the file or when creating a new workfile. But sometimes it may be an advantage to have a record of *exaEdit* command executed at any time. This is possible; not with a *exaEdit* profile but with the *exaEdit* command `EXEC`.

3.1.27 Online Help

exaEdit offers two kinds of online help: first, the short help texts you get with the *exaEdit* command

```
help
```

(compare section 3.2.3, *The Commands in Detail*) and, second, the kinds for help on the screen that display the manual at hand. This section only deals with the second type of help texts.

The *exaEdit* manual is available in four different formats:

DVI
HTML
PDF
Postscript

If you need the manual but cannot find it, ask the person who has installed *exaEdit* on your workstation or turn to the author of *exaEdit*. Of course you can also find it via the homepage of *exaEdit* in the WWW.

As the entire manual does not fit into the program *exaEdit*, it is a separate file which does not necessarily have to be available on a workstation containing *exaEdit*. This may occur, for example, when it was forgotten to put the manual file in an adequate directory. Another problem results from the fact that looking at the manual on the screen needs other programs, which do not belong to *exaEdit*, and which are not available on every workstation. In addition to this, these other programs need different kinds of representation of the manual.

exaEdit has the command

```
manual
```

for looking at the user manual on the screen and uses an external program, a set of parameters (optionally), and a file to do this. After you entered the command, the program together with the parameters is unleashed on the file.

Since, as already mentioned, different programs, files and parameters are possible, *exaEdit* affixes names to sets containing items that go together.

exaEdit always has one such `manual` set predefined. When using it, you will get the *exaEdit* manual that is available in the WWW. After certain types of installation, a second `manual` set is made available. This is then linked to a local copy of the manual.

You can always change or extend the ways how the manuals are called, either in a running *exaEdit* session, or in the *exaEdit* profile file. The command

```
manual * ?
```

gives a list of all current `manual` sets. Detailed further information can be found at the description of the command `MANUAL`.

3.1.28 The Keyboard

exaEdit does not use key combinations for editor functions. Beside the normal character keys the following keys have a meaning:

- The cursor keys (↑ → ← ↓):
  They move the cursor across the window.
- The `backspace` key (⊲—):
  It moves similar to the cursor key ←, but deletes the character on the left of the cursor.
- The `return` key (⊲—┘, `enter`):
  It serves the input of commands, prefix commands and direct changes.
- The key `INS` (or `Einfg` or â):
  It serves to switch the insert mode on or off.
- The key `DEL` (or `Entf` or a̸):
  It serves to delete a character.
- The key `HOME` (or `Pos 1`):
  It serves to undo all typed changes since the `return` key has been pressed the last time.
- The key `PgDn`↓:
  It serves to jump forward in the workfile by one page in the window.
- The key `PgUp`↑:
  It serves to jump backward in the workfile by one page in the window.
- The `Fn` keys:
  Of these only the following ones have a meaning:
  - `F7`: It serves to jump back in the workfile by a whole page.

- F8: It serves to jump forward in the workfile by a whole page.
- F10: It serves to jump backward in the workfile by half a page.
- F11: It serves to jump forward in the workfile by half a page.

- The keys at the key block on the right:

  They are used by some keyboards as an alternative to the Fn keys.

Often there are keyboards, on which not all of the keys mentioned above are existing. Sometimes, however, they are existing physically but logically they are supplied with values that are not recognizable or receivable for *exaEdit*. In such a case you can link some of the functions mentioned above with some of the existing and functioning F keys.

For example, reasonable editing is not possible without the keys to insert or delete characters (Einfg/INS and Entf/DEL). If the F keys work at least, you may connect the two key functions with the command PFK to F keys. For more detail, compare section 3.1.23, *Programmable Function keys*, and the explanations on the command PFK, section 3.2.3, *The Commands in Detail*. Please, note that you may arrange these definitions into the *exaEdit* profile file (compare section 3.1.26) as well.

Information on which functions can be linked to F keys can be found in the section 3.1.30, *exaEdit Functions*.

3.1.29 Keyboard Test

This section provides information on the situation in Unix systems, the instructions regarding the *exaEdit* command KEYBOARD TEST are also valid for Windows systems, however.

Keyboards are one of the saddest chapters in Unix operating systems. In the struggle to respond flexibly to any realizable and unrealizable wish, the allocation of keys and characters or functions was designed variably on several levels. This raised the possibility to produce a giant chaos and, sometimes, this occasion was taken with at least partial success.

If you hit a key on the keyboard and *exaEdit* responds to it in a particular way, there are besides the two natural levels of change in meaning, i.e. labels on the key on the one hand and interpretation of the key by *exaEdit* on the other, seven (7) other levels, at which the meaning of all or some keys may be altered.

To make sure that the keys you press have the meaning you wish, you may use the command KEYBOARD with the parameter TEST.

However, you have to mind the keys which are not handed over to *exaEdit*. If you think that these keys should be available for *exaEdit*, you have to turn to the person who is responsible for the operating system of your workstation.

Testing a key works as follows: You enter the command

```
keyboard test
```

(minimal abbreviation KEYB T). Afterwards *exaEdit* writes this in your window:

```
Press key:
```

Then you press the key you want to test. If *exaEdit* answers this in one line, the test is finished. If you do not see a reaction of *exaEdit*, you have to hit the return key. If you want to test more than one key, it may be an advantage for you to abbreviate the command KEYBOARD TEST with X or Y (compare the respective commands) or you may define the command by means of PFK (compare the corresponding section) to a F key.

To be able to interpret the information provided by KEYBOARD TEST correctly and to be able to carry out the corrections on the keyboard, you should know the following facts:

In the operating system Unix, normally you do not use a physical terminal but a logical one, a so-called terminal emulation, a logical terminal. Information on which terminal emulation you are using is in the environment variable TERM, which you may have displayed with the Unix command echo $TERM.

Every key which is physically available may be used alone or together with the Shift, Alt, Ctrl, ... key. The following text uses 'key' for single keys and such combination keys. The terminal emulation used decides on what the keys mean. Please note that there are keys, the information of which does not go to the terminal emulation because the keys are caught in advance.

The keys which are handed over to the terminal emulation are passed on in two ways by the terminal emulation: either as normal keys (with a character assigned to it) or as a sequence of keys (often called escape sequence for they begin with the Esc key). Example: If you press the key labelled F1, the terminal emulation xterm asserts you have pressed the five keys

```
\E [ 1 1 ~
```

in this sequence (\E means the Esc key). Other terminal emulations provide different key sequences.

To even out the nonsense, there is a so-called terminfo file in the Unix operating systems. This file contains rules for every terminal emulation which is used at the workstation how to compose the original information from the key sequences. For example, in the terminfo file for the terminal emulation xterm is the rule that the key sequence \E[11~ means that the key F1 has been pressed.

exaEdit uses the operating system supplement Curses (compare section 3.1.1, *Starting a exaEdit Session*). Curses passes on the information to *exaEdit*, which Curses receives from terminfo. Now you have got on far enough to understand the information KEYBOARD TEST provides. The response of *exaEdit* always has this form:

```
Curses: ..., Characters: ..., Escape: ..., Function: ...
```

In this pattern, for ... there are always made entries. These entries have the following meaning:

Curses: Here is the numerical value of a (character) key or the function Curses receives from Terminfo. If there is a dash behind curses:, this means that *exaEdit* has read an escape sequence which does not exist in terminfo. This means that a row of single characters has been passed on by Curses to *exaEdit*. In this case, the third category 'escape:' was used and filled in.

Characters: Here is the character which is connected to the key if there is such a character. Else, there is the word 'none'.

Escape: Here is the escape sequence (compare the explanation on 'curses:') if such a sequence was passed on.

Function: Here, there is the word 'none' if it deals with a character key. If any other key was passed through, it has either a function in *exaEdit* (most of the cases) or the output is '?'. If the key which was pressed has a function in *exaEdit*, the name of the function is the output in the window.

If you want to work with KEYBOARD TEST, you should call the command for some keys which are known to you and which work before you try to find out something about unknown keys.

The persons who are responsible for the maintenance of your workstation can change the terminfo file and, by this means, supply missing combinations of escape sequences or correct mismatches between escape sequences and function keys.

3.1.30 *exaEdit* Functions

As explained in section 3.1.28, *The Keyboard*, there are several key functions that may be linked to F keys (by means of PFK). Originally, all these functions existed as real keys on some keyboards but not on every keyboard. So, the wish for substitution came up. However, in some cases the correlation with real keys is not obvious any more and, therefore, these functions are called

exaEdit functions

In the following table you find a description of the *exaEdit* functions. In the column 'name' is the form you need to apply together with the command PFK. In the column 'key', there are the keys which normally should have corresponding functions. If there are F keys in this column, they are the allocations which *exaEdit* makes at the beginning of every new *exaEdit* session.

name	key(s)	function
‘del’	DEL	delete character at the place of the cursor
‘ins’	INS	insert mode on or off
‘+page’	F8, PgDn ↓	whole page forward
‘-page’	F7, PgUp ↑	whole page backward
‘+half’	F11	half a page forward
‘-half’	F10	half a page backward
‘cleft’	←	cursor to the left
‘cright’	→	cursor to the right
‘cup’	↑	cursor upward
‘cdown’	↓	cursor downward
‘pos1’	Pos1, Home	restore picture after the last `return`

The command

```
help function
```

shows a short help message in the usual format, allowing you to find an *exaEdit* function without consulting the user's manual.

3.1.31 Inserting Record Numbers

For records that are inserted between two existing numbers, *exaEdit* creates a new number that lies in the interval between the existing ones. As mentioned in section 3.1.9, *Record Numbers*, this serves to ensure the strict ascendancy of the record numbers.

Usually, the interval between two records will be 100 and both record numbers will also be divisible by 100, for example:

```
800 ...
900 ...
```

Inserting one new record will create a record with the number 820:

```
800 ...
820 new record
900 ...
```

Further records that are inserted will receive the numbers 840, 860 and 880. When even more records are inserted, they will receive suitable numbers between 880 and 900.

If the standard intervals are not 100, the insertion intervals also will not be 20, but an interval suitable in comparison with the standard interval. In all of these cases, *exaEdit* tries to achieve the following:

- Use ‘round’ numbers rather than ‘weird’ ones.
- When creating a new number, choose the lower half of the existing interval instead of the middle. The reason for this is the assumption that further inserted records will more likely follow and not precede the new record.
- Do not renumber as long as it is possible to insert new record numbers between existing ones.
- When renumbering is needed, do not use the standard interval, but a smaller one (e.g. 20 instead of 100 or 100 instead of 500). This way, not all of the following existing record numbers have to be renumbered.

- If a renumbering would result in record numbers larger than the permitted maximum of 99999999, avoid this by choosing smaller intervals and choosing a lower record number as a starting point for the renumbering. Thus the largest needed number will not exceed the maximum (only partially realized in *exaEdit* version 01).

3.1.32 *exaEdit* Errors

It is very difficult to write *exaEdit*, as any large program, without errors. Unfortunately, one has to live with the danger of an unexpected abortion of the program. To save as much as possible of the work you have done, *exaEdit* tries to write the content of the altered workfiles in new files on the disk. In particular, the following things happen:

exaEdit leaves the window mode and performs the rest of its jobs in the line mode. First, there appears one of these messages:

```
exaEdit: Bus error
exaEdit: End process
exaEdit: Illegal instruction
exaEdit: Segmentation fault
```

Second, a file `exaEdit.dmp` is opened, in which *exaEdit* is supposed to write information that could be necessary for the search for mistakes. Success or failure of this task lead to the message

```
exaEdit.dmp [not] opened
```

The content of the file `exaEdit.dmp` cannot be predicted. It depends on the progress of *exaEdit* and on the place, at which the program *exaEdit* was aborted.

To save the changed workfiles, *exaEdit* checks all workfiles on whether they were altered but not saved yet. For every such a workfile a file with a certain name is opened. All the records of the workfile are expected to be written in the new file. Success or failure of the opening of the new file leads to the message

```
exaEdit.jjjj.mm.tt-hh.mm.ss.wfn.dsn [not] opened
```

Here, the following abbreviations are used:

`jjjj`	for the year
`mm`	for the month
`tt`	for the day
`hh`	for the hour
`mm`	for the minute
`ss`	for the second
`wfn`	for the name of the workfile
`dsn`	for the name of the file.

If the records of the workfile could be saved successfully, you receive the message:

```
Data saved
```

After every the workfiles have been checked and saved to files when necessary, the protocol file `exaEdit.dmp` is closed (if it was possible to open it). This is indicated by the message

```
exaEdit.dmp closed
```

3.1.33 *exaEdit* Tests

exaEdit contains several mechanisms which allow the person who develops *exaEdit* to collect information on errors. These mechanisms are possibilities to test the program and they are activated or deactivated with the command `TEST`.

The syntax of the command TEST is described in section 3.2.3, *The Commands in Detail* (page 111). The meaning of the parameters is not specified, because the *exaEdit*-user cannot really utilize this command.

3.2 The Commands

3.2.1 Notation

In the following text, the commands will be presented in detail one after the other. The prefix commands are described in section 3.3.

The description of a command begins with the command name in the upper left corner, followed by the syntax of the command. In the same line there is the minimal abbreviation of the command on the right.

If there are other commands with the identical syntax and the same meaning as the command to be explained, the other commands are listed behind the first command and separated with a vertical line ('|'). Example:

`BACK|UP|- [n]` BA|U|-

This means that the command BACK is identical with the commands UP and - (minus sign). The minimal abbreviation of BACK is BA, the minimal abbreviation of UP is U, and there is no abbreviation for -.

To describe the parameters (or operands) of a command, the following notation is used:

square brackets ([ ]): They occur before and behind parameters which may be either specified or left out. In the latter case a default setting takes place, which is precisely described.

Example: BACK [n] means that you may either use 'BACK' or 'BACK n'. In this case, the default setting is 1. For more information, compare the description of the command BACK.

vertical line (|): This character separates equivalent parameters, of which you must not specify more than one.

Example: SKEY [n|?] means that you may enter either 'SKEY n' or 'SKEY ?' (or only 'SKEY') but not 'SKEY n ?'. You find more details in the explanation on the command SKEY.

small letters: Small letters are used for parameters, for which you have to set (numerical) values.

Example: UP [n] specifies by how many records the current line has to be moved upward. To use this specification you have to use a number for n (if you specify a parameter).

capital letters: Capital letters are used for parameters you have to use precisely the way they are described. Of course, you may write them in small letters to enter them. The distinction in this text only serves to convey the information whether or not you have to replace a parameter word with a value.

Please, keep in mind that parameters may be abbreviated as well if the abbreviation is not ambiguous. The minimal abbreviation of parameters is not explained in this text.

3.2.2 Messages

Basically, all messages and error messages a command can produce are listed in the description of the specific command, like this:

message, *exaEdit* message, ...

All these messages appear in the index of this manual, where instead of message the actual message is listed.

Additionally there is the chapter 5, which is an extract of the index limited to the messages only.

The exception of these rules are some messages generated by many commands. Accordingly many entries in the index and the messages chapter would not be all too informative. These messages are explained in the messages chapter without page references, while their index entries refer to the according page in the message chapter and to the current page.

```
Begin column larger than end column
Begin of data
Character string too long
End of data
Number too large
Parameter variable no character string
Parameter variable not defined
Parameter variable not numerical
SET storage unused
SET storage invalid
There is no next record
There is no previous record
There is no column 0
You cannot ... the top line
```

And finally there are some error messages that have reasons so obvious that probably nobody would want to read the complete command description:

```
Command in error: ...
Number 0 not allowed
Operand missing in: ...
Parameter missing
There is no such command
Wrong parameter
```

Therefore their entries in the index and in the message chapter refer to the current page only.

3.2.3 The Commands in Detail

`+|DOWN|NEXT [n]` `+|DO|N`

The pointer to the current line is set downward (in the direction of the end of the workfile) by `n` lines. Since the current line has a fixed position in the window, the text moves upward.

If you do not specify `n`, the value 1 is assumed.

If `n` is greater than the number of records after the current line, *exaEdit* writes

```
End of data
```

in the dialogue zone and the current line remains unchanged.

`-|BACK|UP [n]` `-|BA|U`

The pointer to the current line is set upward (in the direction of the beginning of the workfile) by `n` records. Since the current line has a fixed position in the window, the text moves downward.

If you do not specify `n`, the value 1 is assumed.

If `n` is larger than the number of records (including the top line), *exaEdit* writes

```
Begin of data
```

in the dialogue zone and the current line remains unchanged.

`_|CALL externalcommand` `_|CAL`

The Unix or shell or DOS command ‘`externalcommand`’ is passed on to external execution.

When the external command is finished, *exaEdit* displays the message

```
exaEdit: Press Enter, when you have seen everything
```

if the program was in the window mode, before. When you have pressed the key, *exaEdit* is in the same status as before.

If *exaEdit* was in the line mode before you called the external command, the message

```
exaEdit: External command ended
```

shows the finishing of the external command. Any command that can be called in the normal status of a Unix session or Windows command line is possible.

When you wish to continue using *exaEdit* while an external command is executed in Unix, you add an `&` at the end of the external command, as usual in Unix.

`&name [+NUMBER|+STRING|+LINE|=value|?|-]` `&name`

The command `&` provides definition and handling of parameter variables. What parameter variables are and what you can do with them is described in section 3.1.25, *The Parameter Variables*.

The command must always contain a variable name, `&` alone is wrong. For the parameters `?` and `-` you may specify an `*` instead of a name which means that all parameter variables are meant.

Specifying no parameter at all is the same as specifying `?`: The specified parameter variable is shown. The information you get is

- The type: One of the letters `N`, `L` or `S` for the type (confer to section 3.1.25.)
- The name of the parameter variable.
- The value of the parameter variable.

With the parameter `+NUMBER` you define a new parameter variable for numerical values. The name must consist of 1 to 8 letters. Capital and small letters are distinguished. Please note that the names of the pre-defined standard parameter variables have a large initial letter and small letters otherwise. In order to assign a value to the newly defined parameter variable you will have to recall the command `&` with the parameter `=value`.

The parameters `+STRING` and `+LINE` serve likewise the definition of parameter variables for character strings and record numbers respectively.

With the parameter `=value` you assign a (new) value to an already existing parameter variable. Such a value is either a constant (suitable for the type of the variable) or a certain simple expression.

Expressions allowed for numerical parameter variables are sum and difference of constants. Also for parameter variables representing record numbers you may add or subtract numerical constants which then gives a new record number which lies some records below or above. Parameter variables which represent character strings can be used in an expression in the following ways: They may be concatenated with a character string by using a plus sign or a substring may be built with the parameter sequence `SUBSTR /string/ begin length`.

As in all *exaEdit* commands constants may be replaced by suitable parameter variables. This is valid for the command `&` also. In this way a relatively powerful "arithmetic" with parameters is possible.

With the parameter `-` the specyfied parameter variable is deleted. If you use an `*` all parameter variables will be deleted. Please note however, that the standard parameter variables cannot be deleted.

Some examples (the record numbers of the file are assumed to be 100, 200, ...):

```
&anz+n
&anz =3
&Col=&anz + 4
&Line = 0500
&Line = &Line - 2
&Loc = -abc
&Loc = &Loc + .de.
&Loc = substr&Loc3 2
```

Then the command `&*?` will show the following:

```
N &Col     =7
N &Count   =0
L &Line    =00000300
S &Loc     =/cd/
N &anz     =3
```

`ALIGN [(l1 [l2])] /string/ [MOVEALL][H][I][n|ALL]` AL
or
`ALIGN [(l1 [l2])] col [LEFT|RIGHT [MOVEALL]][n|ALL]` AL

This command aligns the records horizontally.

Lines may be specified explicitly: from line `l1` to line `l2` or from line `l1` to the end of data. They may also be specified implicitly: `n` lines beginning with the current line or all lines of the workfile.

In the first version all those lines of the concerned lines are aligned which contain the character string `string`. They will be aligned in such a manner that `string` is positioned one below the other. The result goes after that line in which `string` is farthest to the right. The reason for that is that all lines may be scrolled to the right but not all to the left. Those characters which are before the `string` stay in its place. If, for example, you have the lines

```
abc.def.de - - [23/Dec/2005...
xyx.ghijkl.at - - [24/Dec/2005...
```

then they look after the command `ALIGN /[/ ALL` like

```
abc.def.de - -    [23/Dec/2005...
xyx.ghijkl.at - - [24/Dec/2005...
```

Additionally you may align the lines as a whole by specifying the parameter `MOVEALL` which provides for moving of the characters before the `string` as well. Therefore the command `ALIGN /[/ MOVEALL ALL` would have given:

```
   abc.def.de - - [23/Dec/2005...
xyx.ghijkl.at - - [24/Dec/2005...
```

In the second version the alignment of the concerned lines is done according to the specified column. If this column contains a blank character and if there are no other parameters given, then all characters after that column are moved to the left in such a way, that the the first non-blank character comes in that column. But if you specify the parameter `RIGHT` then all characters before that column are moved to the right in such a way, that the first non-blank character comes to this column. If you specify the additional parameter `MOVEALL` then the whole line will be moved to the right, that is, also the characters after that column take part in the moving. If you have, for example, the following lines:

```
abc     def
1234567 90
  xyz         ABC
```

then the command `ALIGN 7 ALL` has the result

```
abc   def
1234567 90
  xyz ABC
```

the command `ALIGN 7 RIGHT ALL` has the result

```
    abc def
1234567 90
    xyz       ABC
```

and the command `ALIGN 7 RIGHT MOVEALL` has the result

```
    abc     def
1234567 90
    xyz         ABC
```

`BACK|-|UP [n]` `BA|-|U`

The pointer to the current line is set upward (in the direction of the beginning of the workfile) by `n` records. Since the current line has a fixed position in the window, the text moves downward.

If you do not specify `n`, the value 1 is assumed.

If `n` is larger than the number of records (including the top line) before the current line, *exaEdit* writes

```
Begin of data
```

into the dialogue zone and the current line remains unchanged.

`BOTTOM` `B`

The pointer to the current line is positioned to the last record of the workfile. Since the current line takes a fixed position in the window, the rest of the text slides upward.

`CALL|_ externalcommand` `CAL|_`

The Unix or shell or DOS command '`externalcommand`' is passed on to external execution.

When the external command is finished, *exaEdit* displays the message

```
exaEdit: Press Enter, when you have seen everything
```

if the program was in the window mode, before. When you have pressed the key, *exaEdit* is in the same status as before.

If *exaEdit* was in the line mode before you called the external command, the message

```
exaEdit: External command ended
```

shows the finishing of the external command. Any command that can be called in the normal status of a Unix session or Windows command line is possible.

When you wish to continue using *exaEdit* while an external command is executed in Unix, you add an `&` at the end of the external command, as usual in Unix.

CASE [?|U|M|L] | [?|I|S] CA

The command CASE has two different uses. First the use of CASE with the parameters U, M, and L.

Normally *exaEdit* uses the letters you typed in the way they are: small letters remain small, capital letters remain capitals. This corresponds to the default setting

`case m` or `case l`

(m = 'mixed', l = 'lower'). Contrasting, if you change the setting in *exaEdit* with

```
case u
```

(u = 'upper') all the small letters you entered will be translated to capital letters (a to z, no German umlauts).

Any line you 'touch' by changing a character is transformed from small to capital letters. As long as U is in effect, a U in the status line will indicate this.

Now for CASE with the parameters I and S. The command

```
case i
```

causes the commands CHANGE, SSPLIT, and all the commands of the LOCATE family to act as if they had been called with the parameter I. So the commands will work case insensitive (i.e. they will not distinguish between upper case characters and lower case characters). The command

```
case s
```

restores the default performance of the mentioned commands, which is to work case sensitive.

```
case ?
```

displays the current setting of both switches in the output of one message of each of the following two message pairs:

```
Mixed (lower): without translation to capital letters
Upper: with translation to capital letters (no German umlauts)

Case-sensitive
Case-insensitive
```

When the command CASE is given without any parameters at all, the default values of both switches are restored: `case m` and `case s` are in effect.

CCOPY [(l1 [l2])] c1 [c2] COLUMN [+|-]col [LINE [+|-]num][n] CC

CCOPY means column copy, i.e. it copies columns of a line. It should not be confused with the command COPY, which copies lines.

```
ccopy 5 column 20
```

copies column 5 to column 20 of the current line,

```
ccopy 5 10 col 20
```

copies the columns 5 to 10 to the columns 20 to 25. Note that you specify only the first of the target columns, since the number of columns has been set when specifying the source columns. Copying works as follows: First, the columns behind the target column are moved to the right by the needed amount of columns. Then, the source columns are copied into the target area. This is the same method that is used with the command COPY, which also does not overwrite but inserts the copied data.

Source and target area may not overlap. If they do, you receive the message

```
Source and target area overlap
```

Instead of using absolute column specifications you can use relative ones if you add a leading sign to the `column` specification:

```
ccopy 5 10 c +15
```

has the same effect as the command above.

Until now, the commands were limited to the current line. As usual, you can have the command applied to `n` following lines by specifying a number as last parameter:

```
ccopy 5 10 colu 20 7
```

applies the command to 7 lines. If the workfile does not have sufficient lines, you get the message

```
End of data
```

The methods described until now are executed from the current line on. Alternatively, you can also specify the lines you want to be affected enclosed in parentheses as first parameter:

```
ccopy (500 1200) 5 10 c20
```

This means, `CCOPY` will be applied to the lines 500 to 1200, wherever the current line is. The line numbers specified may also be symbolic line numbers (`t`, `f`, `p`, `*`, `n`, `l`, `b`, `s`). If you specify only one line number, it is assumed the second line number is equal to the first. Please mind the parentheses.

About the last parameter of `CCOPY`: Until now, the target columns have been in the same line as the source columns were, but you can also specify the target lines separately:

```
cc5 10c20 line 700
```

This copies the columns 5 to 10 of the current line to the columns 20 to 25 of line 700.

As with the `COLUMN` specification, you can precede the `LINE` specification with a leading sign, thus making the column specification a relative one:

```
cc5 10c20 l-6
```

If the line given cannot be found, you get the message

```
Target record not found
```

Of course, you can also add the (two alternative) numbers of affected lines:

```
ccopy (500 b) 8 column 9 line 000400
```

Please keep in mind that you cannot specify a target line that does not exist (yet). So, if you want to copy something behind the last line of the workfile, you have to create that (empty) line first.

`CCOPY` allows for whole rectangles of the workfile to be copied to another part of the workfile.

`CDELETE [(l1[l2])] c1 c2 [n|ALL]` CD

`CDELETE` means column delete, i.e. it deletes columns of a line.

```
cdelete 5 5
```

deletes the column 5 of the current line,

```
cdelete 5 10
```

deletes columns 5 to 10 of the current line. When deleting columns, the characters to the right side of the deleted area are moved to the left into the now empty area. Note that even if you want to delete only 1 column, you have to specify both the beginning and ending column of the area to be deleted.

By specifying a number `n` as last parameter you can achieve that the columns in `n` lines starting with the current one are deleted. If there are not enough lines in the workfile, the command terminates and you receive the message:

```
End of data
```

Instead of the number `n` of lines to be deleted you could also specify the parameter `ALL`, which determines that all the specified columns are to be deleted in all lines of the workfile.

Instead of a number in the last parameter you can specify the area of lines for which the deletion is to be executed. To do so the first parameter has to be the line numbers enclosed in parentheses. You can use explicit or symbolic line numbers:

```
cd (f 1200) 5 10
```

This means that the columns 5 to 10 of all lines from the first one to the line number 1200, both lines inclusively, will be deleted.

The specification of line numbers and of an amount of lines exclude each other.

If you specify only one line number within the parentheses, the second number is assumed to equal the first.

`CHANGE [col1 [col2]] /old[/[new[/ [n][A][D][H][I]]]]` C

Related command: `REPLACE`

In the current line the character string '`old`' is replaced by the character string '`new`'.

The basic syntax of the `CHANGE` command is

```
CHANGE /old/new/
```

In this formula you have to put character strings for '`old`' and '`new`'. The character strings have to be embedded in delimiters, which is a '`/`' in this text. As a delimiter, you may use any character except digits or spaces. In practice, the special character on the lower right of your keyboard # proved to be very useful. You only have to choose a different delimiter if this character occurs in the old or new character string.

Abbreviations:

```
CHANGE /old/new
```

is possible if you do not need one of the parameters.

```
CHANGE /old/
```

is sufficient if you do not want to replace the character string but only delete it. If you want to delete a character string and need one of the parameters, you have to write `CHANGE /old//...`

```
CHANGE /old
```

is a further abbreviation of `CHANGE /old/`.

Restriction to columns:

Normally, the character string may be at any place within a record. You can restrict the possible position of the character string to certain columns with the command `ZONE` (compare the entry for this command). A restriction of the zone for the search with the `CHANGE` command is possible as well by specifying the columns. For example,

```
CHANGE 10 20 /old/new/
```

only changes the character string '`old`' if it occurs between the columns 10 and 20, both inclusive. In this specification the borders of the `ZONE` area are ignored. If you specify only one column, the `CHANGE` command will operate from this column on to the last column of the record.

Expansion on several lines (records):

Without specifications changes are only done within the current line. With the specification of the number of lines `n`, for example

```
CHANGE ... /old/new/ n
```

you ask *exaEdit* to search for the old character string within the current line and the `n - 1` following lines and to do the changes there if necessary.

If the character string '`old`' is not found within the specified lines, the current line keeps its position and you receive the message

```
Character string not found: old
```

In opposition to this, if the character string is found `m` times, the current line moves by `n` lines downward (consequently, the data in the visible part of the window move by `n` lines upward) and you receive the message

```
m times changed
```

By default, the search string is only searched for once per record. If the record contains the string more than once, only the one furthest to the left will be changed. To change all occurrences of the string you have to use the parameter

```
A
```

When using the parameter

```
D
```

(= '**d**isplay'), all lines that have been changed are displayed in the dialogue zone.

You can edit in hexadecimal form by giving the parameter

```
H
```

To this end, the given (one or two) character strings must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, you would like to delete all Carriage Return characters (hexadecimal `0d`) in a file, you select the top line of the workfile and enter

```
change /0d// 999 h
```

(Here the precondition would be that the file has less than 1000 lines and `0d` appears only once per record.)

When the parameter

```
I
```

(= 'case–**i**nsensitive') is given, the search for the character string that is to be changed will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
change /ab/12/ i
```

will change either `ab` or `Ab` or `aB` or `AB`.

If you have to use the parameter `I` regularly, the use of the command `CASE` is recommended. `CASE` can force other commands to act as if the parameter `I` was given. For further information see `CASE`.

You are not allowed to specify the commands `I` and `H` together. If you do it still, the parameter `I` will be ignored with the message

```
'I' will be ignored as H is specified
```

The command will then be executed as if only `H` had been given.

`CMDSEP [?|x]` `CMD`

Usually, you can enter more than one *exaEdit* command at once by separating them with a semicolon (';'). With the command `CMDSEP` you may change the separator (`CMDSEP x`) or ask for the current setting (`CMDSEP ?`).

If you only enter 'CMDSEP' (without any parameter), the default setting is restored. It is equivalent to the command

```
CMD ;
```

As separator you may use any character except the question mark ('?') and letters.

Since you will need the command separator frequently when using *exaEdit*, the separator should be on a key that is easy to reach and you should be able to write the separator without the shift key (key ⇧). So, you are recommended not to use the semicolon if it can be only reached with the shift key. In this case you should use another character as command separator, for example the comma:

```
CMD ,
```

You may save the change of the separator to your *exaEdit* profile file.

`CMOVE [(l1[l2])] c1[c2] COLUMN [+|-]col [LINE [+|-]num] [n]` `CM`

`CMOVE` means column move, i.e. it moves columns of a line. It should not be confused with the command `MOVE`, which moves lines.

```
cmove 5 column 20
```

moves column 5 to column 20 of the current line,

```
cmove 5 10 col 20
```

moves the columns 5 to 10 to the columns 20 to 25. Note that you specify only the first of the target columns, since the number of columns has been set when specifying the source columns.

Moving works as follows: The columns starting with the target column are moved to the right by the number of columns to be moved. The characters to be moved are written into the gap just produced and then deleted at their original place. The gap produced by this is closed again by moving to the left the characters at the right side of the gap. In the example `cmove 5 10 col 20` the characters to be moved can afterwards be found in columns 14 to 19, because they have been moved "between" the original columns 19 and 20. This is the same method that is used with the command `MOVE`, which also does not overwrite but inserts the moved data and leaves no gap at the move source.

Source and target area may not overlap. If they do, you receive the message

```
Source and target area overlap
```

Instead of using absolute column specifications you can use relative ones if you add a leading sign to the `column` specification:

```
cmove 5 10 c +15
```

has the same effect as the command above.

Until now, the commands were limited to the current line. As usual, you can have the command applied to n following lines by specifying a number as last parameter:

```
cmove 5 10 colu 20 7
```

applies the command to 7 lines. If the workfile does not have sufficient lines, you get the message

```
End of data
```

The methods described until now are executed from the current line on. Alternatively, you can also specify the lines you want to be affected enclosed in parentheses as first parameter:

```
cmove (500 1200) 5 10 c20
```

This means, `CMOVE` will be applied to the lines 500 to 1200, wherever the current line is. The line numbers specified may also be symbolic line numbers (`t`, `f`, `p`, `*`, `n`, `l`, `b`, `s`). If you specify only one line number, it is assumed the second line number is equal to the first. Please mind the parentheses.

About the last parameter of `CMOVE`: Until now, the target columns have been in the same line as the source columns were, but you can also specify the target columns in other lines:

```
cm5 10c20 line 700
```

This moves the columns 5 to 10 of the current line to the columns 20 to 25 of line 700.

As with the `COLUMN` specification, you can precede the `LINE` specification with a leading sign, thus making the line specification a relative one:

```
cm5 10c20 l-6
```

If the line given cannot be found, you get the message

```
Target record not found
```

Of course, you can also add the (two alternative) numbers of affected lines:

```
cmove (500 b) 8 column 9 line 000400
```

Please keep in mind that you cannot specify a target line that does not exist (yet). So, if you want to move something behind the last line of the workfile, you have to create that (empty) line first.

`CMOVE` allows for whole rectangles of the workfile to be moved to another part of the workfile.

`CODEPAGE [?|DOS|WIN]` `COD`

The representation of special characters can differ within the 32–bit versions of Windows systems.

With `WIN`, *exaEdit* uses the representation that is commonly used by 'real' Windows programs (such as `notepad`). With `DOS`, *exaEdit* uses the representation that is usually used by `DOS` programs (such as `type`).

The default of this switch is `WIN`. The parameter `?` can be used to display the state of the switch. The resulting messages are

`DOS` resp. `WIN`

If you use `CODEPAGE` in Unix, *exaEdit* you will get the message

```
CODEPAGE is only for Windows systems
```

`COMPRESS [?|#n|n|ALL]` `COM`

Reverse command: `EXPAND`

This command compresses records by replacing appropriate sequences of spaces with tab signs. A tab sign means: Put spaces from here to the next tab stop between the last character to the next character. The tab stops are the columns 1, 9, 17, 25, and so on. If a line contains a row of spaces with a tab stop in between them, the first space is replaced with a tab sign (`x09`) and the rest of the line is moved to the left from the tab stop to the tab sign. This procedure is repeated as often as possible.

Without specification, only the current line is compressed. You may specify the number of lines `n` (from the current line on) or all records of the workfile with the parameter `ALL`.

Since this kind of compressing is not very sensible if only short sequences of spaces are compressed, *exaEdit* only compresses rows of at least four spaces. You can change this number with the command

```
compress #n
```

To display the current setting, you use the command

```
compress ?
```

exaEdit will answer with

```
compress #...
```

This can be used to change the value.

If there have been compressions after the `COMPRESS` command, *exaEdit* provides a message of success:

```
Compressed n times in m records by k blanks
```

If `m = 1`, '`records`' is replaced by '`record`' in the display.

If there is nothing to compress, the current line remains the same. Normally, *exaEdit* moves the current line to the record that is compressed last. If you set `n` to a number that would need *exaEdit* to compress records beyond the end of the file (message `End of data`), the current line is moved to the last record of the workfile. After using parameter `ALL`, the current line remains the same, no matter if there has been any compression or not.

`CONCAT [/string/|n]` `CON`

Reverse command: `SSPLIT`

With this command you concatenate the record of the current line with the following record that disappears as an independent record, as a consequence.

If you do not specify a parameter, the following record is placed immediately behind the last character (no space) in the current line.

If you specify a character string, it is placed between the two records to concatenate. A frequent application of this is to put a space between the two parts.

Instead of a character string you may also specify a column. The column designs the place in the record of the current line where the content of the following line should be placed. If the column is behind the last character (no space), additional blanks are inserted. If the column specified comes before the end of the current record, the characters of the second record overwrite the characters of the current line. If the second record is short enough, the rest of the characters of the current line remains. The behaviour of `CONCAT` is similar to the behaviour of the command `REPLACE`. The only difference is that the characters to be inserted do not have to be specified but are taken from the second record.

The following example illustrates this; the first line is the current line.

```
Do give my kindest regards to
Peter
```

command	result
`concat`	`Do give my kindest regards toPeter`
`concat / /`	`Do give my kindest regards to Peter`
`concat 35`	`Do give my kindest regards to     Peter`
`concat 5`	`Do gPetery kindest regards to`

If the line after the current line is empty when you call `CONCAT`, the command

```
CONCAT /string/
```

works like the expansion of the current line by the characters ‘`string`’. (With the command `SSPLIT` (compare the entry there), you can create an empty line if you wish to use the effect described above to expand a record.)

`COPY [num1[num2] [wfname]]` `CO`

The command `COPY` copies one or more records behind the record of the current line in the workfile. The records to be copied are specified by their number or a symbolic number.

The number of a record may be specified without the leading zeros. Symbolic record numbers resp. line numbers are:

`*`	for the current line.
`p`	(= previous) for the line before the current line.
`n`	(= next) for the line behind the current line.
`f`	(= first) for the first line of the workfile.
`l`	(= last) for the last line of the workfile.
`t`	(= top) for the top line of the workfile.
`b`	(= bottom) for the last line of the workfile.
`s`	(= set) for the line marked with `SET`.

If you specify only one number in the `COPY` command, only this record will be copied. If you specify two numbers, all records from the first to the second number will be copied. For this purpose, it is necessary that `num1` $\leq$ `num2` is valid. If that is not the case, you receive the message

```
First number larger than second
```

If you specify a number that does not exist in your workfile, you receive the message

```
Number ... not found
```

The ... are replaced by the number in the output.

You also have to mind that the target area of the `COPY` command is not within the area to be copied. To be precise, the target area of the copy command is the place between the current and the following record. The area to be copied must not contain the current line and the line following the current line. This would be the case if the command was `COPY * num2` and `num2` was behind the current line. If you do not comply to these restrictions, you will receive the message

```
Target in COPY area
```

Some examples:

`co *`	doubles the current line.
`co b`	copies the last line.
`co f l`	copies the whole workfile beneath the current line (only possible if the current line is the last line or the top line).
`co500*`	copies from line 500 to the current line.
`co p *`	copies from the line before the current line to the current line, i.e. doubles two records at the place of the current line; that is, changes the 2 lines `a b` to the 4 lines `a b a b`, if `b` was the current line.

Without the specification of a third parameter, the records are taken from the workfile where they should be inserted. It is also possible to specify a workfile as third parameter where the records should be fetched for copying. If the specified workfile does not exist, you receive the message

```
Workfile not found
```

For example, the command

```
copy 1000 1500 abc
```

copies the records with the numbers 1000 to 1500 from the workfile abc behind the current line of the active workfile.

Another example: You want the complete content of the main workfile (main) in a new workfile named abc. The easiest solution is to use

```
wf abc;co f l main
```

Please, note that COPY T B copies the top line as well, which will not be desirable in most cases.

When specifying the workfile name, you have to specify the second parameter (the second number) (even if it is identical with the first one) if the workfile name might be mixed up with a symbolic line number. For example:

```
co 500 a
```

copies the record with the number 500 from the workfile a, while

```
co 500b
```

refers to all the records in the active workfile from number 500 onward, so that you have to write

```
co500 500b
```

if you want to copy the record 500 from the workfile b.

The copied records receive numbers that match with the existing numbers of the two records, in which the copied ones have been inserted. Since the method for the definition of the new numbers is the same for other commands, it is described only once in section 3.1.31, *Inserting Record Numbers*.

COUNT num1 [num2] COU

The command COUNT counts the number of records in the workfile. The parameters mark the borders of the area, in which the records should be counted. The two records on the border are counted as well.

The record number(s) may be entered without leading zeros. Record numbers may be symbolic ones as well:

- * for the current line.
- p (= previous) for the line before the current line.
- n (= next) for the line after the current line.
- f (= first) for the first line of the workfile.
- l (= last) for the last line of the workfile.
- t (= top) for the top line of the workfile.
- b (= bottom) for the last line of the workfile.
- s (= set) for the line marked with SET.

If you call COUNT with only one parameter, *exaEdit* completes the command and assumes the record of the current line as second parameter.

As a result, *exaEdit* writes the number found in the dialogue zone. If the number of the first record is greater than the number of the second one, the result is marked as being negative with a minus sign in front of it.

The result of the count is written into the parameter variable &Count.

`DELETE [n|ALL]` DE

Related commands: `DELETEL` and the prefix command `DELETE`.

The command `DELETE` deletes the record of the current line and the `n - 1` following records if `n` is specified.

The specification of `ALL` deletes all records of the workfile.

After the deletion, the last record previous to the deleted records is positioned into the current line.

`DELETEL|DL num1 [num2]` DL

Related commands: `DELETE` and prefix command `DELETE`.

The command `DELETEL` (its abbreviation is only `DL`) deletes only one line if you specify only one parameter. If you specify both parameters, the records from the number `num1` to the number `num2` (including) are deleted.

The numbers of the record(s) may be specified without leading zeros.

Symbolic record numbers resp. line numbers are allowed as well. That may be:

`*`	for the current line.
`p`	(= previous) for the line before the current line.
`n`	(= next) for the line after the current line.
`f`	(= first) for the first line of the workfile.
`l`	(= last) for the last line of the workfile.
`t`	(= top) for the top line of the workfile.
`b`	(= bottom) for the last line of the workfile.
`s`	(= set) for the line marked with `SET`.

Example:

```
dl f l
```

Deletes all records in the workfile and therefore is identical with `DELETE ALL`.

After the erasure, the last record before the deleted records is set into the current line.

`DISPLAY [n|ALL]` D

The command `DISPLAY` shows the record of the current line in the dialogue zone.

If you specify a number `n` of records or the parameter `ALL`, `n` records (starting with the current line) or all records of the workfile will be listed.

The main application of this command is in the line mode of *exaEdit*. In the line mode, the records of the workfile are not displayed in the window if you do not ask for it with the command `DISPLAY`.

Besides, you can also use `DISPLAY` in the (normal) window mode to create a copy of the record of the current line in the dialogue zone, for example. Then you could use the line for the number command (change the line and create a new line out of the old one).

Note, please, that `DISPLAY` only shows the first line of a record that takes more than one line in the data zone.

`DL|DELETEL num1 [num2]` DL

Related commands: DELETE and prefix command DELETE.

The command DELETEL (its abbreviation is only DL) deletes only one line if you specify only one parameter. If you specify both parameters, the records from the number num1 to the number num2 (including) are deleted.

The numbers of the record(s) may be specified without leading zeros.

Symbolic record numbers resp. line numbers are allowed as well. That may be:

- `*` for the current line.
- `p` (= previous) for the line before the current line.
- `n` (= next) for the line after the current line.
- `f` (= first) for the first line of the workfile.
- `l` (= last) for the last line of the workfile.
- `t` (= top) for the top line of the workfile.
- `b` (= bottom) for the last line of the workfile.
- `s` (= set) for the line marked with SET.

Example:

```
dl f l
```

Deletes all records in the workfile and therefore is identical with DELETE ALL.

After the erasure, the last record before the deleted records is set into the current line.

`DOWN|+|NEXT [n]` DO|+|N

The pointer to the current line is moved downward to the end of the workfile by n records. Since the current line has a fixed position in the window, the text moves upward.

If you do not specify n, the value 1 is assumed.

If n is greater than the number of records after the current line, *exaEdit* writes

```
End of data
```

as a message in the dialogue zone and the current line remains unchanged.

`END|QUIT` E|Q

The command END finishes the editor. In advance to the finishing, it is checked whether you made changes in the existing workfiles. If this is the case, you are informed about it and you have a chance to continue the *exaEdit* session to save the changed workfiles onto a data medium.

If there is only the workfile MAIN in the exaEdit session, you receive the message

```
Changes not saved
Press J or Y to stop:
```

In line mode, the second line is:

```
Enter J or Y to stop:
```

If there are other workfiles as well, the message reads instead:

```
Workfiles not saved: ...
```

Instead of the ... the workfiles concerned will be listed.

If you decide not to save the changes, you press the key 'j' or 'y' (small letters or caps) and the editor is stopped immediately. Contrasting, if you press any other key, the processing of the command END stops and you can go on editing as usual.

Please, note that it is sufficient to press a key to answer the question whether you want to leave the editor. It is not necessary to press the return key afterwards.

EXEC EX

The command EXEC requires that a workfile called EXEC exists. This workfile must contain *exaEdit* commands only.

If the command EXEC is called, the *exaEdit* commands from the workfile EXEC are executed in the workfile where the command EXEC was called.

If the workfile EXEC does not exist, you receive the message

```
Workfile not found
```

After the command EXEC nothing else may appear in a command line.

The command lines in the file EXEC may not exceed the window width of the workfile where the command EXEC is given. If you do not comply to this restriction, you receive the message

```
n. EXEC line longer than window width ...
```

For n, *exaEdit* will substitute the line number, and the dots will be replaced with the current window width.

For example, you can use EXEC to save frequently needed allocations of F keys as *exaEdit* commands (compare PFK) together in a file. When there is need for these allocations you can load them in a workfile called EXEC and bring them into effect in all workfiles of your *exaEdit* session. This procedure is an alternative to the application of the *exaEdit* profile.

With EXEC all commands are executed as usual, with one exception: The command FILE does not have the usual safety queries like

```
Old data set, press J or Y to replace it:
```

etc. Thus, special care should be taken when using FILE within EXEC.

EXPAND [n|ALL] EXP

Reverse command: COMPRESS

This command expands tab signs spread among the data into spaces. If you do not specify a parameter, the record of the current line is changed. The specification n denotes that n records, beginning with the current line, should be worked on. If you specify ALL, all records of the workfile will be changed if necessary.

The present setting of *exaEdit* assumes tabulators at columns 9, 17, 25, ...; it is not possible to change the default setting.

The expansion procedure works as follows: If a tab sign is found (hexadecimal value x09), it is replaced by a blank. The text after the tabulator sign is moved to the next tab stop to the right; the gap that might result is filled with space signs. This procedure is used for all tabs of the record.

If some expansions have been made because of the EXPAND command, *exaEdit* provides a message of success:

```
Expanded n times in m records by k blanks
```

If m = 1 is valid, 'records' is replaced by 'record'.

If there is nothing to expand, the current line remains the same. Normally, *exaEdit* moves the current line to the record that is expanded last. If you set n to a number that would need *exaEdit* to expand records beyond the end of the file (message `End of data`), the current line is moved to the last record of the workfile. After using parameter `ALL`, the current line remains the same, no matter if there has been any expansion or not.

`FILE [ filename]` FIL

This command writes the content of the workfile in a file. `FILE` is explained in detail in section 3.1.4, *Saving a File*.

`FILL [ string]` FILL

This command is filling up records by moving so many words from the following record into the the current record as may have place in the visible line (value of `LWWIDTH`). The same action takes place for the next record and so on.

Filling up always starts with the current line.

Filling stops either before the next blank line or before that record which begins with `string`, or at the end of the file. If you do not specify a parameter then the stop record will be the next blank line.

If you have a text where paragraphs are marked by blank lines or by an unambigious mark at the end of the paragraph (but at the beginning of a record), you may by this means fill the text paragraph by paragraph. In all cases you should make sure that the blank lines or the stop strings are present or are typed in the right way. Otherwise the filling up may unwantedly run too far or even to the end of the file.

`HELP [cmd|PREFIX]` H

If this command is entered without parameters, the result is a list of all commands, special help texts and prefix commands displayed in the window. The command words and special help texts in this list are spelled with capital and small letters. The capital letters at the beginning of a command word denote the minimal abbreviation. For example,

```
COpy
```

denotes that the `COPY` command can be entered as `COPY`, `COP` or `CO` but not as `C`. The list of the commands is sorted according to the minimal abbreviations, i.e. not exactly alphabetically with regard to the full command words.

If you specify a command name as parameter of the `HELP` command, you receive the syntax and meaning of that command. In the first line of the output, there is the complete command syntax on the left. In the spelling of the syntax, vertical lines and square brackets have the meaning explained in section 3.2.1, *Notation*. In the first line on the right, there is a short characterization of the command in English. The rest of the lines of the output describe the commands as good as possible within a maximum of five lines. This restriction has been chosen to make it possible for you to watch the current section of the workfile and the help text at the same time. Of course, this works only as long as you do not want to enter a command.

If you specify the parameter `PREFIX` in addition to the command name, you will get the help text for the according prefix command. This parameter is necessary for those commands that exist both as a command line command and as a prefix command, e.g. `D` or `I`.

The longer version of the help text is only available in the manual at hand, which is also relevant in cases of doubt.

With the command `HELP` you can have some special help texts displayed in the window. Formally, this works in the same way as the display of help texts of commands but the parameter words are no commands.

- `function` shows all *exaEdit* functions, compare section 3.1.30.
- `profilex` shows information about the profile files.
- `symbolic` lists all symbolic line numbers allowed (for the commands `COPY`, `DL`, `MOVE`, etc.).

`HEXA` HEX

The first time this command is used during a *exaEdit* session, the display of the current line will change from ASCII to hexadecimal.

The next time `HEXA` is used, the display of the current line will change back to ASCII (i.e. ‘normal’) etc.

The command `HEXA` only changes how the data is displayed. For hexadecimal editing the parameter `H` of the commands `ALIGN`, `CHANGE`, `SSPLIT`, and those of the `LOCATE` family is needed.

`INDENT [? | n | AUTO | ON | OFF]` IND

This command controls the behaviour of *exaEdit* during automatic indenting during input mode.

Indenting will be switched on or off with `ON` or `OFF`. The parameter `n` specifies indenting by n columns. This means that when typing a new line the cursor will be in column n + 1.

If instead the parameter `AUTO` holds, then each new line will be indented the same amount as the previous line. The indenting amount memorised from the previous line by *exaEdit* will not be disturbed by the input of a blank line.

With `INDENT ?` the valid parameter values will be shown.

The default is

```
INDENT AUTO ON
```

`INLENGTH [? | n | AUTO | SOFT | HARD]` INL

This commands controls the behaviour of *exaEdit* during the automatic line break during input mode. Line break means that during input in the input a record), you may in this way fill the text paragraph by paragraph. In the fact how many characters fit into a line (a record). You may find further details in the section 3.1.18.2, *Automatic line break*.

Default is breaking of lines during input mode, if they would get longer than `LWWIDTH` columns. This is the same as the specification of the parameter `AUTO`. Yoe may, however, choose e fixed column by specifying the parameter `n`.

Default is also that the line break occurs at the last blank character before the given or assumed column. You may, however, request by specifying the parameter `HARD`, that the line break should occur exactly at the column, that is, independent of word boundaries.

With `INLENGTH ?` the valid parameter values will be shown.

The default is

```
INLENGTH AUTO SOFT
```

`INPUT [/string/]` I

This command has two meanings, depending on whether you give parameters or not.

When you use `INPUT` without parameters, the command switches from the command mode to the input mode. If you want to switch back from the input mode to the command mode, you have to make an empty input, i.e. press the `return` key without having pressed another key except the `return` key.

When you switch on the input mode,

```
Input
```

is displayed in the window (as an invitation to give some input). This display is overridden with the next input you give. The permanent display for the input mode is an

```
I
```

in the status line. Besides, the ruler slides to the left to make it possible to count the columns correctly if necessary.

You use the input mode to transfer new records into the workfile. The new records are entered in the dialogue zone (i.e. where you usually enter the *exaEdit* commands). These records are inserted after the record in the current line.

Every line entered is repeated in the first line of the dialogue zone, the second line of this zone is erased and the cursor takes the first position of the second line for the next input. But you can make your input in any line in the window, as usual in *exaEdit*. You may apply this possibility especially to the previous line of input that has remained in the first line of the dialogue zone.

As long as *exaEdit* is in the input mode, the new lines in the data zone do not yet receive numbers. The line numbers are only added when the input mode is quit since only then the numbers of new lines is determined. Section 3.1.31, *Inserting Record Numbers*, describes how the numbers of new records are determined.

Now for the parameter use of `INPUT`: This inserts the string you have specified as a new line after the current line.

`INSMODE [? | ON | OFF]` `INS`

This command controls the behaviour of *exaEdit* during insert mode, see also the section 3.1.10, *Deleting and Inserting of Characters*. It allows switching between the ordinary insert mode (`INSMODE OFF`) and the permanent insert mode (`INSMODE ON`). The just mentioned commands may also be replaced by a single or double pressing of the `Ins` key (which is sometimes not part of the keyboard).

With `INSMODE ?` you will be shown the state of the switch.

Default is

```
INSMODE OFF
```

`KEYBOARD [?|EXAEDIT|ALL|TEST]` `KEYB`

This command has two different functions. The first one is useful to change the reaction to the pressing of certain keys on the keyboard. Besides the keys with normal characters, there are others that are linked to certain functions. Depending on the hard- or software environment of *exaEdit*, these special keys may be allocated with different pieces of information. For this reason, there are keys that have no meaning for *exaEdit*. This is not only the present state of affairs, this situation may continue in the future. The keys with no meaning for *exaEdit* are simply ignored if they are hit. This is valid in the default setting

```
keyboard exaedit
```

But if

```
keyboard all
```

is valid, you cause a message when you hit one of these keys. The message explains the information *exaEdit* received from this key and it says that this information is ignored. However, it may happen that the message for the key disturbs your normal usage of *exaEdit*, for example, if you press one of these keys during your input. This is why the silent ignoring of these keys is part of the default setting.

With the question mark as parameter you can ask for the current setting of `KEYBOARD`. The response is either `EXAEDIT` or `ALL`.

The second function of the command `KEYBOARD` is the usage with the parameter

```
TEST
```

If you apply this parameter, *exaEdit* writes

```
Press key:
```

in the window and expects that you press exactly one key. If *exaEdit* answers this, the calling of `KEYBOARD TEST` is finished. If there is no reaction of *exaEdit*, you have to press the `return` key to receive the appropriate reaction of *exaEdit*.

In the section 3.1.29, *Keyboard Test*, you find a detailed explanation on the interpretation of the messages of *exaEdit*.

`LANGUAGE [?|DEUTSCH|UDEUTSCH|ENGLISH]` LA

With this command you adjust the language, in which *exaEdit* provides messages and help texts.

The default setting of *exaEdit* for the parameter is `UDEUTSCH` or `ENGLISH`, depending on the installed version. `UDEUTSCH` means German language and usage of German umlauts and sharp s. If you do not want these special characters, for example because they look strange or they are not displayed at all, you may switch to the circumscriptions `ae`, ..., `ss` by using

```
language deutsch
```

Please, note that there may be another default setting in a profile file.

With the parameter `?` *exaEdit* displays the current language setting.

`LOAD filename` LOA

Related command: calling *exaEdit*

`LOAD` serves to load a file from the disk into the current workfile. The command `LOAD` is described in detail in the section 3.1.3, *Loading a File*.

`LOCATE [col1[col2]] [/string/[H][I]]` L

Related commands: `RLOCATE`, `NLOCATE`, `RNLOCATE/NRLOCATE`

`LOCATE` is used to search for character strings. As parameter you specify a character string. The latter is embedded in two delimiters. In the general syntax above, a slash (‘/’) is used as delimiter but any other character is possible as well. In practice, the special character on the lower right on the keyboard has proved to be helpful. When you get used to this key as general delimiter, you will only have to use another character if the special character occurs in the character string to search for. You may leave out the delimiter at the end of the character string if the rest of the command line is empty.

```
locate /abc/
```

searches for the character string ‘`abc`’,

```
l -a/b-
```

searches for the character string ‘`a/b`’,

```
l abc
```

searches for the character string ‘`bc`’ because ‘`a`’ is the first delimiter and the second delimiter is missing. `l abca` would search for the character string ‘`bc`’ as well.

Please, note that you have to use the final delimiter if you concatenate the `LOCATE` command with other commands:

```
l /abc/;-2
```

searches for the string 'abc' and then executes the command -2 while

```
l /abc ;-2
```

would search for the string 'abc ;-2'.

The search begins at the record after the one in the current line.

If the character string is found, the record containing the string appears in the current line. The section displayed in the workfile is moved according to the change.

Contrasting, if the character string is not found, the following message appears:

```
Character string not found: ...
```

In the output in the window the ... are replaced by the string searched for. In this case, the current line remains unchanged.

If *exaEdit* could not find the character string in the last record of the workfile, normally the search continues at the first record either until the character string is found or until the initial record of the search is reached. 'Normally' means that the switch manipulated by the command WRAP is ON.

To show you that the search continued at the beginning of the workfile, the message

```
Search from begin (wrap)
```

is generated. This message is followed by either the positioning of the workfile section when the character string is found or the negative message from above when the string is not detected.

Anyway, if WRAP OFF is valid, the search ends at the last record of the workfile. If the search was not successful, the messages

```
End of data
Character string not found: ...
```

are generated.

Frequently, it is necessary to search for the same character string more than once. In this case, it is sufficient to enter LOCATE without any parameter: The last character string that has been searched for is used automatically.

The commands LOCATE, RLOCATE, NLOCATE and RNLOCATE/NRLOCATE use the same character string. The character string to be searched for is the same for all workfiles.

By default, the complete record will be searched. By giving two column numbers as first parameters the search is restricted to that area. A character string will then only be found if the given area contains it fully. If only one column number is given, the search area goes from that column to the end of the record. The command ZONE can also be used for column restrictions (see description). When both ZONE and the column number parameters are in effect, the latter one precedes.

If LOCATE is used a second time, and then without any parameters, possible column restrictions are still in effect. But if a new search string is given, the restriction are no longer in effect. A LOCATE with a (new) column restriction after another LOCATE will use the same search string as the first LOCATE.

You can search for data in hexadecimal form by giving the parameter

```
H
```

To this end, the given character string must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, the next record containing a tab sign (hexadecimal 09) should be found, the command

```
locate /09/ h
```

could be used.

When the parameter

```
I
```

(= 'case–**i**nsensitive') is given, the search for the character string will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
locate /ab/ i
```

will find either ab or Ab or aB or AB.

If you have to use the parameter I regularly, the use of the command CASE is recommended. CASE can force other commands to act as if the parameter I was given. For further information see CASE.

LWWIDTH [n|?] LW

Related command: SZONE

LWWIDTH (= logical window width) determines the logical window width, i.e. the maximal line width that would be displayed if the physical window was wide enough. All the records that are longer than LWWIDTH are presented in subsequent lines. A record with 200 bytes would take four lines if LWWIDTH had the value 60. The last line would then contain the last 20 bytes of the record.

If you want to avoid subsequent lines in your window, you can make the rests of the records disappear on the right by applying LWWIDTH properly.

The default setting of LWWIDTH is chosen in such a way that all the data of the workfile are visible in the window or become visible when you leaf through the file. As a consequence, subsequent lines are used if necessary.

The default value of LWWIDTH depends on the physical window width and the value of SKEY (see the description of the command SKEY):

lwwidth = physical window width − skey − 1

If you change the value of SKEY, LWWIDTH adjusts to it automatically, so that the entire window width is used: A window with a physical width of 80 and SKEY = 6 has for LWWIDTH the value 73; with SKEY = 0 the value of LWWIDTH is 79.

If you enter the command LWWIDTH without parameters, the default value of LWWIDTH regarding the current value of SKEY is adopted. For example, a physical window width of 80 and SKEY = 0 combined with the command LWWIDTH alone result in a value of 79.

The parameter '?' displays the current value of LWWIDTH.

MANUAL [name|∗] [?|DEFAULT|DELETE|SET /string/] MAN

This *exaEdit* command serves to create, change, delete, list, and execute Unix or DOS commands to look at the *exaEdit* user manual.

With the command

```
manual * ?
```

you receive a listing of all defined manual parameter sets. If it was not changed or deleted in the installation profile or by yourself, there is at least the parameter set that is included in *exaEdit*. It is similar to one of the following:

```
MAN   0!   start iexplore exaedit.de/dok/en/
MAN   0!   konqueror exaedit.de/dok/en/&
MAN   0!   netscape exaedit.de/dok/en/
```

Lines of the first type appear on Windows systems, lines of the second type on Linux systems, lines of the third type on other Unix systems. The different parts have the following meaning:

`MAN` is the abbreviation of the *exaEdit* command word that is output to allow you to change the line and enter it as a new command.

`0` is the name of the `manual` parameter set. Such a name may consist of one to four letters or digits.

`!` marks the `manual` parameter set as default setting, i.e. this set is taken if no `manual` name is specified when the `manual` command for execution or listing is called.

`start iexplore` resp. `netscape` etc. is the name of the DOS resp. Unix command word that should be executed to display the *exaEdit* user manual online.

`exaedit.de/dok/en/` is the file that the browser commands need for this purpose.

`&` at the end of the Unix command makes sure that `netscape` works asynchronously to *exaEdit*.

With the command

```
manual ?
```

only the `manual` parameter set marked with `!` is displayed.

With the command

```
manual name ?
```

only the specified `manual` parameter set is displayed.

With the command

```
manual
```

the `manual` parameter set marked with `!` is used and the Unix or DOS command in it is executed.

With the command

```
manual name
```

the specified `manual` parameter set is used and the Unix or DOS command implied in it is executed.

With the command

```
manual name default
```

the specified `manual` parameter set is marked with `!` and the set that has been labelled with an `!` so far loses its label.

With the command

```
manual name delete
```

the specified `manual` parameter set is deleted. An `*` instead of `name` deletes all `manual` parameter sets.

With the command

```
manual name set /character string/
```

a new `manual` parameter set is defined. As mentioned above, `name` has to consist of one to four letters or digits. The `character string` has to be a complete Unix or DOS command. Since it is highly probable that it contains slashes, you have to choose another character as a delimiter for the `character string`. You may abbreviate the parameters `DELETE` and `SET` as usual. If you wish to define a parameter set which is very similar to an existing one, you can have the existing set displayed with `manual name ?` and then alter the display as you like it and seal it by pressing the `return` key.

Please, keep in mind that you may organize the changes or supplements of the existing `manual` parameter sets into your private *exaEdit* profile file as well.

`MOVE num1[num2]` M

The command `MOVE` moves one or more records behind the record of the current line of the workfile. The records to be moved are referred to by a symbolic number or the line number.

The record number may be specified without leading zeros. Symbolic record numbers resp. line numbers are:

`*`	for the current line.
`p`	(= previous) for the line before the current line.
`n`	(= next) for the line after the current line.
`f`	(= first) for the first line of the workfile.
`l`	(= last) for the last line of the workfile.
`t`	(= top) for the top line of the workfile.
`b`	(= bottom) for the last line of the workfile.
`s`	(= set) for the line marked with `SET`.

If you specify only one number, only this record is moved; if you specify two numbers, the records from the first to the second number (including the borders) are moved. For this purpose `num1` $\leq$ `num2` has to be valid. If that is not the case, you receive the message

```
First number larger than second
```

If you specify a number that does not exist in your workfile, you receive the message

```
Number ... not found
```

The ... are replaced by the number in the output.

Besides, you have to take care that the area to be moved does not contain the target; the target is the place between the current record and the following record. As a consequence, the area to be moved must not contain the current line and the line following it at the same time. This would be the case in the example `MOVE * num2` with `num2` behind the current line. If you do not comply to these restrictions, you will receive the message

```
Target in MOVE area
```

Some examples:

`m400 700`	moves the records with the numbers `400` to `700` behind the record of the current line.
`m b`	moves the last record.
`m p`	moves the record before the record of the current line behind the record of the current line, i.e. interchanges the two records.

Please, note that the top line cannot be moved. The command `MOVE T 500` will result in the message

```
You cannot move the top line
```

The records moved receive numbers that go well with the already existing numbers before and after the inserted records. Since the procedure to determine the new numbers is the same for several commands, it is explained only once in the section 3.1.31, *Inserting Record Numbers*.

`NEXT|+|DOWN [n]` `N|+|DO`

The pointer to the current line is moved downward by `n` records, in the direction of the end of the workfile. Since the current line takes a fixed position in the window, the text slides upward.

If you do not specify `n`, the value 1 is assumed.

If `n` is greater than the number of records after the current line, *exaEdit* writes

```
End of data
```

in the dialogue zone and the current line remains unchanged.

`NLOCATE [col1[col2]] [/string/[H][I]]` `NL`

Related commands: `RNLOCATE/NRLOCATE`, also `LOCATE` and `RLOCATE`

`NLOCATE` is used to search for the nearest line not containing the specified character string. As parameter you specify a character string. The character string normally has to be surrounded by two delimiters. Above, the slash ('/') is used as delimiter but any other character is permissible. In practice, the special character on the lower right of the keyboard has proved to be effective. You only have to use another character if the character string contains the special character. You may omit the delimiter at the end of the character string if the rest of the input line is empty.

```
nlocate /abc/
```

searches for the nearest line down in the text that does not contain the character string '`abc`';

```
nl -a/b-
```

searches for the nearest line down in the text that does not contain the character string '`a/b`';

```
nl abc
```

searches for the nearest line down in the text that does not contain the character string '`bc`' because '`a`' is the delimiter at the beginning of the character string and the final delimiter is missing. Entering the command line `nl abca` would also result in a search for the nearest line not containing the character string '`bc`'.

Please mind the final delimiter when concatenating commands:

```
nl /abc/;-2
```

searches the nearest line down in the text that does not contain the character string '`abc`' and then executes the command `-2` while

```
nl /abc ;-2
```

would launch a search for lines not containing the string '`abc ;-2`'.

The search always starts at the record after the one in the current line.

If a line that does not contain the specified character string is detected, that line is put into the current line; the visible section of the workfile is moved appropriately.

If *exaEdit* could not find a line without the character string, the following message is generated:

```
Character string in all records: ...
```

for the ... the character string specified is in the output; the current line remains unchanged.

If *exaEdit* has found the character string in the last record of the workfile, the search continues at the first record, normally. The search goes on until either the character string is not found or the record from which the search started is reached again, without success. Above, 'normally' includes that the switch manipulated by `WRAP` (compare explanation at the entry of `WRAP`) is turned `ON`.

To inform you that the search has been continued at the beginning of the workfile, the message

```
Search from begin (wrap)
```

is generated. This message is followed either by the positioning of the current line when the string is not found in a line or by the message of failure mentioned above.

Contrasting, if `WRAP OFF` is valid, the search ends at the last workfile record. If it was not successful, i.e. the character string is in every line searched, *exaEdit* generates the messages

```
End of data
Character string in all records: ...
```

Frequently it will be necessary to launch searches for the same character string more than once. Then it is sufficient to enter `NLOCATE` without parameters; the last character string that has been searched for is used automatically.

The commands `NLOCATE`, `RNLOCATE`/`NRLOCATE`, `LOCATE` and `RLOCATE` use the same character string. The character string to be searched for is the same for any workfile.

By default, the complete record will be searched. By giving two column numbers as first parameters the search is restricted to that area. A record will then be found as long as the given area does not contain the search string completely. If only one column number is given, the search area goes from that column to the end of the record. The command `ZONE` can also be used for column restrictions (see description). When both `ZONE` and the column number parameters are in effect, the latter one precedes.

If `NLOCATE` is used a second time, and then without any parameters, possible column restrictions are still in effect. But if a new search string is given, the restrictions are no longer in effect. A `NLOCATE` with a (new) column restriction after another `NLOCATE` will use the same search string as the first `NLOCATE`.

You can search data in hexadecimal form by giving the parameter

```
H
```

To this end, the given character string must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, the next record not containing a tab sign (hexadecimal `09`) should be found, the command

```
nlocate /09/ h
```

could be used.

When the parameter

```
I
```

(= ‘case–**i**nsensitive’) is given, the search for the character string will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
nlocate /ab/ i
```

can find records that contain neither `ab` nor `Ab` nor `aB` nor `AB`.

If you have to use the parameter `I` regularly, the use of the command `CASE` is recommended. `CASE` can force other commands to act as if the parameter `I` was given. For further information see `CASE`.

`NRLOCATE|RNLOCATE [col1[col2]] [/string/[H][I]]` NRL|RNL

Related commands: NLOCATE, also RLOCATE and LOCATE

NRLOCATE searches backwardly for the nearest line not containing the character string specified (reverse search). As parameter you specify a character string. The character string has to be put in delimiters, normally. Above, the slash ('/') is used as delimiter; any other character is allowed as well. In practice, the special key on the lower right on your keyboard has proved to be useful. You only need a different delimiter if the character you have got used to is part of the character string specified. The delimiter at the end of the character string is omittable if the rest of the command line remains empty.

```
nrlocate /abc/
```

launches a backward search for the nearest line not containing the character string 'abc';

```
nrl -a/b-
```

launches a backward search for the nearest line not containing the character string 'a/b';

```
nrl abc
```

launches a reverse search for the nearest line not containing the character string 'bc' because 'a' is the delimiter at the beginning and the delimiter at the end is missing. `nrl abca` would also launch a reverse search for the nearest line that does not contain the character string 'bc'.

Please mind the final delimiter when concatenating commands:

```
nrl /abc/;-2
```

reversely searches the nearest line that does not contain the string 'abc' and then executes the command -2 while

```
nrl /abc ;-2
```

backwardly searches the nearest line not containing the string 'abc ;-2'.

The search starts at the record before the one in the current line.

If the nearest line that does not contain the specified character string is found, that line is positioned into the current line. The section of the workfile displayed in the window is moved correspondingly.

If *exaEdit* cannot find a line not containing the requested string, the message

```
Character string in all records: ...
```

appears; the ... are replaced by the specified character string; the current line remains unchanged.

If *exaEdit* finds the character string in every record until the first record of the workfile has been reached, the search continues at the last record, normally. The search goes on until either the character string is found or the record from which the search started is reached without success, again. In the sentence above, 'normally' means that the switch manipulated by WRAP (compare the entry for WRAP) is turned ON.

To show that the search has continued at the end of the workfile, *exaEdit* generates the message

```
Search from end (wrap)
```

This message is followed by either the positioning of the workfile section on the display or the message of failure mentioned above.

In contrast, if WRAP OFF is valid, the search ends at the first record of the workfile if the character string could not be identified. If the character string is there in every record that has been checked, these messages appear:

```
Begin of data
Character string in all records: ...
```

Frequently, it will be necessary to search for the same character string several times. Then it is enough to enter the command `NRLOCATE` without parameters; the last character string that has been searched for is used automatically.

The commands `NRLOCATE`, `RNLOCATE`, `NLOCATE`, `LOCATE` and `RLOCATE` use the same character string. The character string is the same for any workfile.

By default, the complete record will be searched. By giving two column numbers as first parameters the search is restricted to that area. A record will then be found if the given character string is not or not completely contained within the area. If only one column number is given, the search area goes from that column to the end of the record. The command `ZONE` can also be used for column restrictions (see description). When both `ZONE` and the column number parameters are in effect, the latter one precedes.

If `NRLOCATE` is used a second time, and then without any parameters, possible column restrictions are still in effect. But if a new search string is given, the restrictions are no longer in effect. A `NRLOCATE` with a (new) column restriction after another `NRLOCATE` will use the same search string as the first `NRLOCATE`.

You can search in hexadecimal form by giving the parameter

```
H
```

To this end, the given character string must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, the nearest previous record not containing a tab sign (hexadecimal `09`) should be found, the command

```
nrlocate /09/ h
```

could be used.

When the parameter

```
I
```

(= 'case–**i**nsensitive') is given, the search for the character string will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
nrlocate /ab/ i
```

will find the nearest previous record containing neither `ab` nor `Ab` nor `aB` nor `AB`.

If you have to use the parameter `I` regularly, the use of the command `CASE` is recommended. `CASE` can force other commands to act as if the parameter `I` was given. For further information see `CASE`.

`PFK [n|ALL] [?|LOCK|UNLOCK|SET /string/]` PF

`PFK` abbreviates program function key, i.e. keys that call program functions (in opposition to functions of the operating system). The keys concerned usually are labelled with `F1`, `F2`, etc. Therefore the keys are referred to as `F` keys. In section 3.1.23, *Programmable Function Keys*, there is detailed explanation on the sense and usage of the allocation of commands or *exaEdit* functions on `F` keys.

`PFK` basically has two parameters. The first one specifies the `F` key concerned and the second parameter specifies the function (to show, lock, unlock, allocate).

The first parameter is either the number of the `F` key or it is `ALL` if every `F` key is referred to. If you leave out the first parameter in combination with the function 'show', `ALL F` keys is assumed; if you omit the first parameter together with one of the other functions, no F key is adopted. In other words, the first parameter is necessary together with the functions LOCK, UNLOCK and SET.

Instead of the single specification n, you may also specify a range n-m or n:m, or a list of numbers and ranges, e. g. 3 4-7 9.

If you leave out the second parameter, *exaEdit* adopts the function to 'show'. As a consequence, the commands

```
pfk all ?      pfk all      pfk ?      pfk
```

are identical, as well as pfk n ? and pfk n, etc.

With the parameter LOCK you lock the specified F key(s), i.e. that you cannot allocate it (or them) by simply entering characters and pressing the F key afterwards. F keys that have been defined with the command SET are locked automatically.

With the parameter UNLOCK you remove the lock of the specified F key(s).

With the parameter SET /string/ you allocate the specified F key(s) with the string. The delimiters of string are arbitrary, as usual, the final delimiter is optional.

```
pfk 5 set qwho
```

allocates 'who' on the F key F5.

Please, note that the apostrophes at the beginning and at the end belong to the *exaEdit* functions. In the following definitions, only F4 results in the *exaEdit* function del while the two at the beginning, F1 and F2, result in the *exaEdit* command delete, and F3 does not make sense.

```
pfk 1 set /del/
pfk 2 set 'del'
pfk 3 set /'del
pfk 4 set /'del'
```

POINT num PO

Related commands: number command, +, -, NEXT, BACK, UP, DOWN, TOP, BOTTOM, RETURN

The command POINT places the record with the number specified into the current line, the workfile section in the window is moved accordingly.

The record number may be specified without leading zeros.

Symbolic record numbers are allowed (compare the explanation at the entries of the commands COPY, DELETEL and MOVE) but the direct positioning commands will be preferable, e. g. TOP instead of POINT T.

If you specify a number that does not exist, you receive the message

```
Number ... not found
```

PROFILE [?|EXEC[ALL]|LIST[ALL]|LOAD[ALL]] PRO

Without parameters or with '?', the command PROFILE shows the profile files that *exaEdit* has searched for and used when starting. You can find details about this in section 3.1.26, *The Profile Files*.

The remaining parameters serve to

- execute again (EXEC),
- display in window (LIST), or

- load into the current workfiles (`LOAD`)

the commands of the profile files in use.

In each case, the parameter `ALL` specifies whether the command refers to all commands in the profile file or only to those that are executed when a new workfile is started (and which are therefore marked with an exclamation mark in column 1).

`QUIT|END` `Q|E`

The command `QUIT` leaves the editor. Before that action happens, it is checked whether you made changes in your workfiles and have not saved them yet. If that is the case, you are informed on it and you have the chance to continue the *exaEdit* session, for example to save your changed `workfiles` onto the data medium.

If there is only the workfile `MAIN` in your *exaEdit* session, you get the message

```
Changes not saved
Press J or Y to stop:
```

In line mode, the second line is:

```
Enter J or Y to stop:
```

If there are some other workfiles as well, the first line of the message reads as follows, instead:

```
Workfiles not saved: ...
```

In the window, the ... are replaced by the names of the workfiles.

If you decide not to save the changes, you simply press the key '`j`' or '`y`' (small or capital letters) and the editor immediately finishes its work. In opposition to this, if you hit any other key, the execution of the command `QUIT` is aborted and you can go on editing as usual.

Please note, that for answering this question it is sufficient to simply hit one key; it is not necessary to press the `return` key.

`REKEY [ base [incr]]|ON|OFF|?` `REK`

The command `REKEY` restores the equidistant numbering in the number area of the workfile.

The default setting is `REKEY 100 100`, resulting in the values used when a file is loaded into the workfile.

As initial value `base` you can choose a not negative digit (`0`, `1`, `2`, `3`, ...); as value for the difference `incr` you choose a positive digit (`1`, `2`, `3`, ...). A difference value `0` produces the message

```
Number 0 not allowed
```

If the line numbers would become larger than `99999999`, the command `REKEY` is rejected with the message

```
REKEY produces too large number
```

If you enter the command `REKEY` without parameters, the values that were last used are taken.

The parameters `ON` and `OFF` are checked and displayed but they have no meaning at present.

With the parameter `?` you ask *exaEdit* to show the current `REKEY` values in the window.

`REPLACE col1[col2]/string/[n]` R

Related commands: `CHANGE`, `CONCAT`

With the command `REPLACE` you insert a character string in the line(s) specified. Contrasting to `CHANGE`, the content of the character string to be replaced is irrelevant.

`REPLACE` substitutes a character string in the record of the current line or in `n` records starting at the current line.

You always have to specify the column, at which the character string has to be inserted. You manipulate the behaviour of `REPLACE` by specifying or omitting the second column as follows:

If you do not specify the end column, as many characters are inserted as the `string` is long. Characters that might be behind the inserted characters remain untouched.

If you specify the begin and the end column, `REPLACE` behaves as `CHANGE` with arbitrary content of the columns to be replaced. In other words, the character string that is limited by initial and final column is cut out of the line and the new `string` is inserted; the original content beyond the final column is moved to the left or to the right, so that it is adjacent to the new characters.

Examples: The current line contains the text

```
The weather has been fine.
```

The two commands

```
replace 13/had/
r22 25/foggy
```

give the following results:

```
The weather had been fine.
The weather had been foggy.
```

`RETURN` RET

Related commands: `POINT`, number command

The command `RETURN` positions the record marked by the `SET` command (compare the entry for `SET`) into the current line. The workfile section on the display is moved according to the change.

If there has not been a `SET` command before the `RETURN` command, you receive the message

```
SET storage unused
```

and the current line does not change.

If the record marked by `SET` does not exist any more, this message appears:

```
SET storage changed, return to previous record
```

and the previous record is moved into the current line.

It is not possible to jump from one workfile to another with `RETURN` because every workfile has its own `SET` storage.

`RLOCATE [col1[col2]] [/string/[H][I]]` RL

Related commands: `LOCATE`, also `NLOCATE` and `RNLOCATE/NRLOCATE`

This command serves to search for character strings backwardly. As parameter you specify a character string that has to be surrounded by delimiters, normally. Above, the slash ('/') is used as delimiter; any other character is allowed. In practice, the special key on the lower left of the keyboard has proved helpful. You only use another one if this character occurs in the character string to be searched for. You may omit the delimiter at the end of the string if the rest of the command line remains empty.

```
rlocate /abc/
```

launches a backwards search for the character string '`abc`',

```
rl -a/b-
```

launches a backwards search for the character string '`a/b`',

```
rl abc
```

launches a backwards search for the character string '`bc`' because '`a`' is the initial delimiter and the final delimiter is missing. `rl abca` would do a reverse search for the character string '`bc`', too.

It is important to note that the final delimiter is essential if you concatenate commands:

```
rl /abc/;-2
```

does a reverse search for the string '`abc`' and executes the command `-2` while

```
rl /abc ;-2
```

would search for the string '`abc ;-2`' backwardly.

The reverse search starts at the record before the one in the current line.

If the character string is found, the record containing it appears in the current line; the workfile section displayed is moved accordingly.

If the character string is not found, the following message appears:

```
Character string not found: ...
```

The ... are replaced by the search string and the current line remains unchanged.

If *exaEdit* could not find the character string until the first record of the workfile has been reached, the reverse search continues at the last record of the workfile, normally. The reverse search goes on until either the character string is found or the starting record of the reverse search is reached, again. The word 'normally' above means that the switch manipulated by the command `WRAP` (compare the entry there) has to be turned `ON`.

To show that the reverse search has been continued at the end of the workfile, you receive the following message in the window:

```
Search from end (wrap)
```

This message either is followed by the positioning of the workfile section on the display when the character string is found or it is followed by the message of failure when the character string could not be found.

Contrasting, if `WRAP OFF` is valid, the reverse search ends at the first workfile record, latest. If the reverse search was not successful, the messages

```
Begin of data
Character string not found: ...
```

are generated.

Frequently, it is necessary to search for the same character string for several times. In this case, it is sufficient to enter RLOCATE without any parameter: The last character string is used automatically.

The commands RLOCATE, LOCATE, NLOCATE and RNLOCATE/NRLOCATE use the same character string.

By default, the complete record will be searched backwardly. By giving two column numbers as first parameters the search is restricted to that area. A character string will then only be found if the given area contains it fully. If only one column number is given, the search area goes from that column to the end of the record. The command ZONE can also be used for column restrictions (see description). When both ZONE and the column number parameters are in effect, the latter one precedes.

If RLOCATE is used a second time, and then without any parameters, possible column restrictions are still in effect. But if a new search string is given, the restrictions are no longer in effect. A RLOCATE with a (new) column restriction after another RLOCATE will use the same search string as the first RLOCATE.

A hexadecimal reverse search can be done by giving the parameter

```
H
```

To this end, the given character string must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, the next record containing a tab sign (hexadecimal 09) should be found, the command

```
locate /09/ h
```

could be used.

When the parameter

```
I
```

(= 'case–**i**nsensitive') is given, the search for the character string will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
rlocate /ab/ i
```

will find either ab or Ab or aB or AB.

If you have to use the parameter I regularly, the use of the command CASE is recommended. CASE can force other commands to act as if the parameter I was given. For further information see CASE.

RNLOCATE|NRLOCATE [col1[col2]] [/string/[H][I]] RNL|RNL

Related commands: NLOCATE, also RLOCATE and LOCATE

RNLOCATE conducts a reverse search for the nearest line not containing the character string specified. As parameter you specify a character string. The character string has to be put in delimiters, normally. Above, the slash ('/') is used as delimiter; any other character is allowed as well. In practice, the special key on the lower right on your keyboard has proved to be useful. You only need a different delimiter if the character you have got used to is part of the character string specified. The delimiter at the end of the character string is omittable if the rest of the command line remains empty.

```
rnlocate /abc/
```

launches a backward search for the nearest line not containing the character string 'abc';

```
rnl -a/b-
```

launches a backward search for the nearest line not containing the character string ‘a/b’;

```
rnl abc
```

launches a reverse search for the nearest line not containing the character string ‘bc’ because ‘a’ is the delimiter at the beginning and the delimiter at the end is missing. `rnl abca` would also launch a reverse search for the nearest line that does not contain the character string ‘bc’.

Please mind the final delimiter when concatenating commands:

```
rnl /abc/;-2
```

reversely searches the nearest line that does not contain the string ‘abc’ and then executes the command -2 while

```
rnl /abc ;-2
```

backwardly searches the nearest line not containing the string ‘abc ;-2’.

The search starts at the record before the one in the current line.

If the nearest line that does not contain the specified character string is found, that line is positioned into the current line. The section of the workfile displayed in the window is moved correspondingly.

If *exaEdit* cannot find a line not containing the requested string, the message

```
Character string in all records: ...
```

appears; the ... are replaced by the specified character string; the current line remains unchanged.

If *exaEdit* finds the character string in every record until the first record of the workfile has been reached, the search continues at the last record, normally. The search goes on until either the character string is found or the record from which the search started is reached without success, again. In the sentence above, ‘normally’ means that the switch manipulated by WRAP (compare the entry for WRAP) is turned ON.

To show that the search has continued at the end of the workfile, *exaEdit* generates the message

```
Search from end (wrap)
```

This message is followed by either the positioning of the workfile section on the display or the message of failure mentioned above.

In contrast, if WRAP OFF is valid, the search ends at the first record of the workfile if the character string could not be identified. If the character string is there in every record that has been checked, these messages appear:

```
Begin of data
Character string in all records: ...
```

Frequently, it will be necessary to search for the same character string several times. Then it is enough to enter the command RNLOCATE without parameters; the last character string that has been searched for is used automatically.

The commands NRLOCATE, RNLOCATE, NLOCATE, LOCATE and RLOCATE use the same character string. The character string is the same for any workfile.

By default, the complete record will be searched. By giving two column numbers as first parameters the search is restricted to that area. A record will then be found if the given character string is not or not completely contained within the area. If only one column number is given, the search area goes from that column to the end of the record. The command ZONE can also be used for column restrictions (see description). When both ZONE and the column number parameters are in effect, the latter one precedes.

If RNLOCATE is used a second time, and then without any parameters, possible column restrictions are still in effect. But if a new search string is given, the restrictions are no longer in effect. A RNLOCATE with a (new) column restriction after another RNLOCATE will use the same search string as the first RNLOCATE.

You can search in hexadecimal form by giving the parameter

```
H
```

To this end, the given character string must be written in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, the nearest previous record not containing a tab sign (hexadecimal `09`) should be found, the command

```
rnlocate /09/ h
```

could be used.

When the parameter

```
I
```

(= 'case–**i**nsensitive') is given, the search for the character string will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
rnlocate /ab/ i
```

will find the nearest previous record containing neither `ab` nor `Ab` nor `aB` nor `AB`.

If you have to use the parameter `I` regularly, the use of the command `CASE` is recommended. `CASE` can force other commands to act as if the parameter `I` was given. For further information see `CASE`.

`SCOPE [?|ON|OFF]` `SC`

You can use this command to switch between window mode (default) or line mode (see section 3.1.20, *The Line Mode*).

`SCOPE ?` shows the setting of this switch.

`SCOPE` without further specifications means `SCOPE ON`.

`SEQUENCE [col1[col2]]/base[incr]]/n[N|R][F]` `SE`

This command inserts numbers in specified columns of `n` consecutive lines.

The default setting begins the numbering with 1 and increases it by 1 in each step. If, for example, you want to fill 5 lines with the numbers 1 to 5, you enter the command

```
sequence /1 1/ 5
```

Taking into account the default values, you could also write

```
sequence //5
```

You can, of course, choose different values for the base number and for the stepping size.

The numbers are written right-justified. As long as not specified differently, the needed column width is defined by the largest number that will be inserted and the field begins in column 1. By specifying the parameter `col1` you can determine the starting column for the inserted numbers. If you specify `col2` additionally, you determine the column width. By specifying `F` (`fill`), you demand that the columns of the inserted field are filled with zeros to the left. An example:

```
se5 7/8/3f
```

results in the following content for the columns 5 to 7:

```
008
009
010
```

The last remaining default to be described is `N` (new): After the current line, `n` new lines with the desired numbers are inserted. If you specify the parameter `R` (replace) instead, the respective columns (with the column width just as described above) in the next `n` lines, including the current line, are overwritten with the desired numbers. If there are not enough lines for this in the workfile, you get the message

```
End of data
```

The largest of the newly created numbers has to fit in the field width and may not exceed 8 digits. If this is violated, you get the message

```
SEQUENCE exceeds 99999999 or field width
```

`SET [?]` SET

If you enter `SET` without parameter, the number of the record in the current line is internally stored.

With the command `RETURN` you can go back to the record stored.

The specification of the parameter '`?`' effects that the record stored before is displayed in the dialogue zone. A leap to the record does not happen, then. You may use this parameter to find out where a `RETURN` command would lead to.

If the marked record does not exist any more, you receive the message:

```
SET storage changed, return to previous record
```

and the record previous to the one in the `SET` storage of the current workfile is displayed.

After having marked a record with the `SET` command, you can refer to that record using the symbolic line number

```
s
```

This works with all commands that use symbolic line numbers, e.g. `COPY`, `COUNT`, `DL`, `MOVE`, `POINT`, `SORT`, etc.

`SKEY [?|n]` SK

The command '`SKEY n`' determines that the number area on the window should be n digits wide. The default setting is 6, the minimum is 0, the maximum is 8.

With '`SKEY ?`' you receive the value of the current `SKEY` setting.

'`SKEY`' is equivalent to '`SKEY 6`'.

Please, note that a change of the `SKEY` value (also a change to the same value) causes a change of the `LWWIDTH` value (compare the entry for `LWWIDTH`).

`SORT [(line1 line2)][[A|D][I][N] where, [A|D][I][N] where,...]` SORT

With the command `SORT` you can sort records. You cannot abbreviate this command, thus you cannot type it by mistake; normally, this would be irreparable.

If the specification `(line1 line2)` is missing, the entire `workfile` is sorted. With the specification of `(line1 line2)` you can restrict the sorting to a part of the records. The numbers may be record numbers with or without leading zeros, or the specification may be of a symbolic line number as described at the entry of the command `COPY`. The first number must not be larger than the second.

Examples:

`sort (500 b) ...`	sorts the lines 500 to the end of the workfile
`sort (f n) ...`	sorts the lines from the first line to the line after the current line

Additionally, you may specify the orientation, type, and fields of the sorting. If you omit all further parameters, the records will be sorted in ascending, case-sensitive and alphabetical order (including numbers). Also the complete records will be used for comparison.

The orientation of the sort is specified by `A` (= ascending) for ascending sort and `D` (= descending) for descending sort. The orientation may be specified for every field separately. If you leave out the specification, *exaEdit* assumes `A` (ascending).

To turn off the case sensitivity you have to use the parameter `I` (case insensitive). This too has to be given for each sorting field where this is wanted.

If the fields that are to be sorted contain numbers, you have to use the parameter `N` (numerical) for each sorting field where this is required. When sorting numerically, the record or the given sorting field is interpreted as a number. The function `atoi` of the programming language C is used for this. This means the interpretation ends when the first character that is not a number is reached (this may be before the end of the record or the sorting field is reached). Additionally, this means the number being interpreted as 0 if no number has been found, an undefined interpretation if the number found was too large, the acceptance of white spaces (i.e. spaces, tabs, line feeds, page feeds, end of record characters) and the possibility of writing numbers using decimal points or not and using exponential format or not.

For the specification of the sorting fields (i.e. the `where` in the `SORT` syntax above) there are two possibilities:

```
initial-column length
```

or

```
initial-column : final-column
```

The two methods may be mixed. The parameters explained above (`A`, `D`, `I` and `N`) can be specified ahead of each sorting field. If you want the same parameters in effect for multiple sorting fields then put these fields in parentheses and specify the parameters just once ahead of those parentheses.

Finally, you may put one of the characters comma, semicolon, or slash between the field specifications in order to make them easier to read.

Some examples:

```
sort d
sort 16:18, 24:26
sort 1 5, d 6 5
sort(300b)16 8
sort d(1 3 4:10
sort n(1 5/6:10) d 11:15 / 16 5
```

The last example shows the use of 4 sorting fields, all of which are 5 characters in length. The first two fields are sorted numerically, the last two alphabetically. Only the third field is sorted descendingly, all others are sorted ascendingly.

If *exaEdit* has sorted successfully, you receive the message:

```
Sorted
```

Contrasting, if an error occurred, you get one of the following messages:

```
First number larger than second
```

In this case, in the parentheses with the line number specification, the value of the first numerical or symbolic line number is larger than the second.

```
Number ... not found
```

You specified the number of a line that does not exist.

```
There is no previous record
```

You used the symbolic line number `p` although there are no records previous to the record in the current line.

```
There is no next record
```

You used the symbolic line number `n` although there are no records behind the record in the current line.

```
You cannot sort the top line
```

You included the top line in the records to sort but it cannot be sorted.

```
Begin column larger than end column
```

You specified a sorting field with the property mentioned.

```
Sort fields overlap
```

There is at least one column that occurs in two field specifications.

`SSPLIT [col1[col2]] [/string/[E][H][I]]` SS

With the command `SSPLIT` (= 'string split') you can divide the record of the current line in two records. The easiest application is the specification of a character string where the record is split. In such a case, the line is always split at the beginning of the character string specified. For example, if the current line has the following content

```
and the dog watched the cat
```

the command

```
ssplit -the-
```

makes the following two lines out of the initial one:

```
and
the dog watched the cat
```

If you want to split the line above at the second '`the`' in the text, you can enter

```
ssplit 21 -the-
```

which means, you restrict the area where the character string is searched.

```
ssplit col1 col2 ...
```

only searches the columns `col1` to `col2` for the characters to split at.

```
ssplit col1 ...
```

searches the columns starting at `col1` for the characters.

If you enter an `E` after the character string, the dividing of the line does not happen at the beginning of the character string but at the end of the string.

```
ss-the-e
```

results in the two lines

```
and the
 dog watched the cat
```

You do not necessarily need to specify a character string to split at. You may also specify a column to divide a record:

```
ss 5
```

provides the result

```
and t
he dog watched the cat
```

The latter method is also useful to create an empty line before or after the current line:

`ss1` or `ss`

results in an empty record before the current line.

```
ss n
```

with a column `n` specified generates an empty line after the current line. The column `n` must be situated beyond the last character that is no space. For the example above, you may enter

```
ss33
```

The parameter

```
H
```

(= ‘hexadecimal’) allows you to enter the character string in hexadecimal form. Since 1 byte is always defined by 2 hexadecimal characters, *exaEdit* always demands that the amount of the entered hexadecimal characters is even. Otherwise *exaEdit* will generate the error message

```
Odd number of hex characters
```

Entering a character that is not hexadecimal results in the error message

```
Wrong hex character
```

If, for example, you would want to split a record at a tab sign (in hexadecimal form 09) that occurs somewhere in the record, you would enter

```
ssplit /09/ h
```

When the parameter

```
I
```

(= ‘case–**i**nsensitive’) is given, the search for the character string, where the record is to be split, will be case insensitive (i.e. no distinction between lower and upper case characters). This means that

```
ssplit /ab/ i
```

will split the record after either `ab` or `Ab` or `aB` or `AB`.

If you have to use the parameter `I` regularly, the use of the command `CASE` is recommended. `CASE` can force other commands to act as if the parameter `I` was given. For further information see `CASE`.

`TEST [NO]LOG[n]|[NO]DUMP|[NO]KEEP|SHOW|EXAMINE|REPAIR|[NO]MON` TE

The command `TEST` serves to obtain some information when debugging the editor. It is never needed when editing.

You will only use this command if the author of *exaEdit* will ask you to do so.

`TOP` T

The pointer to the current line is set to the top line. Since the current line has a fixed position in the window, the text slides down accordingly.

`TRANSLAT [(col1[col2])] [U|L|?] [n|ALL]` TR

This command (please note: just like with all other *exaEdit* commands, the maximal length of this command is 8 letters. This means there is no `E` at the end of it.) translates lower case in upper case letters (parameter `U`) and upper case in lower case letters (parameter `L`). In this context, 'letters' means the 26 letters of the common alphabet (i.e. no German umlauts).

If neither `U` nor `L` is given, `TRANSLAT` will do the same as it has done in the preceding call during the current *exaEdit* session. When starting *exaEdit*, the behaviour of `TRANSLAT` is switched to `U`. The status of the switch can be queried by using the parameter `?`, after which no other parameters may be given. The possible messages *exaEdit* returns are:

```
Translation to upper case
Translation to lower case
```

If no further parameters are given, all letters that can be translated will be translated.

If a number `n` is given as last parameter, `n` lines will be translated, beginning with the current line. If there are fewer than `n` lines left in the file, the usual message

```
End of data
```

will be generated. If the last given parameter is `ALL`, which can also be abbreviated, then all lines of the workfile will be translated, independent of the current line. In this case, the line pointer will stay on the current line, but if a number `n` was given, it will change to the last line that was edited.

All columns of the line to be changed will be translated if no other specification is given. By giving the first and the last column in parentheses as first parameter, the columns can be limited. If you give only one number within the parentheses, the translation will take place from this column to the end of the line. Another way of limiting the translation area is using the command `ZONE` (see command description). As usual, if both `ZONE` and a restriction in the `TRANSLAT` command are active, the latter one prevails.

`UP|-|BACK [n]` U|-|BA

The pointer to the current line is moved upward by `n` records; in the direction of the top of the workfile. Since the current line has a fixed position in the window, the text slides down correspondingly.

If you do not specify `n`, the value 1 is assumed.

If `n` is larger than the number of workfile records (including the top line), *exaEdit* writes

```
Begin of data
```

in the dialogue zone and the current line remains unchanged.

`WF [wfname|*[DELETE]] | [?[All]]` WF

`WF` is an abbreviation of the command `WORKFILE`.

With the command `WF` you may create workfiles, activate them, list or delete workfiles. A workfile name consists of one to eight letters or digits and it has to start with a letter.

The command

```
WF wfname
```

generates a workfile named `wfname`, if that workfile does not already exist. Otherwise the workfile `wfname` is activated by entering the command above. The active workfile is the one that is displayed in the window and that the editing commands have effect on.

With the command

```
WF wfname DELETE
```

the workfile `wfname` is deleted and the workfile `MAIN` becomes the active workfile. You may abbreviate `DELETE` with `D`. If you want to delete the active workfile (that must not be the workfile `MAIN`), you should enter `WF * DELETE`.

With the command

```
WF ?
```

the name of the active workfile is listed.

With the command

```
WF ? ALL
```

the names of all workfiles areas listed. You can abbreviate `ALL` with `A`.

`WIDTH [?|n|V]` WIDTH

With the command `WIDTH` you can define how the data in the file to edit should be divided in editor lines. This command cannot be abbreviated.

The default setting of `WIDTH` is the value `V` (= variable); that means that the editor lines correspond to the file records 1:1. In the file, records are separated by a special character. This character is the so-called newline character (`\n`, `X'0A'`). When carrying out the command `LOAD`, the newline characters are not taken into the workfile if `WIDTH V` is valid, which is supposed to be normal. When the command `FILE` is executed, the newline characters are added to every record of the workfile.

However, if you specified `WIDTH n`, the file to be read is split up into pieces of equal length, the length being `n`. Each of these pieces forms one record in the workfile. The newline characters in the file are read as normal data characters. When you write back onto the disk, the pieces from the workfile are aligned without the addition of other characters.

`WORKFILE [wfname|*[DELETE]] | [?[All]]` WF

With the command `WORKFILE` you may create workfiles, activate them, list or delete workfiles. A workfile name consists of one to eight letters or digits and it has to start with a letter.

The command

```
WORKFILE wfname
```

generates a workfile named `wfname`, if that workfile does not already exist. Otherwise the workfile `wfname` is activated by entering the command above. The active workfile is the one that is displayed in the window and that the editing commands have effect on.

With the command

```
WORKFILE wfname DELETE
```

the workfile `wfname` is deleted and the workfile `MAIN` becomes the active workfile. You may abbreviate `DELETE` with `D`. If you want to delete the active workfile (that must not be the workfile `MAIN`), you should enter `WORKFILE * DELETE`.

With the command

```
WORKFILE ?
```

the name of the active workfile is listed.

With the command

```
WORKFILE ? ALL
```

the names of all workfiles areas listed. You can abbreviate `ALL` with `A`.

`WRAP [?|ON|OFF]` `WR`

The command `WRAP` influences the behaviour of the command `LOCATE` and its related commands (i.e. `RLOCATE` etc.).

If `WRAP OFF` is valid, the search for character strings ends at the last or the first record of the workfile when the search string is not found.

On the other hand, if `WRAP ON` is valid, which is the default setting, either the search continues at the beginning of the file when the last record is reached or the search is continued at the end of the `workfile` when the first record is reached in reverse search.

With '`WRAP ?`' you can ask for the current setting of the `WRAP` switch.

`X [?|n|string]` `X`

Related command: `Y`

With the command `X` you can define an abbreviation for any desired sequence of commands and have them executed in a row.

With the command

```
X string
```

you define `X` as abbreviation of '`string`'; '`string`' has to consist of a valid command line (with one or more commands).

With the command

```
X n
```

you ask *exaEdit* to execute `X` n times. `X` alone executes the commands once. If you want to execute `X` without having it defined, the error message

```
X is not defined
```

occurs.

In '`string`' any command except `X` is allowed. The command `Y` is only allowed if it does not call `X`. In this case, the execution will be terminated at the appropriate point, generating the message:

```
Cancelled at recursive X
```

Example:

```
x n2;c/A/B/
```

For example, you might want to replace `A` with `B` in every second line. To do so, you have to call the commands `n2` and `c/A/B/` repeatedly. Instead of repeating the commands, you put them, combined, in `X`. Then you call '`x 100`', which executes the content of `X` 100 times. As soon as one of the commands contained in `X` terminates with a warning or an error message (`End of data`, `String not found`, etc.), the execution of `X` will be terminated.

With the command

```
X ?
```

you ask *exaEdit* to write the current definition of `X` in the window. You may use this command, for example, to change the content of `X` slightly. You move the cursor to the output line, make your changes and redefine `X` by pressing the `return` key.

`Y [?|n|string]` `Y`

Related command: `X`

With the command `Y` you can define an abbreviation for any desired sequence of commands and have them executed in a row.

With the command

```
Y string
```

you define `Y` as abbreviation of '`string`'; '`string`' has to consist of a valid command line (with one or more commands).

With the command

```
Y n
```

you ask *exaEdit* to execute `Y` n times. `Y` alone executes the commands once. If you want to execute `Y` without having it defined, the error message

```
Y is not defined
```

occurs.

In '`string`' any command except `Y` is allowed. The command `X` is only allowed if it does not call `Y`. In this case, the execution will be terminated at the appropriate point, generating the message:

```
Cancelled at recursive Y
```

Example:

```
y n2;c/A/B/
```

For example, you might want to replace `A` with `B` in every second line. To do so, you have to call the commands `n2` and `c/A/B/` repeatedly. Instead of repeating the commands, you put them, combined, in `Y`. Then you call '`y 100`', which executes the content of `Y` 100 times. As soon as one of the commands contained in `Y` terminates with a warning or an error message (`End of data`, `String not found`, etc.), the execution of `Y` will be terminated.

With the command

```
Y ?
```

you ask *exaEdit* to write the current definition of `Y` in the window. You may use this command, for example, to change the content of `Y` slightly. You move the cursor to the output line, make your changes and redefine `Y` by pressing the `return` key.

ZONE [?|n[m]] Z

There are commands, the result of which depends on the location of a certain character string within the line that has to be worked on. To make the use of the command ZONE easier to understand, we will have a look at the command SSPLIT, at first.

```
ssplit /abc/
```

means, as you know, that the current line should be split in two parts at the beginning of the character string 'abc'. Sometimes you might want that this only happens if 'abc' occurs in column 10 but not if the string comes before column 10. The command SSPLIT offers the option to specify the desired area directly, but in some cases it might be more appropriate to restrict both the command SSPLIT and similar commands to a certain column area. The command ZONE provides this option.

Specifying one column in the ZONE command, you restrict the operating area in such a way that it starts at the column specified and extends to the end of the record.

By specifying two columns you define the beginning and the end of the operation area within the record.

ZONE ? shows the current setting.

ZONE without parameter restores the default setting without any restriction.

The working area set by ZONE affects the commands CHANGE, the LOCATE family, SSPLIT, and TRANSLAT.

3.3 The Prefix Commands

Prefix commands are extremely abbreviated commands which are entered in the number prefix of the line of the workfile they refer to, not in the dialogue zone of *exaEdit*.

For example, if you position the cursor in the number area of a workfile, enter the character " there and then press enter, the respective prefix command is executed and doubles the line thus marked.

You can enter any desired prefix command in multiple number areas in the part of the workfile that is currently shown. They will all be executed consecutively (in top-down order).

Additionally, before pressing enter, you can enter a normal *exaEdit* command in the dialogue zone. When you press enter, first the prefix commands and then the command in the dialogue zone will be executed.

At the moment, there are the following prefix command (additions are in planning):

" This prefix command doubles the marked line.

i This prefix command inserts an empty line after the marked line.

d This prefix command deletes the line marked. To delete several lines, you can mark as many lines as you desire with d and delete them all with pressing enter once.

If the lines to be deleted are consecutive ones, you can delete all of them at once with a special form of this prefix command:

```
dn
```

Here, n lines starting with the one you marked, are deleted. Between d and the number no spaces are allowed.

Since there are digits in the number area, you have to consider this: If you enter e.g. d3 without modifying another column of the number area, *exaEdit* recognizes the command d3 no matter if there are any digits after the number 3. However, if you have modified any other columns (no matter if you restored the original content or not), there might be misunderstandings between *exaEdit* and yourself about how many lines there are to be deleted. To be on the safe side, you should enter the desired number of lines at the right end of the number area, or follow the number with a space.

`dd` This prefix command comes as a pair. With the following Enter all records from the first to the second `dd` will be deleted, the marked records included. If you have marked only one record with `dd`, then this mark will be ignored.

It is also possible to enter several pairs of `dd` prefix commands and let them be executed with one Enter. If the number of `dd` marks is odd then the last `dd` will be ignored as well.

With the command `MARK` you can see where a single `dd` is or you can delete it (only the mark, the record will be kept).

Chapter 4

`exaEdit` Synopsis

In this chapter, functions for editing are listed in alphabetical order. For each entry there is an explanation of the methods to perform the function.

Please note the different font faces in the following text.

Commands are written in equidistant font with capital and small letters. The capital letters of a command word mark its minimal abbreviation. Of course, you may enter the commands in either small or capital letters, as you like.

Italic (equidistant) font is used for parts of a command (name, numbers character strings etc.). You have to replace these parts with your own characters.

Square brackets [ ] enclose optional specifications; you decide whether you use or omit them.

Browsing

- Keys page ↑ and page ↓ browse 1 page.
- Keys F7 and F8 browse 1 page.
- Keys F10 and F11 browse 1/2 page.

Changing the command separator

- Command `CMDsep`.

Changing the display

- Command `HEXa` to show current line in hexadecimal form.
- Command `CODepage` (with parameters) for German special characters in Windows operating systems.
- Command `LWwidth` *n* for manipulating the line break of long records.
- Command `SKey` *n* to set the width of the number area.
- Command `SCope OFF|ON` to switch between window mode and line mode.

Changing the language

- Command `LAnguage`.

Columns replace

- Command `Change`.
- Command `Replace`.

Concatenating two records

- Command `CONcat`.

Copying columns

- Command `CCopy`.

Copying records

- Command `COpy`.
- Prefix Command `"` doubles the marked record.

Counting records

- Command `COUnt` *num1* [*num2*].
- See also total number of lines in the status line.

Deleting a record

- Prefix command `d` deletes the record of the marked line.
- Command `DElete` deletes the record of the current line.
- Command `DL` *num1* deletes the record with the number *num1*.

Deleting a workfile

- Command `WF` *wfname* `Del`.

Deleting all records of a workfile

- Command `DElete All`.

Deleting columns

- Command `CDelete`.

Deleting several records

- Prefix command `d`*n* deletes *n* records starting with the record of the current line.
- Prefix command `dd` deletes the area of records labelled with it.
- Command `DElete` *nnn* deletes *nnn* records starting with the record of the current line.
- Command `DL` *num1* *num2* deletes the records from *num1* to *num2*.

Doubling a record

- If the record is in the current line, a simple `COpy` will suffice.
- Prefix command `"`.

Empty line to be inserted

- Compare 'Inserting an empty line'

Exchanging two records

- If the current line shows the second record, the command `Move p` will suffice.

Inserting an empty line

- Command `SSplit` generates an empty line before the current line.
- Command `SSplit` *n* with sufficiently large *n* creates an empty line after the current line.
- Command `Input //` generates an empty line after the current line.
- Switch to the insert mode and enter a line that only contains an empty space.
- Prefix command `i` creates an empty line below the current line

Line to be deleted

- Compare 'Deleting a record'.

Lines to be deleted

- Compare 'Deleting all records of a workfile'.
- Compare 'Deleting several records'.

Line to be inserted

- Compare 'Records to be inserted'.

Marking a record

- Command `SET`.

Moving Columns

- Command `CMove`.

Moving a record

- Command `Move`.

Records to be concatenated

- Command `CONCAT` concatenates the record of the current line with the following one.

Record to be deleted

- Prefix command `d` deletes the record of the marked line.
- Command `DElete` deletes the record of the current line.
- Command `DL` *num1* deletes the record with the number *num1*.

Records to be deleted

- Prefix command `d`*n* deletes *n* records starting at the record of the current line.
- Prefix command `dd` deletes the area of lines marked with it (in preparation).
- Command `DElete` *nnn* deletes *nnn* records starting with the record of the current line.
- Command `DL` *num1 num2* deletes the records from *num1* to *num2*.

Records to be inserted

- Command `Input /line/` .
- Enter the Input mode (command `Input`), hit Enter twice after written a line.

Searching of characters or strings

- Command `Locate` to search in forward direction.
- Command `RLocate` to search in reverse direction.
- Command `NLocate` to search for lines that must not contain the character or string.
- Command `NRLocate` or `RNLocate` for a reverse search for lines that must not contain the character or string.

Sorting all or some records

- Command `SORT` (possibly with parameters).

Splitting a record

- Command `SSplit` (with parameters).

Tabulator character to be removed

- Command `EXPAND` expands the tab characters to spaces in the specified records.

Tabulator character to be set

- Command `COMPRESS` compresses adequate sequences of spaces to tab characters in the specified records.

View help texts

- Command `Help` shows all available commands.
- Command `Help cmd` gives a short help about `cmd`.
- Command `MANual` shows the complete *exaEdit*–manual.

View manual

- Command `MANUAL`.

Chapter 5

The `exaEdit` Messages

In this chapter you find most of the messages that *exaEdit* can generate listed in an alphabetical order. Beneath the messages, there are the page(s) that offer a more detailed description.

Page numbers up to 32, including, refer to chapter 2, *First Steps* while the pages from 33 upward belong to chapter 3, *The Editor and its Commands*.

`... files loaded`
40

`... subdirectories skipped`
40

`... times changed`
79

`1 file loaded`
40

`1 subdirectory skipped`
40

`A directory cannot be edited`
37, 43

`access errno = ...`
38, 44

`Access not allowed`
37, 43

`ATTENTION: Data not saved!`
26, 43

`Begin column larger than end column`

This error message can occur with many *exaEdit* commands, but is described only here:

If a command needs a parameter which restricts the affected columns, the left column number has to given prior to the right one. Reversing this leads to the given error message. Mind that the column parameters may be parameter variables.

`Begin of data`

This error message can occur with many *exaEdit* commands, but is described only here:

A *exaEdit* positioning command demands that the current line is positioned before the first line of the workfile. In contrast to searching (see the description of `RLOCATE`), it is not possible for positioning commands to continue with the last line after stepping over the first line of the workfile.

`Bus error`
70

`Cancelled at recursive X`
61, 113

`Cancelled at recursive Y`
61, 114

`Case-insensitive`
76

`Case-sensitive`
76

`Changes not saved`
26, 44, 86, 101

`Character string in all records: ...`
96, 97, 98, 99, 105, 105

`Character string not found: ...`
79, 92, 92, 103, 103

`Character string too long`

This error message can occur with many *exaEdit* commands, but is described only here:

If a *exaEdit* command needs a string for a parameter, the programming language reserves a certain amount of memory for that string. If the string is too long to be contained in that memory, the programming language generates an error. This results in the *exaEdit* message `string too long`. Since the amount of memory maximally available is large enough, this does not restrict working with *exaEdit*.

`CODEPAGE is only for Windows systems`
81

`Compress #...`
82

`Compressed n times in m records by k blanks`
82

`Curses: ..., Characters: ..., Escape: ..., Function: ...`
68

`Data saved`
23, 43, 70

`Data set may be read only`
43

`Data set not opened (does not exist?)`
38

`Directory not found`
40, 43

`Directory not opened`
40

`DOS`
81

`End of data`

This error message can occur with many *exaEdit* commands, but is described only here:

This error message has two possible reasons:

One would be a *exaEdit* positioning command like `DOWN` that demands the current line to be positioned past the last line of the workfile. In contrast to searching (see the description of `LOCATE`), it is not possible for positioning commands to continue with the first line after stepping over the last line of the workfile.

The second possible reason would be a command that tries to obtain data from behind the last line of the workfile during its execution. This could, for example, happen when you use the command `CHANGE` together with the parameter `n`.

`Number too large`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used a parameter variable for a string parameter, but the parameter variable used was defined for (line) numbers, and not for strings.

`Parameter variable no character string`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used a parameter variable for a string parameter, but the parameter variable used was defined for (line) numbers, and not for strings.

`Parameter variable not defined`

This error message can occur with many *exaEdit* commands, but is described only here:

Parameter variables that are not included in *exaEdit* by default have to be defined via the command `&` before use.

`Parameter variable not numerical`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used a parameter variable for a number parameter (column, line, etc.), but the used parameter variable was defined for a character string and not for numbers.

`SET storage invalid`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used the symbolic line number `s` with a command such as `COPY`. `s` refers to the record that has been marked with the command `SET`. The command `SET` has been used in the current workfile, but the record it referred to has been deleted in the meantime. As a consequence, the symbolic line number `s` cannot be used any more. Note that some commands create the message `SET storage changed, return to previous record` instead.

`SET storage unused`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used the symolic line number `s` in a command, e.g. `COPY`. `s` refers to the record that has been marked via the `SET` command. But this command has either not yet been given in the current workfile or the complete content of the workfile has been deleted in the meantime. Due to that, `s` is not defined, so you receive an error message. The command `RETURN` can produce the same error message.

`There is no column 0`

This error message can occur with many *exaEdit* commands, but is described only here:

Some *exaEdit* commands need a column specification for a parameter. The columns are counted from number 1 onwards, a column with the number 0 is not allowed. Mind that a parameter variable also can have the value 0 and thus result in the given error message.

`There is no next record`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used the symbolic line number `n` with a command, e.g. `MOVE`. `n` refers to the record following the current record, but since the current line is the last line of the workfile, there is no next record.

`There is no previous record`

This error message can occur with many *exaEdit* commands, but is described only here:

You have used the symbolic line number `p` with a command, e.g. `COUNT`. `p` refers to the line prior to the current one, but since the current line is the first line of the workfile, there is no previous record.

... can be replaced with: `change`, `concatenate`, `delete`, `move` or `sort`.

This error message can occur with many *exaEdit* commands, but is described only here:

The execution of the command you have given would concern the top line, while it is not possible for the given command to work with the top line, e.g. `copy t 500`. This can often be corrected by replacing the symbolic line number `t`, which denotes the top line, with the symbolic line number `f`, which denotes the first line.

Index

Page numbers up to 32, including, refer to chapter 2, *First Steps*, while page numbers from 33 upward are part of chapter 3, *The Editor and its Commands*.

Index