Der Editor exaEdit

Benutzeranleitung

Version 02.1
17. Februar 2009

Peter Preus

Diese Anleitung wurde mittels LaTeX erstellt.

Version 02.1 — 17. Februar 2009

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

Vorwort

Wahrscheinlich geht es dieser Anleitung auch nicht besser als den meisten anderen: sie wird viel zu selten gelesen, weil sie (wenn überhaupt!) meist erst dann zu Rate gezogen wird, wenn Online–Hilfen, Ausprobieren und Herumfragen nicht weiterführen. Damit für die wenigen Male, die diese Anleitung gelesen wird, sie dir, liebe Leserin und lieber Leser, nicht ganz so unpersönlich fremd bleibt, habe ich eine möglichst direkte Ansprache gewählt. Diese Ansprache soll aber nicht nur Stilmittel sein, sondern daneben noch andeuten, dass ich mir während der Programmierung immer bewusst war, dass ich etwas produziere, was von Menschen benutzt wird. Außerdem kann ich auf diese Weise vielleicht die Hemmschwelle, mich mit Fragen, Hinweisen oder Vorschlägen zu *exaEdit* zu kontaktieren, möglichst niedrig halten, weil ich dadurch effektiver an der Verbesserung und dem weiteren Ausbau von *exaEdit* arbeiten kann.

Die vorliegende Anleitung besteht aus 5 Kapiteln.

Kapitel 1, *Überblick*, liefert einen groben Überblick über die Eigenschaften und Fähigkeiten des Editors, ohne im einzelnen Details aufzuzeigen.

Kapitel 2, *Erste Schritte*, ist ein Tutorial zum Erlernen des Editors, das mit vielen Bildern und Beispielen insbesondere für ein Selbststudium geeignet ist. Es werden nur die wichtigsten Elemente des Editors dargestellt. Ausführliche Informationen findest du dann bei Bedarf in den Kapiteln 3 bis 5.

Kapitel 3, *Der Editor und seine Befehle*, enthält eine vollständige Beschreibung aller Eigenschaften des Editors. Dieses Kapitel ist in allen Zweifelsfällen der Benutzung maßgebend. Du solltest es mindestens einmal ganz gelesen haben, wenn du den Editor souverän benutzen können möchtest.

Kapitel 4, *Das exaEdit–Lexikon*, ist zum schnellen Nachschlagen gedacht. Die Stichwörter sind Editorfunktionen (etwa „Löschen einer Zeile“), zu denen dann die oft mehrfach vorhandenen Realisierungen beschrieben werden.

Kapitel 5, *Die exaEdit–Meldungen*, bringt eine ziemlich vollständige Liste der Meldungen, die von *exaEdit* erzeugt werden können, zusammen mit der Angabe der Seite(n), auf denen sie näher beschrieben sind.

Die Anleitung beschreibt *exaEdit* in der Version 02.

Inhaltsverzeichnis

Kapitel 1

Überblick

Hier findest du nach der Information, wie du *exaEdit* für deinen Gebrauch erhältst, in zusammengefasster Form die Haupteigenschaften von *exaEdit*. Zum Verständnis dieses Kapitels solltest du bereits wissen, was ein Editor ist, und wie Editoren im allgemeinen funktionieren. Falls du diese Kenntnisse noch nicht hast, solltest du dieses Kapitel vielleicht lieber überspringen und statt dessen sogleich mit dem einführenden Kapitel, *Erste Schritte*, beginnen.

1.1 Wie ist exaEdit erhältlich?

exaEdit ist ein Editor, den es für viele Betriebssysteme gibt. Aktuelle Versionen gibt es für die UNIX–Betriebssysteme AIX und Linux und für das PC–Betriebssystem Windows (alle 32-Bit-Versionen). Für die Systeme HP–UX, IRIX, OSF1, OS/2 und SunOS gibt es *exaEdit* nur in älteren Versionen, da diese Systeme dem Autor von *exaEdit* derzeit nicht zur Verfügung stehen. Bei Bedarf (auch für hier nicht genannte Betriebssysteme) kannst du dich deswegen jederzeit an den Autor wenden. Meistens ist es vergleichsweise einfach, *exaEdit* für neue Betriebssysteme zu erstellen.

Die Programmnummer setzt sich aus einer 2-stelligen Versionsnummer und einem Aktualisierungsbuchstaben zusammen, etwa 02B. Werden in *exaEdit* lediglich Fehler behoben, ändert sich in der Programmnummer nur der Buchstabe. Werden dagegen in *exaEdit* neue Eigenschaften und Funktionen eingebaut, so wird die Versionsnummer weitergezählt, und die Aktualisierungsbuchstaben beginen wieder von vorne.

Die aktuelle Programmnummer von *exaEdit* ist 02B. Alle weiteren Informationen zum Herunterladen und Installieren von *exaEdit* findest du auf der Seite

```
http://exaedit.de/
```

1.2 Die Geschichte

Der Editor *exaEdit* — wie er heute vorliegt — wurde nicht in einem Anlauf entworfen und verwirklicht, sondern ist im Laufe der Zeit gewachsen.

Der erste dem jetzigen Autor bekannte Vorläufer hieß *XEDIT* und tauchte im Jahre 1975 im Rechenzentrum der Universität Heidelberg auf. Dieses *XEDIT* hat nichts mit den heute allgemein bekannten Editoren gleichen Namens zu tun, die später mehr oder weniger weltweite Verbreitung erfahren haben, sondern war eine lokale Erscheinung im Rechenzentrum der Universität Heidelberg. Auch der restliche Teil dieses Absatzes beschränkt sich strikt auf die Heidelberger Verhältnisse. *XEDIT* war (recht bald) ein Fullscreen–Editor für das Timesharing–System TSO des Betriebssystems MVS der Firma IBM. Da die Benutzeroberfläche von *XEDIT* ansprechend war, wurde sie ziemlich vollständig als Benutzeroberfläche des Editors HADES (in frühen Tagen AMOS genannt) nachgebaut, der ein wesentlicher Bestandteil des gleichnamigen Timesharing–Systems war (ebenfalls unter MVS betrieben).

Als der jetzige *exaEdit*–Autor Ende 1992 mit der Notwendigkeit konfrontiert wurde, sich dem Betriebssystem Unix zuzuwenden, erlebte er — nicht nur, aber vor allem auf dem Gebiet der Editoren — einen Kulturschock (Stichwörter

`vi`, `emacs`), aus dem er mit der festen Absicht hervorging, die nach seiner Ansicht bewährten Teile von *XEDIT* (wie Konzept und Benutzeroberfläche) ins Unix hinüberzuretten. Da ein mit den Software–Methoden der 70er Jahre erstelltes großes und komplexes Programm nicht übertragbar ist, bedeutete dies ein Neuschreiben des Editors. Nicht aus Neigung, sondern als Ergebnis der Wahl des kleinsten Übels wurde als Programmiersprache für dieses Unterfangen C gewählt.

Als Programmname wurde zunächst *xed* gewählt, um einen Unterschied zu anderen Editoren namens *xedit* im Betriebssystem VM der IBM und im Betriebssystem Unix zu haben. Da sich jedoch herausstellte, dass *xed* häufig als Kurzform (etwa in Verzeichnissen) für Editoren namens *XEDIT* in Unix verwendet wurde, wurde der hier zu beschreibende Editor im März 1996 in *pedit* umbenannt.

Seit Mitte 1993 war der Editor im Betriebssystem AIX benutzbar, obwohl er natürlich seither umfangreiche Verbesserungen und Erweiterungen erfahren hat.

Eine stabile Version (`10A`) wurde im März 1994 im Universitätsrechenzentrum Heidelberg zur Benutzung im Betriebssystem AIX angeboten und auch als public domain software bereitgestellt.

Nach und nach wurden Versionen für die Unix–Betriebssysteme HP–UX, IRIX, Linux, OSF1 und SunOS, für OS/2 und für alle 32bit–Windows–Systeme entwickelt.

Die 1996 erfolgte Umbenennung von *xed* in *pedit* erwies sich im Nachhinein leider nicht als besonders praktisch, da *pedit* ebenfalls ein weltweit häufig verwendeter Name für die verschiedensten Editoren darstellt. Daher wurde im Jahr 2004 der Name erneut geändert, diesmal in

`exaEdit`

Da hiermit auch ein Internet-Domain-Name verbunden ist (wenn auch nur für die Endung `.de`), besteht Hoffnung, dass es nicht so schnell wieder zu Mehrdeutigkeiten kommt.

Derzeit gültig ist die Version 02.

1.3 Die Konzepte

Hier erfährst du etwas über die große Linie, von der sich die Autorinnen und Autoren der *exaEdit*–Vorgänger und des heutigen *exaEdit* bei der Entwicklung des Editors leiten ließen. Weitere Eigenschaften von *exaEdit*, für die der Begriff *Konzept* zu hoch gegriffen wäre, sind im Abschnitt 1.4, *Weitere Eigenschaften*, beschrieben.

1.3.1 Der workfile

exaEdit ist ein Datei–Editor, bearbeitet aber Dateien, wie du sie etwa von einer Festplatte her kennst, nicht direkt. Statt dessen werden zu Anfang einer *exaEdit*–Sitzung die Daten einer Datei satzweise vom Datenträger gelesen und im Hauptspeicher des Computers abgelegt. Alle Änderungen nimmst du dann an diesem Hauptspeicher–Abbild der Datei vor. Am Ende der *exaEdit*–Sitzung kannst du dann die (geänderte) Hauptspeicherversion der Datei auf den Datenträger zurückschreiben. Das Abbild der Datei im Hauptspeicher nennen wir

workfile

Dieses Konzept bietet den Vorteil, dass sich Fehler, die du beim Editieren der Daten machst, nicht automatisch auf die Originaldatei auswirken. Andererseits sind natürlich alle Änderungen verloren, wenn der Computer oder sein Betriebssystem ausfallen, bevor du die Änderungen gespeichert hast. Ein solcher Fall wird jedoch sehr selten auftreten; außerdem gibt es Möglichkeiten, den Schaden sehr stark zu begrenzen.

exaEdit bietet die Möglichkeit, mehrere workfiles (mit verschiedenen oder auch gleichen Dateien) auf einmal zu haben. Einer dieser workfiles ist der jeweils aktuelle, auf den sich die gegebenen Befehle beziehen und der im Fenster zu sehen ist.

1.3.2 Fenster– und Zeilenmodus

exaEdit ist ein Ganzfenster–Editor (full–screen), d.h. er belegt das ganze Fenster mit seiner Ausgabe und nimmt (fast) vom ganzen Fenster Eingabe entgegen. Ein typisches Bild ist in einem Fenster der Größe 24×80 etwa dieses

```
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
```

Der obere Teil des Bildes ist die Datenzone, in der der workfile (zumeist natürlich nur ein Teil davon) gezeigt wird. Der untere Teil nach der Linealzeile ist die Dialogzone, in der du (aber nicht nur dort) Befehle eingeben kannst und in der *exaEdit* Antworten gibt oder Fragen stellt. Die letzte Zeile ist die Statuszeile, die über bestimmte Zustände von *exaEdit* Auskunft gibt.

Die Größe des Fensters ist für *exaEdit* nahezu beliebig, du kannst sie auch während einer *exaEdit*–Sitzung beliebig ändern (wenn dein Betriebssystem dies zulässt, z.B. X–Window), *exaEdit* passt sich sofort an.

Welchen vertikalen Ausschnitt *exaEdit* von deinem workfile zeigt, ist im nächsten Abschnitt beschrieben.

Horizontal werden als Voreinstellung jeweils die Satzanfänge gezeigt. Sind Sätze zu lang, werden die überschüssigen Daten in Fortsetzungszeilen untergebracht. Du kannst aber auch die „logische“ Fensterbreite beliebig einstellen, also damit festlegen, welche Daten in Fortsetzungszeilen gezeigt werden sollen. Schließlich kannst du (aber das gibt es derzeit in *exaEdit* noch nicht!) den horizontalen Ausschnitt beliebig über die logische Fensterbreite verschieben.

Kann *exaEdit* kein Ganzfenster benutzen, so arbeitet der Editor im Zeilenmodus, den du auch explizit verlangen kannst.

1.3.3 Die aktuelle Zeile

Eine der wichtigsten *exaEdit*–Eigenschaften ist die automatisch nachgeführte aktuelle Zeile.

Die mittlere Zeile der Datenzone des *exaEdit*–Fensters, die in der Regel auch optisch hervorgehoben ist, heißt aktuelle Zeile.

Sie ist Ziel oder Ausgangspunkt von Editorbefehlen. Zum Beispiel bedeutet der Befehl

```
copy
```

„kopiere die aktuelle Zeile (hinter die aktuelle Zeile)“, bewirkt also ein Verdoppeln der Zeile.

Ein anderes Beispiel. Der Befehl

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

besagt „ändere in 6 Zeilen, beginnend mit der aktuellen Zeile, die Zeichenkette 'abc' in 'xyz' um".

Die aktuelle Zeile steht im allgemeinen nicht fest, sondern wird automatisch nachgeführt. Dies bedeutet grob gesagt, dass die jeweils zuletzt geänderte Zeile des workfiles zur aktuellen Zeile wird.

Dahinter steht folgendes Konzept: *exaEdit* geht davon aus, dass du in dem workfile, von oben beginnend, Änderung für Änderung vornimmst, bis alles fertig ist (so wie du etwa einen Brief korrigierst). In jedem Augenblick ist also vermutlich das Zentrum deines Interesses die zuletzt gemachte Änderung, die deshalb in die Mitte des Fensters gerückt wird.

1.3.4 Die Eingabe

exaEdit kennt 3 Möglichkeiten für die Eingabe:

- direkte Änderung der Daten
- (Zeilen–)Befehle
- Präfixbefehle

Direkte Änderung der Daten bedeutet, dass du mit dem Cursor in das Datenfeld fährst und dort Zeichen übertippst, löschst oder einfügst.

Zeilenbefehle (meist nur Befehle genannt) sind Anweisungen an den Editor zum Ändern, Einfügen und Löschen von Zeichen oder Sätzen (Beispiele etwa das `copy` oder `change` von oben), zum Positionieren der aktuellen Zeile, zum Laden oder Speichern von workfiles und vieles mehr. Zeilenbefehle kannst du im Prinzip an beliebiger Stelle des Fensters eingeben, wenn es auch meistens die Dialogzone sein dürfte.

Präfixbefehle sind Befehle zum Einfügen, Verdoppeln, Löschen, Kopieren und Verschieben von Zeilen, die du am Anfang der Zeilen in der Datenzone (vgl. Bild in 1.3.2) im Präfixfeld (wo die Zeilennummer steht) eingibst.

Das wichtigste Prinzip von *exaEdit* bei der Eingabe ist, dass alles, was du eintippst, solange noch ohne jegliche Folgen zurücknehmbar oder änderbar ist, bis du die `Enter`–Taste drückst. Erst dann wird deine Eingabe verarbeitet.

Dabei kannst du für die Eingabe, die du mit dem Drücken der `Enter`–Taste wirksam machen willst, alle drei Formen der Eingabe beliebig kombinieren, wenn natürlich auch nicht alle Kombinationen sinnvoll sind. Häufig jedoch ist es vorteilhaft, Direktänderungen und Präfixbefehle mit einem einzigen `Enter` abschicken zu können.

1.3.5 Die Tastaturbenutzung

exaEdit nutzt die Tastatur in konservativer Weise. Dies bedeutet, dass es keine Tastenkombinationen gibt (die ja meist auswendig gelernt werden müssen), um bestimmte Funktionen des Editors anzustoßen. Zwar musst du dir natürlich auch die Befehle irgendwie merken, nur dürfte dies leichter sein, da es sich um bekannte Begriffe aus der englischen Sprache handelt (siehe auch den Abschnitt 3.2, *Die Befehle*).

Zu einem vernünftigen Arbeiten mit *exaEdit* gehören außer den Zeichentasten (dazu gehören auch die mit „`Alt`" usw. modifizierten) und der `Enter`–Taste

- die Cursorbewegungstasten (↑, ↓, →, ←),
- die Löschtaste (`Entf`, `DEL`, ⌦) oder die Backspace–Taste,
- die Einfügetaste (`Einfg`, `INS`, ⎀) oder eine *exaEdit*–Funktion.

Zu einem bequemen Arbeiten brauchst du in vielen Fällen die Tasten `F1` bis `F12` und die Taste `Pos 1` (`Home`), sie sind aber nicht lebensnotwendig. Umgekehrt kannst du die Funktionen der Lösch–, Einfüge– und Cursor–Tasten (neben anderen) auf die `F`–Tasten definieren.

Weitere Tasten werden von *exaEdit* erkannt.

1.3.6 Die Befehlssyntax

(Zeilen–)Befehle von *exaEdit* sind von der Gestalt

```
befehl operand operand ...
```

Dabei sind die Befehle und die Operanden mit festem Wert meistens englische Wörter, so dass du sie dir leicht merken kannst.

In *exaEdit*–Befehlen darfst du alles beliebig abkürzen (von hinten anfangend), solange der Befehl eindeutig bleibt. Dabei haben Befehle festgelegte Minimalabkürzungen. So ist beispielsweise die Minimalabkürzung von `copy` die Buchstabenfolge `co`, während `change` sogar bis auf `c` abgekürzt werden kann (`change` wird im allgemeinen häufiger als `copy` verwendet).

Praktisch ist auch, dass Leerzeichen weggelassen werden können, wenn dies die Erkennung des Befehls und seiner Operanden nicht beeinträchtigt. So kannst du zum Beispiel

```
COPY 1000 B
```

erstzen durch

```
CO1000B
```

1.4 Weitere Eigenschaften

1.4.1 Editieren von Verzeichnissen

exaEdit erlaubt es, alle Dateien eines Verzeichnisses gleichzeitig zu editieren. Die einzelnen Dateien werden zu diesem Zweck unter einer jeweils eindeutigen Überschrift zusammen in einen einzigen workfile geladen und beim Zurückschreiben auf den Datenträger auf Wunsch wieder auseinandergenommen. Zusätzlich können die zu bearbeitenden Dateien aufgrund von Namensbestandteilen ausgewählt werden. Einzelheiten findest du im Abschnitt 3.1.3.5, *Laden aller Dateien eines Verzeichnisses*.

1.4.2 Programmierbarkeit

exaEdit weist einige Eigenschaften auf, die es ermöglichen, Editiervorgänge programmiert ablaufen zu lassen. *exaEdit* kommt dabei einer Skriptsprache schon ziemlich nahe. Einzelheiten dazu findest ab dem Abschnitt 3.1.21, *Programmieren des Editors*.

1.4.3 Profildateien

Eine Profildatei für *exaEdit* ist eine Datei, in der sich *exaEdit*–Befehle befinden, die zu Beginn der *exaEdit*–Sitzung oder beim Start eines neuen workfiles ausgeführt werden. Mit diesen Befehlen kannst du Einstellungen, die du ändern möchtest, ein für allemal ändern.

Es kann eine Installationsprofildatei geben, die normalerweise für alle gleich ist, die dasselbe installierte Editorprogramm verwenden (zum Beispiel bei einer Workstation). Du kannst aber auch private Profildateien haben, deren Inhalt du selbst bestimmst. Einzelheiten findest du in dem Abschnitt 3.1.26, *Die Profildateien*.

1.4.4 Absturzverhalten

Wenn *exaEdit* infolge eines Programmfehlers abstürzt, rettet es noch alle workfiles in neue Plattendateien, so weit es möglich ist. Auf diese Weise wird die Gefahr von Datenverlusten stark verringert.

Kapitel 2

Erste Schritte

2.1 Voraussetzungen

Das Kapitel ist so aufgebaut, dass du ohne fremde Hilfe zurechtkommen solltest. Wenn du trotzdem steckenbleibst, muss es nicht unbedingt an dir liegen, es kann durchaus auch der Text mangelhaft sein. Ich beantworte gerne alle Fragen und bin besonders dankbar für Fehlerhinweise und Verbesserungsvorschläge:

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

Nach Möglichkeit solltest du das Kapitel an einem Terminal oder Computer durcharbeiten und alle Befehle wie angegeben auch tatsächlich eintippen.

2.1.1 Für Unix-Systeme

Du solltest in der Lage sein, ein geeignetes Terminal samt Tastatur zu bedienen und ein `login` erfolgreich durchzuführen.

Standardmäßig erhältst du meist nach dem `login` ein Fenster in der Größe 24×80 (24 Zeilen zu je 80 Spalten). Der Editor ist zwar in der Lage, auch Fenster in anderen Abmessungen zu nutzen, jedoch ist es sicherlich zweckmäßig, wenn du zunächst das Standardfenster benutzt, weil dies auch in den folgenden Abbildungen Verwendung findet.

2.1.2 Für Windows-Systeme

Du startest die sogenannte Eingabeaufforderung. Dieses Programm ist u.a. so zu finden: Klicke links unten auf „Start“, dann auf „Alle Programme“, dann auf „Zubehör“ und schließlich auf „Eingabeaufforderung“.

Das dann aufgehende Fenster wird normalerweise 24 Zeilen zu je 80 Spalten aufweisen. Für ein vernünftiges Arbeiten wirst du vielleicht ein größeres Fenster verwenden wollen, aber es ist sicher zweckmäßig, wenn du zunächst die Standardgröße verwendest, weil dies auch in den folgenden Abbildungen Verwendung findet.

2.2 Ein wenig Editor–Logik

Die kleinste Informationseinheit, die uns im Zusammenhang mit einem Editor interessiert, ist das Byte oder Zeichen.

Mehrere Zeichen zusammen können eine Einheit bilden, den sogenannten Satz (oder „record“). Eine solche Einheit kann mit einem besonderen Schlusszeichen markiert sein oder sich auf andere Art von den übrigen Bytes im Computersystem abgrenzen; das ist für uns im Augenblick nicht von Belang.

Wichtiger ist es, zu wissen, was eine Datei und ein Editor sind und wie man Dateien mit Hilfe eines Editors ändern oder erstellen kann. Dies wird, auf einem sehr grundlegendem Niveau, hier erklärt.

Eine Datei (oft auch „file“ genannt) ist eine Folge von Sätzen, etwa auf einem Datenträger (Festplatte, CD–ROM, Diskette usw.).

Eine Datei wird meistens so im Fenster gezeigt, dass jeder Satz der Datei eine Zeile einnimmt.

Ein Editor ist ein Programm zum Erstellen oder Ändern von Dateien. Die meisten Editoren, so auch *exaEdit*, funktionieren so, dass sie das Abbild einer Datei im Hauptspeicher (Arbeitsspeicher) des Computers halten.

Willst du zum Beispiel eine Datei verändern, so rufst du einen Editor auf und teilst ihm den Dateinamen mit. Der Editor kopiert dann die Datei vom Datenträger in den Hauptspeicher. Was der Editor im Hauptspeicher hat, zeigt er normalerweise in Ausschnitten im Fenster. Der Editor nimmt dann die von dir gewünschten Änderungen an den Daten vor. Danach werden die gesamten Daten vom Hauptspeicher wieder auf den Datenträger zurückgeschrieben, wodurch erst die Änderung der Datei beendet ist.

Das vom Editor im Hauptspeicher gehaltene Abbild einer Datei nennen wir (für den Editor *exaEdit*) einen

workfile

Willst du eine Datei neu erstellen, beispielsweise alle Daten dieser Datei erst eintippen, so lässt du den Editor mit einem *leeren* workfile beginnen und gibst ihm am Ende den Befehl, aus dem jetzt mit Inhalt gefüllten workfile eine Datei zu machen. Das zuletzt genannte Beispiel wird im übernächsten Abschnitt ausführlich beschrieben.

2.3 Zur Orientierung

Wie oben beschrieben, sollte dein Fenster 24 $\times$ 80 Zeichen groß sein. Es muss sich im Grundzustand deiner Unix-Sitzung bzw. deiner Einagabeaufforderung befinden, also Unixbefehle bzw. Zeilenkommandos als Eingabe annehmen. Dann tippst du bitte

```
exaedit
```

als Befehl ein. Als Ergebnis der Ausführung dieses Befehls erhältst du in etwa folgendes Bild:

Genau besehen, erhältst du natürlich nur das Innere des gestrichelten Kastens im Fenster. Hier in dieser Anleitung ist ein Rahmen mit der Numerierung 1 – 80 oben und 1 – 24 links dazu abgebildet, damit die einzelnen Bestandteile des jeweils gezeigten Bildes angesprochen werden können.

In Zeile 8, die (im echten Editorfenster) optisch hervorgehoben ist, identifiziert sich *exaEdit* selbst mit seinem Namen in Spalte 10 bis 16 und seiner Versionsnummer (2 Ziffern + 1 Buchstabe) in Spalte 18 bis 20.

Dahinter stehen die Wörter `TOP LINE`. Dies bedeutet, dass es sich um die erste Zeile des workfiles handelt. Der workfile ist sonst leer, da du ja dem Editor keine Datei genannt hast, die er editieren soll. Ein workfile besteht für *exaEdit* immer aus der top line und den eigentlichen Daten. Die top line gibt es nur im workfile, sie wird niemals mit den Daten in eine Datei geschrieben.

In Spalte 1 bis 8 der Zeile 8 steht

```
MAIN
```

Dies ist der Name, den *exaEdit* dem workfile gegeben hat. Der Name deutet an, dass es mehrere workfiles geben kann, aber das lernst du erst in einer späteren Lektion.

In Zeile 16 des Bildes ist ein Lineal zu sehen, das dir die horizontale Orientierung in den vom Editor gezeigten Daten erleichtern soll.

In Zeile 18 des Bildes steht

```
exaEdit
```

was von *exaEdit* als Aufforderung zur Eingabe von Befehlen ausgegeben wird, wenn *exaEdit* nichts Besseres zu sagen hat (wird später noch erläutert).

Die letzte Zeile des Bildes ist die Statuszeile, in der verschiedene Zustände und andere Informationen angezeigt werden.

Ab Spalte 5 steht dort noch einmal der Name des workfiles.

In Spalte 73 steht die Anzahl der Datensätze, die der workfile enthält. Da der workfile noch leer ist, steht dort die Zahl `0`.

In Spalte 75 bis 79 ist die Position des Cursors angegeben. Da dieser sich im Augenblick in Zeile 19, Spalte 1 befindet, steht in der Statuszeile die Angabe „`19/ 1`“.

Die Zeilen vor der Linealzeile, in unserem Fall also die Zeilen 1 bis 15, nennen wir die

Datenzone

Die Zeilen zwischen der Linealzeile und der Statuszeile, hier die Zeilen 17 bis 23, nennen wir die

Dialogzone

Nach so viel trockener Information darfst du jetzt etwas tun: Bitte bewege den Cursor mit den Pfeiltasten durch das Fenster und beobachte dabei, wie sich die Anzeige in der Statuszeile ändert. Achte auch darauf, was passiert, wenn der Cursor über den Fensterrand hinausläuft. Präge dir dieses Verhalten ein und nutze es aus, wenn du während des Editierens den Cursor bewegen musst. Da ein nicht kleiner Teil der Arbeit mit dem Editor in der Bewegung des Cursors besteht, ist es für die Effizienz deiner Arbeit nützlich, wenn du dies mit minimalem Aufwand tun kannst.

Bewege jetzt den Cursor wieder an die Ausgangsstelle zurück (so dass also unten rechts „`19/ 1`“ angezeigt wird).

2.4 Erstellen einer Datei

Wie schon im Abschnitt 2.2, *Ein wenig Editor–Logik*, angedeutet, kannst du eine Datei erstellen, indem du den Editor mit einem leeren workfile startest, den du dann mit Daten füllst, die du am Ende in eine Datei schreiben lässt.

Den leeren workfile hast du vor dir. Zum Füllen mit Daten gibt es verschiedene Methoden. In diesem Abschnitt lernst du den Eingabemodus kennen. Als erstes gibst du den Befehl

`input`

ein, indem du (in der Zeile 19) das Wort `input` eintippst und dann die `Enter`–Taste drückst. Das Fenster sollte daraufhin die folgende Gestalt annehmen:

```
          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|Eingabe                                                                         |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN      I                                                  0 19/ 1 |
 +--------------------------------------------------------------------------------+
```

Was hat sich gegenüber dem vorigen Bild geändert?

- Der von dir gegebene Befehl wird am Anfang der Dialogzone, also in Zeile 17, wiederholt.
- In der nächsten Zeile erscheint das Wort

  `Eingabe`

  als Aufforderung zum Eingeben von Daten.
- Der Cursor steht am Anfang der nächsten Zeile (19), bereit zur Eingabe von Daten.
- In der Statuszeile ist in Spalte 15 ein

  `I`

  erschienen um anzuzeigen, dass sich *exaEdit* jetzt im Eingabemodus befindet.
- Das Lineal beginnt jetzt ganz links, weil du die Eingabezeilen ebenfalls ab Spalte 1 schreibst.

Gib jetzt bitte die erste Eingabezeile ein, etwa

`Dies ist die erste Zeile.`

indem du diesen Satz eintippst und dann die `Enter`–Taste drückst. Daraufhin zeigt sich das Fenster so:

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

Geändert hat sich diesmal:

- Deine Eingabe wird (wie schon beim Befehl `INPUT`) in der ersten Zeile der Dialogzone (Zeile 17) wiederholt.
- Die eingegebene Zeile wird in den workfile übernommen, was sich darin zeigt, dass sie in der Zeile 8 der Datenzone erscheint.
- In der Statuszeile ist in Spalte 73 der Zähler der workfile–Sätze auf 1 gesprungen.
- Der Cursor steht in der nächsten freien Zeile der Dialogzone (Zeile 18), bereit zur Eingabe von Daten.

Gib jetzt bitte die zweite Zeile ein, etwa

```
Die 2. Zeile
```

indem du wieder die Zeichen eintippst und die `Enter`–Taste drückst.

Auch diese Eingabe wird in Zeile 17 wiederholt und in den workfile übernommen, was sich in ihrem Erscheinen in der Zeile 8 widerspiegelt. Die erste eingegebene Zeile ist um eine Zeile nach oben gerutscht.

Nach Eingabe der dritten Zeile

```
Zeile drei
```

hast du folgendes Bild

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

Jetzt hast du genug eingegeben und möchtest den Eingabemodus beenden. Dies geschieht dadurch, dass du die `Enter`–Taste drückst, ohne vorher eine andere Zeichentaste gedrückt zu haben. Das Fenster sieht dann so aus:

```
          1         2         3         4         5         6         7         8
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
 1|                                                                                |
 2|                                                                                |
 3|                                                                                |
 4|                                                                                |
 5|MAIN     exaEdit 02B TOP LINE                                                  |
 6|                                                                                |
 7|000100 Dies ist die erste Zeile.                                               |
 8|000200 Die 2. Zeile                                                            |
 9|000300 Zeile drei                                                              |
10|                                                                                |
11|                                                                                |
12|                                                                                |
13|                                                                                |
14|                                                                                |
15|                                                                                |
16|                                                                                |
17|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
18|Zeile drei                                                                      |
19|_                                                                               |
20|                                                                                |
21|                                                                                |
22|                                                                                |
23|                                                                                |
24|    MAIN                                                          3 18/ 1 |
 +--------------------------------------------------------------------------------+
```

Das Zeichen `I` in der Statuszeile ist verschwunden, weil *exaEdit* nicht mehr im Eingabemodus ist.

Die 3 Zeilen im workfile haben jede eine Nummer bekommen.

Als nächstes möchtest du den gefüllten workfile als Datei auf den Datenträger schreiben. Dies veranlasst du durch Eingabe des Befehls `FILE`, in welchem du den gewünschten Dateinamen mit angeben musst. Soll die Datei etwa `exadat1` heißen, so gibst du jetzt den Befehl

```
file exadat1
```

ein (die `Enter`–Taste nicht vergessen!). Dieser Befehl wird ebenfalls in der ersten Zeile der Dialogzone wiederholt. In der nächsten Zeile steht vermutlich:

```
Neue Datei, drücke J oder Y, um sie zu erstellen:
```

Dies bedeutet, dass *exaEdit* festgestellt hat, dass du noch keine Datei namens `exadat1` hast. Gäbe es diese Datei schon, so würde die Meldung

```
Alte Datei, drücke J oder Y, um sie zu ersetzen:
```

heißen. Dieses Verhalten von *exaEdit* soll dich ein wenig davor schützen, dass du versehentlich einen anderen Dateinamen angibst (und wenn du dich nur vertippt hast), als du eigentlich wolltest.

Die Buchstaben `j` oder `y` stehen für „ja“ bzw. „yes“. Wenn du die Taste `j` oder `y` drückst, wird das von *exaEdit* sofort erkannt, ein zusätzliches Drücken der `Enter`–Taste ist nicht notwendig und in gewisser Hinsicht auch schädlich, weil auf diese Weise die Reaktion von *exaEdit* auf die gedrückte Taste nicht sichtbar wird.

Nach soviel Erklärung: drücke bitte die Taste `j` oder `y`. *exaEdit* schreibt dann den Inhalt des workfiles in die Datei mit dem angegebenen Namen (die bei dieser Gelegenheit neu erstellt wird). Zum Zeichen, dass diese Aktion erfolgreich war, erhältst du die Meldung

```
Daten gesichert
```

Jetzt darfst du diese Lektion beenden und *exaEdit* verlassen, indem du einen der folgenden Befehle eingibst:

```
end      oder      quit
```

2.5 Groß–/Kleinschreibung, Abkürzungen

Wie du schon in der vorigen Lektion gesehen hast, kannst du deine Daten ganz normal mit Groß– und Kleinbuchstaben schreiben, die so, wie sie getippt werden, im workfile ankommen.

Bei den eingegebenen Befehlen (z. B. `INPUT`, `FILE`, `END`, `QUIT`) oder bei den Antworten (`j`, `y`), kannst du kleine oder große Buchstaben ganz beliebig verwenden, sie werden von *exaEdit* nicht unterschieden. Das bezieht sich natürlich nur auf die Befehlsnamen und die Operanden, die in *exaEdit* eine feste Bedeutung haben, während die selbstgewählten Operanden wie etwa Dateinamen oder Zeichenketten selbstverständlich groß/klein-sensibel sind.

In dieser Anleitung werden *exaEdit*–Befehle im Text mit Großbuchstaben geschrieben, damit sie sich besser als Befehle hervorheben. In den Beispielen dagegen werden sie meist klein geschrieben. Auf der Tastatur wirst du selbstverständlich nur kleine Buchstaben tippen.

Die allermeisten *exaEdit*–Befehle kannst du abkürzen, um Tipparbeit zu sparen, beispielsweise `INPUT` mit `I`, `FILE` mit `FIL`, `END` mit `E` und `QUIT` mit `Q`. Welche Minimalabkürzung möglich ist, wird später erklärt. In dieser Anleitung werden im Text immer die unabgekürzten Befehle verwendet, damit du sie dir besser einprägen kannst (was „`END`“ macht, ist ziemlich klar, was aber bewirkt „`E`“?). In den Beispielen wird ein neu eingeführter Befehl ebenfalls ausgeschrieben, während er später, wenn er dir bekannt sein müsste, oft abgekürzt wird. Wenn du den Editor intensiver benutzt, wirst du dir von den gängigen Befehlen sicher bald die Minimalabkürzungen einprägen, weil dies viel Arbeitszeit einsparen hilft.

2.6 Tasten für Entfernen und Einfügen

Beim Tippen kommen sehr leicht Fehler vor, die du natürlich möglichst frühzeitig berichtigen möchtest. Das geht sehr einfach, wenn du die `Enter`–Taste, mit der du die *exaEdit*–Eingabe abschließt, noch nicht gedrückt hast.

Hast du beispielsweise

```
Diesz ist die ...
```

getippt, so fährst du mit dem Cursor bis zu dem überzähligen Zeichen zurück und drückst dann die Taste

```
Entf
```

die du hoffentlich auf deiner Tastatur hast. (Sie ist manchmal auch mit `Del` oder anderem beschriftet.)

Eine Alternative dazu ist die `Backspace`–Taste. Sie liegt oberhalb der `Enter`–Taste und ist mit einem nach links weisenden Pfeil beschriftet (bitte verwechsle sie nicht mit der ähnlich aussehenden Taste „Cursor nach links“). Diese Taste bewirkt, dass das links vom Cursor stehende Zeichen gelöscht wird. Alles was rechts davon steht, rückt um eine Stelle nach links. Zum Vergleich: Die `Entf`–Taste löscht das Zeichen an der Cursorstelle, die `Backspace`–Taste löscht das Zeichen vor dem Cursor; in beiden Fällen rutscht alles rechts davon um 1 Stelle nach links.

Hast du andererseits

```
Die ist die ...
```

getippt, also das `s` vergessen, so fährst du mit dem Cursor bis zu der Stelle zurück, an der das Zeichen einzufügen ist (also zu dem Leerzeichen nach „`Die`“), drückst dann die Taste

```
Einfg
```

die du hoffentlich auf deiner Tastatur hast (sie ist manchmal auch mit „`Ins`“ oder anderem beschriftet), und drückst schließlich die Taste mit dem vergessenen Zeichen.

Durch das Drücken der `Einfg`–Taste hast du *exaEdit* in den Einfügemodus gebracht (bitte unterscheide dies von dem Begriff Eingabemodus in Abschnitt 2.4). Dies bedeutet, dass alle weiteren eingegebenen Zeichen an der Stelle des Cursors eingefügt werden. Dass *exaEdit* im Einfügemodus ist, erkennst du an dem Zeichen ∧, das in Spalte 14 der Statuszeile erscheint. Den Einfügemodus verlässt du wieder, indem du die `Einfg`–Taste erneut drückst. Außerdem

wird nach jedem Drücken der Enter–Taste ein etwa vorhandener Einfügemodus ausgeschaltet. Du kannst allerdings *exaEdit* dazu veranlassen, den Einfügemodus dauerhaft beizubehalten (siehe Kapitel 3).

Wie du Tippfehler in den eingegebenen Zeilen beseitigen kannst, die du erst erkennst, wenn du die Zeilen mittels Enter–Taste in den workfile gegeben hast, lernst du später in Abschnitt 2.8 und 2.16.

2.7 Editieren einer vorhandenen Datei

Als nächstes möchtest du die zuvor erstellte Datei exadat1 erneut editieren. Du tippst also

```
exaedit exadat1
```

ein und erhältst folgendes Bild

```
          1         2         3         4         5         6         7         8
  12345678901234567890123456789012345678901234567890123456789012345678901234567890
 +--------------------------------------------------------------------------------+
1|                                                                                |
2|                                                                                |
3|                                                                                |
4|                                                                                |
5|                                                                                |
6|                                                                                |
7|                                                                                |
8|MAIN     exaEdit 02B TOP LINE exadat1                                           |
9|000100 Dies ist die erste Zeile.                                               |
10|000200 Die 2.Zeile                                                            |
11|000300 Zeile drei                                                             |
12|                                                                               |
13|                                                                               |
14|                                                                               |
15|                                                                               |
16|       ....;....1....;....2....;....3....;....4....;....5....;....6....;....7...|
17|                                                                               |
18|exaEdit                                                                        |
19|_                                                                              |
20|                                                                               |
21|                                                                               |
22|                                                                               |
23|                                                                               |
24|    MAIN          exadat1                                        3 19/ 1 |
 +--------------------------------------------------------------------------------+
```

Gegenüber dem letzten Bild der Lektion *Erstellen einer Datei* hat sich folgendes geändert: Sowohl in der top line, als auch in der Statuszeile ist jetzt der Dateiname

```
exadat1
```

eingetragen. Wenn du jetzt Änderungen am workfile vornimmst, brauchst du am Ende nur den Befehl

```
file
```

zu geben, *exaEdit* schreibt dann den workfile in die Datei exadat1.

Du kannst aber auch den workfile in eine andere Datei schreiben, indem du ihren Namen im FILE–Befehl angibst:

```
file datneu
```

Gib diesen Befehl bitte ein. Du erhältst dann wieder die Aufforderung

```
Neue Datei, drücke J oder Y, um sie zu erstellen:
```

Im Gegensatz zur Lektion *Erstellen einer Datei* sollst du jetzt bitte anstelle von `J` oder `Y` eine beliebige andere (Zeichen–)Taste drücken. Weil dies auch versehentlich geschehen kann (du wolltest `J` drücken, hast die Taste aber verfehlt), wirst du jetzt akustisch und optisch darauf hingewiesen, dass der `FILE`–Befehl nicht abgeschlossen wurde:

```
ACHTUNG: Daten nicht gesichert!
```

Damit du für spätere Lektionen eine zweite Datei hast, gibst du bitte den `FILE`–Befehl erneut und beantwortest ihn mit `J` oder `Y`. Beende dann bitte die *exaEdit*–Sitzung mit dem Befehl `QUIT` oder `END`.

2.8 Direktes Ändern von Daten

In dieser Lektion lernst du, wie du vorhandene Daten ändern kannst.

Gib bitte wie in der vorigen Lektion den Befehl

```
exaedit exadat1
```

ein. `exadat1` ist die Datei, die du in der Lektion *Erstellen einer Datei* erzeugt hast. Das Fenster sollte wieder so wie in der letzten Lektion aussehen.

Die von dir beabsichtigte Änderung sei etwa, das Wort „`erste`“ durch „`1.`“ und „`drei`“ durch „`3`“ zu ersetzen. Zu diesem Zweck musst du nur den Cursor mit den Pfeiltasten an die richtige Stelle der Daten bewegen und dort einfach drüberschreiben. Bei dem Wort „`erste`“ hast du nach dem Eintippen von „`1.`“ noch ein paar überzählige Zeichen: diese löschst du durch Betätigen der Taste `Entf` (vgl. Abschnitt 2.6, *Tasten für Entfernen und Einfügen*). Müsstest du Zeichen einfügen, so würdest du die Taste `Einfg` verwenden.

Damit die Änderungen auch im workfile ankommen, musst du jetzt die `Enter`–Taste drücken. Wenn dir dagegen plötzlich einfällt, dass du die Änderungen doch nicht haben willst, kannst du anstelle der `Enter`–Taste die Taste `Pos 1` (oder `Home`) drücken: damit wird der Zustand wiederhergestellt, den du vor Beginn deiner Änderungen hattest.

Nach dem Drücken der `Enter`–Taste hast du keine Möglichkeit mehr, die Änderungen mit einem Schlag rückgängig zu machen. Zwar kannst du auf das Sichern des workfiles verzichten, aber damit sind auch alle anderen Änderungen weg, die du in der *exaEdit*–Sitzung vielleicht zuvor schon gemacht hast.

2.9 Beenden von *exaEdit*

Diese Lektion schließt sich unmittelbar an die vorige an, in der du an deiner Datei Änderungen vorgenommen hast, aber am Ende nicht den Befehl `FILE` gegeben hast.

Bitte versuche, die *exaEdit*–Sitzung mit dem Befehl

`quit` oder `end`

zu beenden. Da *exaEdit* weiß, dass der geänderte workfile noch nicht auf den Datenträger zurückgeschrieben wurde, sollst du vor einem unüberlegten Schritt bewahrt werden. *exaEdit* antwortet nämlich mit dem Hinweis und der Aufforderung

```
Änderungen nicht gesichert
Drücke J oder Y, um aufzuhören:
```

Drückst du jetzt irgendeine andere (Zeichen–)Taste als die angegebene (bitte mach es), so bricht *exaEdit* die Bearbeitung des `QUIT`– oder `END`–Befehls ab, schreibt `exaEdit` in die Dialogzone und wartet auf deine nächsten Befehle.

Hättest du dagegen eine der Tasten `J` oder `Y` gedrückt, so hätte *exaEdit* geendet, ohne deine Änderungen zu sichern.

2.10 Aktuelle Zeile, Positionieren

Wie du vielleicht in der vorigen Lektion bemerkt hast, haben die drei Zeilen deiner Datei nach der Textänderung, die du gemacht hast, ihren Platz auf dem Fenster geändert. Wenn vorher die top line („`MAIN exaEdit...`“) in der optisch hervorgehobenen (etwa der Text weiß auf schwarzem Hintergrund) Mittelzeile der Datenzone stand, ist es jetzt vermutlich die dritte Textzeile, die diesen hervorgehobenen Platz einnimmt.

Die hervorgehobene Mittelzeile der Datenzone heißt

aktuelle Zeile

(in englischen Texten als current line bezeichnet). Sie hat in *exaEdit* zwei besondere Eigenschaften:

- sie ist Ausgangspunkt von Befehlen, die eine Zeilenangabe benötigen,
- sie wird automatisch nachgestellt.

Wegen ihrer ersten Eigenschaft muss es möglich sein, dass du sie durch Befehle verschieben kannst. Üben wir zunächst dieses Positionieren (bitte alles ausprobieren).

Mit dem Befehl

```
top
```

wird die top line zur aktuellen Zeile. Mit dem Befehl

```
bottom
```

wird die letzte Zeile zur aktuellen. Da beides häufig benutzte Befehle sind, haben sie schöne Minimalabkürzungen `T` und `B`.

Mit den Befehlen

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

worin du für „`n`“ eine Zahl einzusetzen hast, wird die aktuelle Zeile um `n` Zeilen weiter nach unten, also zum Ende der Datei hin, versetzt.

Mit den Befehlen

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

wird die aktuelle Zeile nach oben, also zum Anfang der Datei hin, versetzt.

Gibst du mehr Zeilen an, als verschoben werden können, so bleibt die aktuelle Zeile unverändert, und *exaEdit* gibt die Meldung `Datenende` bzw. `Datenanfang` aus.

Willst du die aktuelle Zeile nur um 1 Zeile verschieben, so kannst du die Zeilenangabe `n` weglassen, also zum Beispiel nur `+` eingeben.

Häufig kommt es vor, dass du die aktuelle Zeile um ganze oder halbe Fensterseiten verschieben möchtest. Es gibt dafür spezielle Tasten

`F7`	eine Seite rückwärts
`F8`	eine Seite vorwärts
`F10`	eine halbe Seite rückwärts
`F11`	eine halbe Seite vorwärts

Anstelle von F7 und F8 kannst du auch die Tasten Bild↑ und Bild↓ verwenden, wenn sie vorhanden sind. Wenn du die genannten Tasten jetzt ausprobierst, kannst du bei deiner kleinen Datei den Effekt nicht genau erkennen, du kannst aber sehen, wie diese besonderen Tasten in die *exaEdit*–Befehle

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

umgesetzt werden (wenn dein Fenster 24 Zeilen hat).

Das Blättern um 1 Seite vorwärts geht übrigens so, dass die letzte Zeile des alten Fensters zur ersten Zeile des neuen Fensters wird, so dass dir beim Blättern ein gewisser Zusammenhang erhalten bleibt.

Jetzt bleibt noch festzustellen, wie das automatische Positionieren der aktuellen Zeile erfolgt.

Wenn du direkt in der Datenzone änderst (siehe Abschnitt 2.8), wird nach dem Drücken der Enter–Taste diejenige geänderte Zeile zur aktuellen, die auf dem Fenster am weitesten unten stand. Der Grund für dieses Verhalten ist, dass *exaEdit* vermutet, dass du die unterste Änderung zuletzt gemacht hast, und dass die letzte Änderung in die Mitte des Fensters (in das Zentrum deiner Aufmerksamkeit) gerückt werden soll.

Auch in der Folge der Ausführung mancher Befehle verschiebt sich die aktuelle Zeile. Auch dabei gibt es feste Regeln, die aber von Befehl zu Befehl verschieden sein können und deshalb bei der Beschreibung der Befehle im nächsten Kapitel genau angegeben werden.

2.11 Einfügen von Zeilen

Hierfür gibt es verschiedene Möglichkeiten. Die erste ist der Eingabemodus, den du in Kapitel 2.4 kennengelernt hast. Wenn du beispielsweise zwischen der 1. und der 2. Zeile der Daten Zeilen einfügen möchtest, so musst du zunächst den workfile so positionieren (siehe Kapitel 2.10, *Aktuelle Zeile, Positionieren*), dass die 1. Zeile die aktuelle ist. Dann versetzt du *exaEdit* mit dem Befehl

```
input
```

in den Eingabemodus. Alles weitere kennst du schon von Kapitel 2.4.

Eine andere Möglichkeit zum Einfügen von Zeilen besteht darin, dass du den

Nummerbefehl

verwendest. Zu diesem Zweck tippst du eine Zeile ein (nicht im Eingabemodus, sondern im *exaEdit*–Normalzustand), die außer den gewünschten Daten vorne dran noch eine Zeilennummer hat, also insgesamt etwas, das wie eine Zeile aus der Datenzone aussieht:

```
222 Text
```

Daraus erzeugt dann *exaEdit* eine entsprechende Zeile in der Datenzone. Durch die Wahl der Zeilennummer legst du dabei fest, wo die Zeile hinkommen soll. Die Zeilennummer kannst du mit oder ohne führende Nullen schreiben.

Es ist übrigens gleichgültig, ob du nach der Nummer ein Leerzeichen setzt oder nicht:

`222Text` und `222 Text`

sind dasselbe, weitere Leerzeichen werden dagegen beachtet.

Wenn die eingetippte Zeilennummer mit einer schon vorhandenen Nummer übereinstimmt, so wird keine neue Zeile eingefügt, sondern die alte Zeile mit der neuen überschrieben.

Weitere Methoden zum Einfügen von (speziellen) Zeilen werden in späteren Abschnitten beschrieben.

2.12 Löschen von Zeilen

Auch hierfür gibt es verschiedene Möglichkeiten. Der Befehl

```
delete n
```

löscht `n` Zeilen, beginnend mit der aktuellen Zeile. Lässt du die Angabe `n` weg, so wird nur 1 Zeile (die aktuelle) gelöscht. Der Befehl `DELETE` hat die Minimalabkürzung `DE`.

Eine andere Möglichkeit zum Löschen von Zeilen ist der Befehl

```
dl  von  bis
```

in welchem du für `von` und `bis` die erste und letzte Nummer der zu löschenden Zeilen angeben musst. `DL` steht für delete lines.

Eine dritte Möglichkeit zum Löschen von Zeilen besteht darin, im Nummernfeld der zu löschenden Zeile ein „`d`“ einzutippen:

```
0003d0
```

Nach dem Drücken der `Enter`–Taste werden alle so markierten Zeilen gelöscht. Das Zeichen `d` kannst du an beliebiger Stelle des Nummernfeldes eingeben.

Du kannst auf die eben beschriebene Art auch mehrere aufeinanderfolgende Zeilen löschen, indem du die gewünschte Anzahl mit eintippst:

```
0003d2
```

löscht die markierte und die folgende Zeile. Tippst du das Zeichen `d` weiter vorne in der Nummer, so musst du nach der Anzahlangabe ein Leerzeichen tippen:

```
00d300
```

löscht nur die markierte Zeile, während das folgende Beispiel 3 Zeilen löscht:

```
00d3 0
```

Weitere Möglichkeiten zum Löschen von Zeilen sind in späteren Abschnitten beschrieben.

2.13 Kopieren von Zeilen

Oft kommt es vor, dass du bereits vorhandene Zeilen noch einmal benötigst. Diese brauchst du natürlich nicht noch einmal einzugeben. Der Befehl

```
copy  von  bis
```

kopiert die angegebenen Zeilen (`von` und `bis` sind Zeilennummern) hinter die aktuelle Zeile. Willst du nur 1 Zeile kopieren, so brauchst du nur diese Nummer anzugeben. Lässt du beide Nummerangaben weg, so kopiert (verdoppelt) *exaEdit* die aktuelle Zeile.

Außer realen Zeilennummern kennt *exaEdit* auch symbolische Zeilennummern und zwar

- `*` für die aktuelle Zeile,
- `p` (= previous) für die Zeile vor der aktuellen,
- `n` (= next) für die Zeile nach der aktuellen,
- `f` (= first) für die erste (Daten–)Zeile des workfiles,
- `l` (= last) für die letzte (Daten–)Zeile des workfiles,
- `b` (= bottom) für die letzte (Daten–)Zeile des workfiles,
- `t` (= top) für die erste Zeile des workfiles, wobei *exaEdit* je nach Zusammenhang die top line oder die 1. Datenzeile meint,
- `s` für die mit dem Befehl `SET` markierte Zeile.

Beispiele für den COPY–Befehl:

`copy 100 200`	kopiert die Zeilen von 100 bis 200 hinter die aktuelle Zeile.
`co`	verdoppelt die aktuelle Zeile.
`co *`	verdoppelt die aktuelle Zeile.
`co p`	verdoppelt die Zeile vor der aktuellen.
`co p *`	kopiert die Zeile vor der aktuellen und die aktuelle Zeile hinter die aktuelle, macht also aus den beiden Zeilen `a b` die 4 Zeilen `a b a b`, wenn `b` die aktuelle Zeile war.
`co f l`	kopiert die gesamten Daten in der Weise wie im vorigen Beispiel.

Eine andere Möglichkeit zum Kopieren von Zeilen findest du im Abschnitt 3.3, *Präfixbefehle*.

2.14 Verschieben von Zeilen

Für das Verschieben von Zeilen gibt es den Befehl

```
move  von  bis
```

der genauso wie COPY funktioniert mit dem einzigen Unterschied, dass die angegebenen Zeilen nicht kopiert, sondern verschoben werden. Beispiele

`move 100 200`	versetzt die Zeilen von 100 bis 200 hinter die aktuelle Zeile,
`mo p`	vertauscht die aktuelle Zeile mit der vorigen.

Eine andere Möglichkeit zum Verschieben von Zeilen wird es in späteren Versionen von *exaEdit* geben.

2.15 Aufsuchen von Daten

Wenn du eine bestimmte Stelle in den Daten suchst, kannst du natürlich blättern. Du kannst aber auch eine solche Stelle direkt aufsuchen, wenn du weißt, was dort genau steht. Der Befehl

```
locate /Zeichenkette/
```

sucht, von der aktuellen Zeile ausgehend, nach der angegebenen Zeichenkette. Wird sie gefunden, so wird ihre Zeile zur aktuellen Zeile gemacht. Beispiel

```
locate /ist/
```

Die zu suchende Zeichenkette musst du in ein Paar begrenzender Zeichen einschließen (wobei du das Schlusszeichen manchmal auch weglassen kannst). Das Zeichen zur Kettenbegrenzung (oben ein „/“) kann jedes Zeichen außer einer Ziffer, dem Zeichen „&“ oder dem Befehlsseparator sein. So sucht zum Beispiel

```
locate ist
```

nach der Zeichenkette „`st`“ („`i`“ ist der Anfangsbegrenzer, der Schlussbegrenzer fehlt). Am praktischsten ist es, wenn du dir angewöhnst, als Zeichenkettenbegrenzer dasjenige Sonderzeichen zu nehmen, das sich rechts unten auf der Tastatur (im Zeichenbereich) befindet. Nur wenn dieses Zeichen in der Zeichenkette selbst vorkommt, musst du einen anderen Begrenzer verwenden.

Wenn die Suche nach der Zeichenkette am Ende des workfiles angelangt ist, wird sie vom Anfang der Daten aus fortgesetzt. Diesen Vorgang macht *exaEdit* dir bewusst, indem beim Durchgang durch das Dateiende

```
Suche ab Anfang (wrap)
```

in das Fenster geschrieben wird. (Du kannst auch verlangen, dass die Suche am workfile–Ende stoppt, siehe Befehl WRAP im nächsten Kapitel.)

Musst du eine Zeichenkette mehrfach suchen, so genügt bei den folgenden Malen die Eingabe von

```
locate
```

alleine. Da LOCATE die Minimalabkürzung L hat, kommst du sogar mit der folgenden Eingabe aus:

```
l
```

Außer von oben nach unten kann *exaEdit* auch umgekehrt von unten nach oben suchen. Dies geht mit dem Befehl

```
rlocate /Zeichenkette/
```

(rlocate = „reverse locate“). Geht hier die Suche über den Anfang des workfiles hinweg und beim workfile–Ende weiter, so schreibt *exaEdit*

```
Suche ab Ende (wrap)
```

in das Fenster.

Das Gedächtnis für die Suche ist für LOCATE und RLOCATE dasselbe.

2.16 Ändern von Daten

Das direkte Ändern von Daten hast du in Kapitel 2.8 kennengelernt. Eine weitere Möglichkeit war die Verwendung eines Nummerbefehls mit einer bereits vorhandenen Nummer, mit dem du allerdings nur die ganze Zeile ändern kannst.

Eine weitere Möglichkeit für Änderungen bietet der Befehl CHANGE, Minimalabkürzung C:

```
change /alt/neu/
```

Er ändert in der aktuellen Zeile die Zeichenkette

```
alt
```

in die Zeichenkette

```
neu
```

um. Das Zeichen „/“ stellt wieder den Zeichenkettenbegrenzer dar, den du im vorigen Abschnitt kennengelernt hast. Wie dort kannst du auch hier andere Zeichen dafür verwenden.

Oft willst du in aufeinanderfolgenden Zeilen die gleiche Änderung machen. Für diesen Zweck kannst du eine Zeilenanzahl hinzufügen. Beispielsweise rückt

```
c // /5
```

beginnend mit der aktuellen Zeile, in 5 Zeilen die vorhandenen Daten um 1 Stelle nach rechts. Die „alte“ Zeichenkette ist leer (2 Begrenzer folgen aufeinander), diese leere Zeichenkette kommt in jeder Zeile am Anfang vor. Die „neue“ Zeichenkette besteht aus einem einzelnen Leerzeichen.

Weitere Möglichkeiten für den CHANGE–Befehl findest du im Kapitel 3.

2.17 Hilfe

exaEdit enthält auch Hilfetexte in recht komprimierter Form. Sie sind vor allem dann nützlich, wenn du einen Befehl in seiner (gewünschten) Funktion schon in etwa kennst, aber die genaue Syntax dir nicht gegenwärtig ist. Der Befehl

```
help
```

Minimalabkürzung H, liefert eine Liste aller *exaEdit*–Befehle. Die groß geschriebenen Anfänge der Befehlsnamen stellen die Minimalabkürzungen dar, nach diesen Abkürzungen ist die Liste sortiert.

Mit dem Befehl

```
help befehl
```

erhältst du Informationen über den gewünschten Befehl. Beispielsweise ergibt `HELP CHANGE`:

```
help change
Change [col1 [col2]] /kette1/kette2/ [n] [A] [D] [H] [I]          change data
 kette1 wird in n Zeilen (Standard 1) gesucht und durch kette2 ersetzt. kette1
 wird gelöscht, wenn du kette2 nicht angibst. A ändert alle Vorkommnisse in
 der Zeile, D zeigt alle geänderten Zeilen. H sucht hexadezimal, I groß/klein-
 insensibel. Suche ist durch ZONE oder die (vorgehenden) Spalten beschränkt.
```

Der Aufbau der Information ist der folgende:

In der 1. Zeile steht links die Syntax des Befehls, rechts eine ganz kurze (englische) Charakterisierung der Funktion. Darunter kommt eine Erläuterung der Funktion und der Befehlsparameter.

Bei der Syntax sind die in eckige Klammern [ ] gesetzten Angaben solche, die auch weggelassen werden können.

Bitte sieh dir einige Hilfetexte zu Befehlen an, die du schon kennst, damit du mit der Darstellung vertrauter wirst.

Die vollständige *exaEdit*–Benutzeranleitung ist meistens auch mit dem Befehl `MANUAL` am Bildschirm zu lesen (über einen WWW–Browser).

2.18 Ein wichtiger Befehl

Bitte entschuldige, wenn ich dich an der Nase herumführe: Es gibt keine wichtigen oder unwichtigen Befehle. Ich habe versucht, in diesem Kapitel, *Erste Schritte*, alles vorzuführen, was du brauchst, wenn *exaEdit* ein nützliches Werkzeug für dich werden soll.

Für viele spezielle Anwendungen reicht das Gelernte aber nicht aus, vor allem, wenn du effektiv arbeiten willst. Es wird dir also nichts übrigbleiben, als das nächste Kapitel, *Der Editor und seine Befehle*, wenigstens einmal konzentriert und in aller Ruhe durchzulesen, damit du bei Bedarf auf das Gelesene zurückgreifen kannst.

Kapitel 3

Der Editor und seine Befehle

Dieses Kapitel enthält die vollständige, ausführliche Beschreibung aller Eigenschaften des Editors. Bei (vermeintlichen) Differenzen zu Ausführungen in anderen Kapiteln dieser Anleitung oder zu *exaEdit*–Informationen an anderer Stelle ist immer dieses Kapitel maßgebend. Beachte, dass im ersten Unterkapitel mit den verschiedenen Funktionen bei weitem nicht alle Befehle erwähnt werden, die *exaEdit* kennt. Das Studium des Unterkapitels mit den einzelnen Befehlen ist daher sehr zu empfehlen.

3.1 Die Funktionen

Hier werden die einzelnen Funktionen beschrieben, die *exaEdit* aus deiner Sicht hat. Diese Funktionen können manchmal auf verschiedene Weise realisiert werden und zu ihrer Realisierung werden manchmal verschiedene *exaEdit*–Befehle gebraucht.

3.1.1 Aufnahme einer *exaEdit*–Sitzung

Die *exaEdit*–Sitzung wird im allgemeinen begonnen durch Eingabe des Zeilenbefehls

```
exaedit [filename]
```

Die Angabe `[filename]` in eckigen Klammern besagt, dass du einen Dateinamen angeben kannst oder auch nicht.

Falls du *exaEdit* mit mehr als einem Parameter aufrufst, werden dennoch nur der erste beachtet und alle anderen stillschweigend ignoriert. Was der Parameter im einzelnen bedeutet, ist in Abschnitt 3.1.3, *Laden einer Datei*, beschrieben.

Als erstes liest *exaEdit* etwa vorhandene Profildateien. Dies ist in Abschnitt 3.1.26, *Die Profildateien*, genauer beschrieben.

Als zweites muss sich *exaEdit* über die Art des benutzten Fensters informieren und die Wege vom und zum Fenster aktivieren. *exaEdit* ist ein Ganzfenster–Editor, d.h. *exaEdit* produziert die Daten für das Beschreiben des kompletten Fensters und kann im Gegenzug Änderungen von jeder Fensterstelle entgegennehmen.

Um die Ganzfensterfunktion zu erfüllen, bedient sich *exaEdit* eines Betriebssystemzusatzes namens

Curses

Darunter ist eine Sammlung von Programmfunktionen zu verstehen, mit denen die Ganzfensternutzung begonnen, durchgeführt und beendet werden kann. Da du normalerweise die Ganzfensternutzung haben möchtest, wird an dieser Stelle von *exaEdit* untersucht, ob sie möglich ist und zutreffendenfalls die Initialisierung von Curses durchgeführt.

Der Rest dieses Abschnittes trifft nur auf Unix–Betriebssysteme zu.

Erste Voraussetzung ist das Vorhandensein der Umgebungsvariablen

```
TERM
```

Umgebungsvariablen werden vom Betriebssystem bereitgehalten. Du kannst sie dir mit dem Unix–Befehl

```
set
```

auflisten lassen. Falls TERM wider Erwarten nicht vorhanden ist, schreibt *exaEdit* die beiden Meldungen

```
TERM nicht definiert
exaEdit im Zeilenmodus
```

in das Fenster. Die erste Meldung ist selbsterklärend, die zweite bedeutet, dass *exaEdit* nicht im Ganzfenster– oder Fenstermodus arbeiten kann (eine Initialisierung von Curses also nicht stattfinden kann), sondern nur zeilenweises Arbeiten möglich ist. Näheres zum Zeilenmodus findest du in Abschnitt 3.1.20, *Der Zeilenmodus*.

Ist die Umgebungsvariable TERM vorhanden, so überprüft *exaEdit*, ob die Variable einen Wert hat, der für den Fenstermodus geeignet ist. Derzeit werden

```
TERM=NETWORK        und
TERM=IBM3278-x
```

(mit beliebigem x) als ungeeignet erkannt. In einem solchen Fall schreibt *exaEdit* die beiden Meldungen

```
Terminaltyp ist ...
exaEdit im Zeilenmodus
```

in das Fenster und arbeitet im Zeilenmodus weiter (siehe Abschnitt 3.1.20, *Der Zeilenmodus*).

Leider ist *exaEdit* derzeit nicht in der Lage, alle TERM–Werte, für die kein Fenstermodus möglich ist, zu erkennen. Es kann also vorkommen, dass *exaEdit* die Initialisierung von Curses versucht, Curses aber mit einer Fehlermeldung *exaEdit* abbricht. Solche Meldungen sind

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

Was kannst du tun, wenn — entgegen deinen Wünschen — *exaEdit* in den Zeilenmodus schaltet oder gar eine der Sorry–Meldungen erscheint? Du kannst mit dem Unix–Befehl

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

in welchem du die Punkte durch den gewünschten Terminaltyp ersetzt, die Umgebungsvariable TERM setzen. Wenn du nach einer der Sorry–Meldungen wenigstens im Zeilenmodus arbeiten willst, kannst du zum Beispiel `export TERM=NETWORK` eingeben.

3.1.2 Workfiles

exaEdit arbeitet — wie viele andere Editoren auch — gewöhnlich so, dass eine Datei vom Datenträger in den Hauptspeicher geladen, dort von dir geändert, und zum Schluss wieder auf den Datenträger geschrieben wird. Das Abbild der Datei im Hauptspeicher heißt

workfile

exaEdit kann mehrere workfiles haben, ihre Anzahl ist nicht begrenzt (ausgenommen natürlich durch die Größe des Hauptspeichers).

Jeder workfile hat einen eigenen Namen, der aus 1 bis 8 Buchstaben oder Ziffern besteht und mit einem Buchstaben beginnt.

Der erste (oder einzige) workfile einer *exaEdit*–Sitzung trägt den Namen

```
MAIN
```

Workfile–Namen kannst du mit großen oder kleinen Buchstaben schreiben, die nicht unterschieden werden. *exaEdit* schreibt workfile–Namen immer mit großen Buchstaben.

Der workfile–Name steht in der Statuszeile (das ist die letzte Fensterzeile) in Spalte 5 bis 11 und in der top line des workfiles (zur top line siehe nächsten Absatz).

Der workfile enthält neben den eigentlichen Daten noch die sogenannte

top line

Darunter kannst du dir einen fiktiven 0. Satz vorstellen, der nur dazu da ist, den Anfang des workfiles zu bezeichnen. Die top line wird im Fenster mit angezeigt. Sie ist jedoch nicht in der Datei auf dem Datenträger vorhanden, da sie beim Laden der Daten erst erzeugt wird und beim Speichern der Daten auf den Datenträger nicht mitgeschrieben wird.

Manche Editorbefehle behandeln die top line wie eine normale Datenzeile, die allerdings nicht änderbar ist.

Das Anlegen und Löschen von workfiles und das Umschalten von einem workfile in einen anderen erfolgt mit dem Befehl

```
workfile
```

Es gibt immer einen aktuellen workfile, der als einziger im Fenster zu sehen ist und auf den alle eingegebenen Befehle wirken. Allerdings ist es möglich, Daten von einem workfile in einen anderen zu kopieren.

Möchtest du zwei (oder mehr) workfiles gleichzeitig sehen, so kannst du dies nicht mit einer einzigen *exaEdit*–Sitzung machen, sondern brauchst dazu zwei (oder mehr) Fenster oder Bildschirme.

3.1.3 Laden einer Datei

3.1.3.1 Laden im Normalfall

Dieser Abschnitt beschreibt den Normalfall, wie du eine Datei in einen *exaEdit*–workfile bringst. Besonderheiten, die du vermutlich seltener benötigst, stehen in den folgenden Abschnitten.

Es gibt zwei verschiedene Arten, eine Datei zu laden (= vom Datenträger zu lesen und in den workfile zu bringen). Die erste Methode besteht darin, beim Aufruf von *exaEdit* einen Dateinamen mitanzugeben:

```
exaEdit datei
```

Enthält der Dateiname Sonderzeichen, so musst du entsprechende Vorkehrungen treffen, etwa den Namen in Anführungszeichen (") setzen. Dies hilft nicht in allen Fällen. Wie es wirklich zu machen ist, wird dir in Unix von der Benutzeroberfläche (shell) vorgeschrieben, die du verwendest, oder ist bei Windows nachzulesen.

War das Laden erfolgreich, so ist der Dateiname dem workfile zugeordnet, erscheint also in der top line und in der Statuszeile. Dies bedeutet, dass beim Befehl `FILE` ohne Angabe eines Dateinamens diese Datei beschrieben wird.

Die andere Möglichkeit, eine Datei zu laden, besteht darin, in einer bereits bestehenden *exaEdit*–Sitzung den Befehl `LOAD` mit einem Dateinamen zu geben, etwa

```
load datei
```

Der Dateiname, sei es der dem workfile zugeordnete oder der von dir im Befehl `LOAD` angegebene, kann ein absoluter oder ein relativer Dateiname sein.

Hier müssen wir jetzt zwischen Unix– und Windows–Systemen unterscheiden.

In Unix–Systemen:

Der absolute Dateiname beginnt mit einem Schrägstrich, der relative nicht. Gibst du einen relativen Dateinamen an, so wird er durch Voransetzen deines aktuellen Arbeitsverzeichnisses zu einem absoluten Dateinamen ergänzt. Wenn du wissen möchtest, wie das aktuelle Arbeitsverzeichnis heißt, kannst du dafür den *exaEdit*–Befehl

```
call pwd      oder      _pwd
```

geben, der dir das Ergebnis des Unixbefehls pwd (pathname of the working directory) zeigt. Ein Beispiel: Ist

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

dein aktuelles Arbeitsverzeichnis, so lesen die Befehle

```
load etc/eins      bzw.      load /etc/eins
```

die Dateien

```
/u/fmath/ppreus/exaEdit/etc/eins      bzw.      /etc/eins
```

Du kannst beim Dateinamen mit der dafür üblichen Schreibweise auch einen Wechsel ins übergeordnete Verzeichnis verlangen, also im obigem Beispiel mittels

```
load ../abc/zwei
```

die Datei

```
/u/fmath/ppreus/abc/zwei
```

lesen, obwohl dein Arbeitsverzeichnis dieses ist:

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

Schließlich kannst du beim Dateinamen auch die Schreibweise mit der Tilde (~) verwenden. In

```
~/...
```

steht die Tilde für dein Home–Verzeichnis, in

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

steht `~uid` für das Home–Verzeichnis von `uid`.

In Windows–Systemen:

Der absolute Dateiname beginnt mit einem Rückstrich („\“) oder mit einem Laufwerksbuchstaben, der relative nicht. Gibst du einen relativen Dateinamen an, so wird er durch Voransetzen deines aktuellen Arbeitsverzeichnisses zu einem absoluten Dateinamen ergänzt. Wenn du wissen möchtest, wie das aktuelle Arbeitsverzeichnis heißt, kannst du dafür den *exaEdit*–Befehl

```
call cd      oder      _cd
```

geben, der dir das Ergebnis des DOS–Befehls `cd` zeigt. Ein Beispiel: Ist

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

dein aktuelles Arbeitsverzeichnis, so lesen die Befehle

```
load winnt\win.ini      bzw.      load \winnt\win.ini
```

die Dateien

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

Du kannst beim Dateinamen mit der dafür üblichen Schreibweise auch einen Wechsel ins übergeordnete Verzeichnis verlangen, also in obigem Beispiel mittels

```
load ..\abc\zwei
```

die Datei

```
d:\pe\abc\zwei
```

lesen, obwohl dein Arbeitsverzeichnis dieses ist:

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

Ab hier wieder für **Unix– und Windows–Systeme** gemeinsam.

Enthält der Dateiname Sonderzeichen, so musst du folgende Regel anwenden: Schließe den Dateinamen in einfache Apostrophe ein (’); ersetze einen Apostroph, der Bestandteil des Dateinamens sein soll, durch zwei Apostrophe.

Beim Laden mit dem LOAD–Befehl ist zu beachten, dass du dies auch bei einem nicht leeren workfile tun kannst. In diesem Fall wird die Datei hinter den Satz der aktuellen Zeile geladen. Um auf diese Weise etwa zwei Dateien zusammenzufügen, lädst du erst die eine (mittels *exaEdit* oder mittels LOAD), gibst dann den Befehl BOTTOM und lädst dann mittels LOAD die zweite Datei.

Wenn das Laden mit dem LOAD–Befehl erfolgreich war, wird der Name der geladenen Datei dem workfile zugeordnet (wird also sozusagen zum Dateinamen des workfiles), wenn es zum workfile noch keinen Dateinamen gab. Gibt es dagegen bereits einen Dateinamen zum workfile, so wird er durch das Ausführen eines LOAD–Befehls nicht mehr geändert.

Möchtest du das Ergebnis deines Editierens in eine andere Datei bringen, so kannst du diese dann beim FILE–Befehl unterbringen, siehe Abschnitt 3.1.4, *Speichern einer Datei*.

Nun zu den Fehlermöglichkeiten beim Laden einer Datei.

Findet *exaEdit* die angegebene Datei nicht, weil du vielleicht einen Tippfehler gemacht hast oder die Datei sich in einem anderen Verzeichnis befindet, so erhältst du die Meldung:

```
Datei nicht gefunden
```

Trotzdem wird der angegebene Dateiname dem workfile zugeordnet, wenn es zum workfile noch keinen Dateinamen gab und wenn du das Laden der Datei durch die Namensangabe im *exaEdit*–Aufruf verlangt hast. Dieses Verhalten ermöglicht es dir, bei der Erstellung einer noch nicht vorhandenen Datei sogleich deren Namen festzulegen, den du dann im FILE–Befehl nicht mehr zu wiederholen brauchst.

„Gewöhnliche“ Fehler, die du beim Laden einer Datei machen kannst, sind die folgenden:

```
Parameter fehlt
```

Diese Meldung tritt auf, wenn du den Befehl LOAD ohne Dateinamen gibst.

```
Schließendes ' fehlt
```

Diese Meldung tritt auf, wenn du im LOAD–Befehl den Dateinamen mit einem Apostroph (') beginnst, aber *exaEdit* keinen zugehörigen Schlussapostroph findet und sich keine rechte Vorstellung über den gemeinten Dateinamen machen kann. Letzteres ist dann der Fall, wenn nach dem Anfangsapostroph irgendwo in der Zeile ein Leerzeichen steht, nach dem noch andere Zeichen kommen. Mit anderen Worten: Enthält der Dateiname, den du in Apostrophe einschließen möchtest, keine Leerzeichen, so darfst du den Schlussapostroph weglassen.

```
Zugriff nicht erlaubt
```

Diese Meldung tritt auf, wenn dir der Zugriff auf die Datei nicht erlaubt ist, etwa, weil sie jemand anders gehört.

```
Verzeichnis kann nicht editiert werden
```

Diese Meldung tritt auf, wenn du keinen Dateinamen, sondern ein Verzeichnis laden wolltest. *exaEdit* kann nur einzelne Dateien bearbeiten (siehe aber auch Abschnitt 3.1.3.5, *Laden aller Dateien eines Verzeichnisses*).

```
Keine Datei und kein Verzeichnis
```

Diese Meldung tritt auf, wenn das Objekt, das du laden wolltest, weder eine Datei noch ein Verzeichnis ist. Es kann dann nicht editiert werden.

```
Keine Verbindung zu anderem Rechner
```

Diese Meldung tritt auf, wenn bei vernetzten Rechnern das Aufsuchen einer Datei oder die Überprüfung der Zugriffsberechtigung die Dienste eines anderen Rechners („server“) benötigt, aber das Betriebssystem die Verbindung zu diesem Rechner nicht aufnehmen kann.

```
Teil des Namens kein Verzeichnis
```

Diese Meldung tritt auf, wenn du einen qualifizierten Dateinamen angegeben hast (mehrere durch „/“ bzw. „\“ verbundene Teilnamen) und nicht alle Teile außer dem letzten Verzeichnisse sind.

```
Zu viele symbolic links, Verweis auf sich selbst?
```

Diese Meldung tritt auf, wenn das Betriebssystem Teile des Dateinamens, die auf andere Namen verweisen, auflösen möchte und dabei die im Betriebssystem vorgesehene Maximalzahl solcher Verweise übertroffen wird. Die häufigste Ursache für diesen Fehler ist ein Dateiname, der unmittelbar oder mittelbar auf sich selbst verweist.

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

Diese Meldungen sollten nie auftreten. Sie kommen dann, wenn bestimmte Fehler auftreten, für die *exaEdit* keine besondere Meldung vorgesehen hat. Du solltest dann die vollständige Meldung zusammen mit den näheren Umständen festhalten und diese Information dem Autor von *exaEdit* zukommen lassen.

```
Datei nicht geöffnet (nicht da?)
```

Diese Meldung tritt auf, wenn die Vorprüfungen, die *exaEdit* vornimmt, alle positiv ausgehen, aber die Datei dennoch nicht zum Lesen geöffnet werden konnte.

3.1.3.2 Laden einer Datei mittels DD–Name

Als Alternative zur Angabe eines Dateinamens beim `LOAD`–Befehl kannst du eine Umgebungsvariable benutzen, die du vorher auf bestimmte Art mit dem Dateinamen verbunden hast.

Hast du beispielsweise die Umgebungsvariable `DD_ABC` in einem Unix–System mittels

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

oder in einem Windows–System mittels

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

definiert, so kannst du die Datei in *exaEdit* mit dem Befehl

```
LOAD (abc)
```

laden. Dabei unterscheiden die Klammern den hier besprochenen Sonderfall vom Normalfall eines Dateinamens.

Im `LOAD`–Befehl kann der DD–Name („Datei-Definition“) beliebig mit großen oder kleinen Buchstaben geschrieben werden, aber in der Umgebungsvariablen in Unix nur mit Großbuchstaben. Die drei Zeichen `DD_` müssen Bestandteil der Umgebungsvariablen sein, damit selbstdefinierte Variablen nicht so leicht mit schon vorhandenen kollidieren.

Der Befehl `FILE` unterstützt bisher noch nicht die Verwendung von DD–Namen. Du kannst selbstverständlich eine mit `load (abc)` geladene Datei mit `file` wieder abspeichern, da *exaEdit* ja weiß, welche Datei es ist. Versuchst du aber etwa `file (abc)`, so erstellt *exaEdit* eine Datei namens „`(abc)`“.

3.1.3.3 Dateien mit besonderen Satzformaten

exaEdit operiert normalerweise auf konventionellen Textdateien, bei denen Sätze durch ein Linefeed–Zeichen (`x0A`) oder durch die beiden Zeichen Carriage Return (`x0D`) und Linefeed voneinander getrennt sind (diese Zeichen nach dem letzten Satz sind üblich, werden auch von *exaEdit* geschrieben, sind aber für das Lesen nicht erforderlich).

Es gibt, insbesondere in anderen Betriebssystemen als Unix oder Windows, auch andere Satzformate. Eines davon kann *exaEdit* lesen: Variabel lange Sätze, deren Längenangabe in einem 2 Byte langen Feld am Anfang eines Satzes steht. Das Längenfeld selbst ist in der Längenangabe nicht mit eingeschlossen.

Beim Lesen einer solchen Datei wertet *exaEdit* die Längenangaben aus und fasst nach diesen die gelesenen Daten zu Sätzen zusammen. Soll ein so entstandener workfile wieder geschrieben werden, so kann *exaEdit* derzeit nur eine herkömmliche Textdatei (siehe oben) erstellen.

Eine weitere Möglichkeit, Daten anders als durch Abteilen mittels `x0A` zu lesen, bietet die Verwendung des Befehls

```
WIDTH
```

Einzelheiten dazu findest du bei der Beschreibung des Befehls.

3.1.3.4 Parameter für große Dateien

Insbesondere bei sehr großen Dateien kann es vorteilhaft sein, nicht die ganze Datei, sondern nur einen Teil davon zu laden. LOAD kennt dazu drei Parameter:

```
COUNT
```

lädt die Datei nicht, sondern zählt nur, wieviele Sätze sie enthält. Gleichzeitig wird berechnet, wieviele Bytes der workfile belegen würde. Diese Zahl ist größer als die Datenmenge auf dem Datenträger, da *exaEdit* zu jedem Datensatz noch einige Bytes an Kontrollinformationen benötigt. Die ausgewiesene Zahl ist andererseits kleiner als das, was das Programm *exaEdit* mit den geladenen Daten im Hauptspeicher des Computers benötigt. Die Meldung für COUNT lautet:

```
Gezählte Sätze: ..., Workfile-Größe: ...
```

Die ausgewiesene Maßeinheit bei der workfile–Größe ist B, KB, MB oder GB.

Gibst du beim Laden gleichzeitig MULTIPLE an, so musst du den Parameter COUNT vor MULTIPLE schreiben, es sei denn, dass MULTIPLE den Unterparameter /kette/ hat.

Wenn du nur die ersten n Sätze einer Datei laden möchtest, verwendest du

```
RECORDS n
```

Möchtest du die ersten Sätze einer Datei überspringen, verwendest du dafür den Parameter

```
IGNORE n
```

Du kannst natürlich auch beide zusammen angeben, auch zusammen mit COUNT. IGNORE und RECORDS sind nicht möglich, wenn du den Parameter MULTIPLE verwendest.

Den Hilfetext zum Befehl LOAD, der sich mit COUNT, IGNORE und RECORDS beschäftigt, erhältst du mit

```
HELP LOADX
```

3.1.3.5 Laden aller Dateien eines Verzeichnisses

exaEdit bietet die Möglichkeit, alle Dateien eines Verzeichnisses auf einmal in einen workfile zu laden, dort Änderungen vorzunehmen und alle Dateien wieder in das Verzeichnis zurückzuschreiben.

Möchtest du dieses mehrfache Laden verwenden, so geht dies nur mit dem *exaEdit*–Befehl LOAD, nicht jedoch beim Aufruf von *exaEdit*. Der notwendige Parameter heißt

```
MULTIPLE
```

Seine Minimalabkürzung ist M.

Über die Reihenfolge, in der die einzelnen Dateien geladen werden, kann nichts ausgesagt werden, es ist auf jeden Fall nicht notwendig die alphabetische.

Enthält das Verzeichnis Unterverzeichnisse, so werden diese ignoriert.

Damit die Information, welche Sätze zu welcher Datei des Verzeichnisses gehören, auch im workfile vorhanden ist, wird vor jeden ersten Satz einer Datei ein sogenannter Trennsatz in den workfile geschrieben. Dieser Trennsatz hat normalerweise die Gestalt

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

wobei die Zeichen DDD durch den jeweiligen Dateinamen ersetzt sind.

Du hast die Möglichkeit, Trennsätze eigener Wahl zu verwenden, indem du bei LOAD den Parameter

```
MULTIPLE /kette/
```

angibst. Die Begrenzungszeichen der Trennkette sind wie bei *exaEdit* üblich beliebig. Kommt in der Trennkette die Zeichenfolge DDD vor, so wird sie beim Laden durch den jeweiligen Dateinamen ersetzt. Möchtest du ohne jeden Trennsatz arbeiten, so gibst du eine leere Zeichenkette (//) an.

Wird LOAD ... MULTIPLE ... in einem leeren workfile ausgeführt, so merkt sich *exaEdit*, dass der workfile auf diese Weise entstanden ist und zeigt dies mit dem Buchstaben M in der Statuszeile. Außerdem zeigen die Top– und Statuszeile anstelle eines Dateinamens den Verzeichnisnamen. Auswirkungen hat dies bei der Ausführung des Befehls FILE.

Die Angabe des Verzeichnisnamens kann auf die übliche Art erfolgen, zum Beispiel ist . (= ein Punkt) der Name des aktuellen Verzeichnisses.

Folgende Fehlermeldungen kannst du (zusätzlich zu denen bei LOAD sonst bekannten Meldungen) erhalten:

```
Verzeichnis nicht gefunden
```

Dies bedeutet, dass du MULTIPLE angegeben hast und kein Objekt dieses Namens zu finden war.

```
Objekt ist kein Verzeichnis
```

Dies bedeutet, dass du MULTIPLE angegeben hast, das gesuchte Objekt auch vorhanden ist (im Gegensatz zur vorigen Meldung), dass es sich aber nicht um ein Verzeichnis, sondern vermutlich um eine Datei handelt.

```
Verzeichnis nicht geöffnet
```

Dies bedeutet, dass trotz aller vorheriger Prüfungen, die sämtlich bestanden wurden, das Verzeichnis nicht bearbeitet werden kann. Die genaue Ursache kann nicht angegeben werden.

Wurden aus dem Verzeichnis Dateien geladen, so erhältst du danach die Meldung

```
... Datei[en] geladen
```

Enthält das Verzeichnis auch Unterverzeichnisse, so erhältst du auch noch die Meldung

```
... Unterverzeichnis[se] übergangen
```

Du hast auch die Möglichkeit, lediglich Dateien mit bestimmten Namen zu laden oder alle Dateien mit bestimmten Namen auszuschließen. Die Parameter dafür heißen

```
SELECT ...      bzw.      EXCLUDE ...
```

Beide Parameter müssen vor dem Parameter MULTIPLE stehen, es sei denn, MULTIPLE enthält auch den Unterparameter /kette/: dann ist ihre Stellung beliebig. Ist einer von beiden angegeben, so wird automatisch auch MULTIPLE angenommen.

Wie gibst du die gewünschten Dateinamen an? Die erste Möglichkeit besteht in der Angabe eines workfile–Namens. In diesem workfile müssen dann zeilenweise die Dateinamen stehen. Die zweite Möglichkeit besteht darin, eine durch Leerzeichen getrennte Liste von Dateinamen in runden Klammern hinzuzufügen. Beispiele:

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

Damit werden aus dem Verzeichnis alle Dateien geladen, die im workfile LISTE aufgeführt sind, abzüglich jedoch der beiden Dateien ABC und XY. Selbstverständlich kannst du auch die Parameter SELECT und EXCLUDE einzeln verwenden.

Bei den Dateinamen musst du nicht unbedingt vollständige Namen angeben. Ein Fragezeichen (?) steht für ein beliebiges Zeichen, ein Stern (*) für eine (auch leere) Folge beliebiger Zeichen, in eckige Klammern ([]) werden Zeichen eingeschlossen, von denen wenigstens eins an der bezeichneten Stelle vorkommen muss. In der eckigen Klammer kannst du auch durch ein Paar von Zeichen, das mit einem Minuszeichen (-) verbunden ist, einen Bereich von Zeichen angeben: alle Zeichen, die lexikalisch zwischen den beiden angegebenen liegen. Soll in der eckigen Klammer

das Minuszeichen oder die schließende eckige Klammer als Auswahlzeichen vorkommen, so musst du sie als erstes oder letztes Zeichen bringen. Beginnen die Zeichen in der eckigen Klammer mit dem Ausrufezeichen (`!`), so werden diejenigen Dateinamen ausgewählt, die die folgenden Zeichen an der betrachteten Stelle nicht enthalten.

Den Hilfetext zum Befehl `LOAD`, der sich mit `EXCLUDE` und `SELECT` beschäftigt, erhältst du mit

```
HELP LOADX      und
HELP LOADY
```

3.1.4 Speichern einer Datei

Das Speichern einer Datei erfolgt nicht automatisch, sondern erfordert die Eingabe des Befehls

```
file [datei]
```

Gibst du keinen Dateinamen an, so wird der Inhalt des workfiles (ohne die top line) in die dem workfile zugeordnete Datei geschrieben. Den Namen der zugeordneten Datei findest du in der Statuszeile ab Spalte 19 oder in der top line nach den Wörtern `TOP LINE`. Ist der zugeordnete Dateiname länger als 41 Zeichen, so steht in der Statuszeile nur sein Anfang, und du musst notfalls die top line heranziehen, um den Dateinamen vollständig sehen zu können.

Gibt es keinen dem workfile zugeordneten Dateinamen, so erhältst du die Meldung

```
Parameter fehlt
```

Gibst du dagegen im Befehl `FILE` einen Dateinamen an, so wird in diese Datei geschrieben, und die dem workfile zugeordnete Datei mit davon verschiedenem Namen bleibt unverändert.

Der Dateiname, sei es der dem workfile zugeordnete oder der von dir im Befehl `FILE` angegebene, kann ein absoluter oder ein relativer Dateiname sein.

Hier müssen wir jetzt zwischen Unix– und Windows–Systemen unterscheiden.

In Unix–Systemen:

Der absolute Dateiname beginnt mit einem Schrägstrich, der relative nicht. Gibst du einen relativen Dateinamen an, so wird er durch Voransetzen deines aktuellen Arbeitsverzeichnisses zu einem absoluten Dateinamen ergänzt. Wenn du wissen möchtest, wie das aktuelle Arbeitsverzeichnis heißt, kannst du dafür den *exaEdit*–Befehl

```
call pwd      oder      _pwd
```

aufrufen, der dir das Ergebnis des Unixbefehls `pwd` (pathname of the working directory) zeigt. Ein Beispiel: Ist

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

dein aktuelles Arbeitsverzeichnis, so beschreiben die Befehle

```
file etc/eins      bzw.      file /etc/eins
```

die Dateien

```
/u/fmath/ppreus/exaEdit/etc/eins      bzw.      /etc/eins
```

Du kannst beim Dateinamen mit der dafür üblichen Schreibweise auch einen Wechsel ins übergeordnete Verzeichnis verlangen, also in obigem Beispiel mittels

```
file ../abc/zwei
```

in die Datei

```
/u/fmath/ppreus/abc/zwei
```

schreiben, obwohl dein Arbeitsverzeichnis

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

ist.

Schließlich kannst du beim Dateinamen auch die Schreibweise mit der Tilde (~) verwenden. In

```
~/...
```

steht die Tilde für dein Home–Verzeichnis, in

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

steht ~uid für das Home–Verzeichnis von uid.

In Windows–Systemen:

Der absolute Dateiname beginnt mit einem Rückstrich („\“) oder mit einem Laufwerksbuchstaben, der relative nicht. Gibst du einen relativen Dateinamen an, so wird er durch Voransetzen deines aktuellen Arbeitsverzeichnisses zu einem absoluten Dateinamen ergänzt. Wenn du wissen möchtest, wie das aktuelle Arbeitsverzeichnis heißt, kannst du dafür den *exaEdit*–Befehl

```
call cd      oder      _cd
```

geben, der dir das Ergebnis des DOS–Befehls cd zeigt. Ein Beispiel: Ist

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

dein aktuelles Arbeitsverzeichnis, so beschreiben die Befehle

```
file winnt\win.ini      bzw.      file \winnt\win.ini
```

die Dateien

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

Du kannst beim Dateinamen mit der dafür üblichen Schreibweise auch einen Wechsel ins übergeordnete Verzeichnis verlangen, also in obigem Beispiel mittels

```
file ..\abc\zwei
```

die Datei

```
d:\pe\abc\zwei
```

schreiben, obwohl dein Arbeitsverzeichnis

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

ist.

Ab hier wieder für **Unix– und Windows–Systeme** gemeinsam.

Enthält der von dir im Befehl FILE angegebene Dateiname Leerzeichen, Apostrophe oder den Befehlsseparator, so musst du den Dateinamen in Apostrophe einschließen, etwa

```
file 'a ;'
```

wenn der Dateiname aus dem Buchstaben a, einem Leerzeichen und dem Semikolon (deinem Befehlsseparator) bestehen soll. Apostrophe als Bestandteil des Dateinamens musst du dabei durch 2 aufeinanderfolgende Apostrophe wiedergeben.

Bevor der workfile tatsächlich in die Datei geschrieben wird, stellt *exaEdit* fest, ob es sich um eine bereits existierende Datei handelt oder ob es eine neue Datei ist. Du erhältst dann eine der beiden Meldungen

```
Alte Datei, drücke J oder Y, um sie zu ersetzen:
Neue Datei, drücke J oder Y, um sie zu erstellen:
```

Diese Fragen dienen dazu, die Gefahr zu verringern, dass du durch Vertippen einen ungewollten Dateinamen angibst.

Willst du den Prozess des Speicherns wie gewünscht fortsetzen, so brauchst du nur die Taste „`J`“ oder „`Y`“ (für „ja“ oder „yes“) zu drücken (Kleinbuchstabe genügt), die `Enter`–Taste ist nicht erforderlich. Drückst du — versehentlich oder absichtlich — eine andere Taste, so erhältst du die Meldung

```
ACHTUNG: Daten nicht gesichert!
```

und der Bildschirmalarm, sofern vorhanden, ertönt. Hast du aber die Erlaubnis gegeben und ist alles gut gegangen, so erhältst du die Meldung

```
Daten gesichert
```

Damit du die beiden zuletzt beschriebenen Meldungen auch wirklich siehst, ist es notwendig, dass du auf die Aufforderung „`drücke J oder Y ...`“ die entsprechende Taste drückst, ohne danach die `Enter`–Taste zu drücken.

Beim Speichern eines workfiles kannst du die folgenden Fehlermeldungen erhalten:

```
Datei kann nur gelesen werden
```

Die Zugriffsrechte für die Datei, in die geschrieben werden soll, besagen, dass nur gelesen werden kann.

```
Datei nicht gefunden
```

Diese Meldung tritt auf, wenn *exaEdit* die gewünschte Datei nicht findet. Vielleicht hast du einen Tippfehler gemacht oder die Datei befindet sich in einem anderen Verzeichnis.

```
Dateisystem kann nur gelesen werden
```

Die Zugriffsrechte für das Verzeichnis, in das geschrieben werden soll, besagen, dass nur gelesen werden kann.

```
Kein Home-Verzeichnis für ... gefunden
```

Diese Meldung tritt auf, wenn du `~uid` verwendet hast (s.o.) und das Betriebssystem kein Home–Verzeichnis für `uid` findet.

```
Keine Datei und kein Verzeichnis
```

Diese Meldung tritt auf, wenn das Objekt, in das du speichern wolltest, weder eine Datei noch ein Verzeichnis ist. Es kann dann nicht editiert werden.

```
Keine Verbindung zu anderem Rechner
```

Um festzustellen, ob du auf die gewünschte Datei zugreifen darfst, muss das Betriebssystem bei einem anderen Computer als dem, auf dem du gerade arbeitest, nachfragen. Dieser andere Computer oder die Leitung dorthin sind aber gerade außer Betrieb. Wahrscheinlich kannst du in diesem Augenblick gar keine Datei lesen oder schreiben.

```
Schließendes ’ fehlt
```

Diese Meldung tritt auf, wenn du im `FILE`–Befehl einen Dateinamen mit einem Apostroph (`’`) beginnst, aber *exaEdit* keinen zugehörigen Schlussapostroph findet und sich keine rechte Vorstellung über den gemeinten Dateinamen machen kann. Letzteres ist dann der Fall, wenn nach dem Anfangsapostroph irgendwo in der Zeile ein Leerzeichen steht, nach dem noch andere Zeichen kommen. Mit anderen Worten: Enthält der Dateiname, den du in Apostrophe einschließen möchtest, keine Leerzeichen, so darfst du den Schlussapostroph weglassen, sonst nicht.

```
Teil des Namens kein Verzeichnis
```

Ein Bestandteil des absoluten Dateinamens, welcher nicht der letzte ist, wurde nicht als Verzeichnis, sondern zum Beispiel als Datei vorgefunden.

```
Verzeichnis kann nicht editiert werden
```

Diese Meldung tritt auf, wenn du nicht in eine Datei, sondern in ein Verzeichnis speichern wolltest. *exaEdit* kann nur einzelne Dateien bearbeiten.

```
Verzeichnis nicht gefunden
```

Diese Meldung erhältst du, wenn in dem Dateinamen ein Verzeichnis nicht gefunden wurde.

```
Zu viele symbolic links, Verweis auf sich selbst?
```

Diese Meldung tritt auf, wenn das Betriebssystem den Dateinamen, der Verweise auf andere Dateien enthält, nicht auflösen kann. Ursache ist meist eine Zirkeldefinition, d.h. ein (auch indirekter) Verweis auf sich selbst.

```
Zugriff nicht erlaubt
```

Diese Meldung tritt auf, wenn dir der Zugriff auf die Datei nicht erlaubt ist, etwa, weil sie jemand anders gehört.

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

und ähnliche Meldungen. Diese Meldungen sollten nie auftreten. Sie kommen dann, wenn bestimmte Fehler auftreten, für die *exaEdit* keine besondere Meldung vorgesehen hat. Du solltest dann die vollständige Meldung zusammen mit den näheren Umständen festhalten und diese Information dem Autor von *exaEdit* zukommen lassen.

Beim Schreiben des workfiles auf den Datenträger werden jedem Satz alle Leerzeichen am Ende weggenommen und (auch beim letzten Satz) ein newline–Zeichen (`\n`, `x0a`) angefügt.

3.1.5 Beenden von *exaEdit*

exaEdit wird mit dem Befehl `QUIT` oder `END` beendet. Die beiden Befehle sind gleichwertig.

Um dich vor einem versehentlichen Beenden von *exaEdit* zu schützen, überprüft *exaEdit* vor dem Aufhören, ob es in der *exaEdit*–Sitzung noch workfiles gibt, die geändert, aber noch nicht abgespeichert wurden. Hast du nur einen workfile überhaupt in der *exaEdit*–Sitzung (das ist dann `MAIN`), so schreibt *exaEdit* die Meldungen

```
Änderungen nicht gesichert
Drücke J oder Y, um aufzuhören:
```

in das Fenster. Befindet *exaEdit* sich im Zeilenmodus, so lautet die zweite Zeile:

```
Gib J oder Y ein, um aufzuhören:
```

Gibt es in der *exaEdit*–Sitzung dagegen mehr als einen workfile, so lautet die erste Meldung statt dessen

```
Nicht gesicherte workfiles: ...
```

wobei anstelle der Punkte alle workfiles erscheinen, die geändert aber nicht gesichert wurden.

Möchtest du jetzt auf das Sichern des oder der workfiles verzichten, brauchst du nur die Taste `J` (für „ja“) oder `Y` (für „yes“) zu drücken, und *exaEdit* hört sofort auf. Bitte beachte, dass du die `Enter`–Taste nicht zu drücken brauchst, es reicht die Taste `J` oder `Y`. Selbstverständlich brauchst du `J` oder `Y` auch nicht als Großbuchstabe zu drücken, es reicht die Taste alleine.

Willst du statt dessen das Sichern der workfiles nachholen oder aus anderem Grunde *exaEdit* nicht sofort beenden, so musst du nur anstelle von `J` oder `Y` irgendeine andere Zeichentaste drücken. Daraufhin wird das Beenden abgebrochen, und du befindest dich ganz normal wieder in der *exaEdit*–Sitzung.

Beachte übrigens, dass ein workfile auch dann schon als geändert gilt, wenn du ein Zeichen durch sich selbst ersetzt hast.

In seltenen Fällen kann *exaEdit* vermutlich nur in Unix–Systemen auch nach Beenden der *exaEdit*–Sitzung noch eine Meldung in das Fenster schreiben:

```
exaEdit: Escape-Folgen statt Tasten: ...
```

Damit hat es folgende Bewandtnis: Die Information, dass eine Sondertaste, wie zum Beispiel *Cursor nach rechts* gedrückt wurde, erreicht eine bestimmte Instanz im Betriebssystem manchmal in der Form einer Folge von anderen Tasten. So kann zum Beispiel *Cursor nach rechts* die Folge der 3 Tasten `Esc[C` sein. Kommen diese 3 Zeichen in genügend kurzem Abstand, so werden sie von der genannten Betriebssysteminstanz in die Information *Cursor*

nach rechts umgesetzt. Ist ihr Abstand zeitlich zu groß, etwa bei Belastungen der Leitung zwischen Terminal und Rechner, so werden sie als Einzelzeichen weitergegeben, die für *exaEdit* in dieser Form unbrauchbar wären. *exaEdit* hat daher einen Mechanismus, die Einzelzeichen wieder logisch zu *Cursor nach rechts* zusammenzusetzen. Dabei wird gezählt, wie oft der Fall vorkommt, und die Anzahl der Vorkommnisse nach Beenden der eigentlichen *exaEdit*–Sitzung herausgeschrieben.

Ist die genannte Zahl klein, so kannst du die Meldung ignorieren. Handelt es sich jedoch um größere Zahlen, so ist etwas faul. *exaEdit* setzt zwar in den erkannten Fällen alles richtig zusammen, es kann aber auch Fälle geben, die *exaEdit* nicht erkennt.

3.1.6 Struktur und Eingabe von Befehlen

Befehle sind immer Wörter oder Zusammensetzungen aus der englischen Sprache oder einzelne Sonderzeichen. Mit Wörtern als Befehle soll erreicht werden, dass du dir die Befehle leichter merken kannst. Die Benutzung von Sonderzeichen ist aus dem gleichen Grund auf ein Minimum beschränkt, Doppeltasten gibt es gar keine.

Damit andererseits der Tippaufwand nicht zu groß wird, kannst du die meisten Befehle abkürzen, etwa `C` für `CHANGE`, `CO` für `COPY`, `DE` für `DELETE` usw. Außerdem kannst du in den Befehlen alle Leerzeichen weglassen, die nicht aus syntaktischen Gründen notwendig sind.

Zwischen der erlaubten Minimalabkürzung und dem ausführlichen Befehlsnamen sind alle Zwischenstufen erlaubt, so dass du als Anfänger mit den ausgeschriebenen Befehlsnamen anfangen und dich nach Belieben an die vom Profi benutzten Minimalabkürzungen heranarbeiten kannst. Beispiel: `change chang chan cha ch c` sind erlaubte Schreibweisen für den `CHANGE`–Befehl.

Befehle können Operanden besitzen, die normalerweise durch Leerzeichen vom Befehlsnamen und voneinander abgetrennt sind. Wie schon angedeutet, kannst du die trennenden Leerzeichen weglassen, wenn das entstehende Gebilde noch eindeutig interpretierbar ist.

Ein Beispiel: Der Befehl `COPY` kopiert die angegebenen Zeilen (von der ersten angegebenen bis zur zweiten angegebenen einschließlich) hinter die aktuelle Zeile. (Der `COPY`–Befehl wird an anderer Stelle noch ausführlich inhaltlich beschrieben, hier geht es nur um ein Beispiel zur Verdeutlichung der möglichen Schreibweisen.)

```
copy 100 200
```

Die Minimalabkürzung von `COPY` ist `CO`, das Leerzeichen vor der Angabe 100 ist entbehrlich, so dass die kürzeste Schreibweise

```
co100 200
```

lautet. Als dritter Operand kann der Name eines workfiles angegeben werden (aus dem die Zeilen herauskopiert werden). Da ein solcher Name (etwa `abc`) mit einem Buchstaben beginnen muss, kann er ohne Leerzeichen an den Operanden `200` angeschlossen werden:

```
co100 200abc
```

Enthält der Kopierbereich die erste oder die letzte Zeile des workfiles, so brauchen diese Zeilen nicht mit ihrer Nummer angegeben zu werden, sondern können statt dessen mit den symbolischen Werten `f` (für first line) bzw. `l` (für last line) bezeichnet werden. In diesem Fall musst du mindestens

```
co f l abc
```

schreiben; hier sind alle Leerzeichen erforderlich.

Mehrere Befehle kannst du auf einmal eingeben, wie du aus dem nächsten Abschnitt sehen kannst.

3.1.7 Verketten von Befehlen, Befehlsseparator

Beim Editieren gibt es oft den Fall, dass du eine Folge von Befehlen geben musst, die du schon voraussehen kannst. Beispiel:

- suche den nächsten Satz, der die Zeichenfolge `'abc'` enthält,
- gehe von dort 2 Sätze zurück,
- ändere in diesem Satz `'xyz'` in `'12'` um.

Die dafür nötigen Befehle sehen etwa so aus:

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

Diese Befehle können verkettet werden, indem du sie, getrennt durch den Befehlsseparator „;“, in eine Zeile schreibst und auf einmal mit `Enter` abschickst:

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

(Selbstverständlich kannst du dabei die Befehle auch ausschreiben, so wie sie oben einzeln aufgeführt wurden.)

Der Vorteil der Verkettung liegt beispielsweise darin, dass einige der Tätigkeiten, die *exaEdit* ausführen muss, dann nur 1–mal statt 3–mal gemacht werden müssen, was zu einer Zeitersparnis führt. Eine andere Verwendung der Befehlsverkettung findest du im Abschnitt 3.1.22, *Befehlsspeicher*.

Der Befehlsseparator ist voreingestellt auf das Zeichen „;“. Du kannst aber auch jedes andere Zeichen dafür verwenden, außer dem Fragezeichen und außer Ziffern. Der Befehl zum Ändern oder zum Ansehen, welches der Befehlsseparator ist, heißt `CMDSEP`. Bitte beachte, dass der Befehl zum Ändern des Befehlsseparators der letzte Befehl in einer Eingabezeile sein muss. Weitere Einzelheiten in Abschnitt 3.2, *Die Befehle*.

Auf manchen Tastaturen ist das Zeichen „;“ nur über die `Shift`–Taste zu erreichen. Für ein so häufig verwendetes Zeichen, das der Effizienz deiner Arbeit dienen soll, ist dies eine schlechte Plazierung. In einem solchen Fall ist es vielleicht besser, ein anderes, direkt erreichbares Zeichen zu verwenden, wofür sich dann oft das Komma „,“ anbietet. Damit du nicht bei jedem Start von *exaEdit* den Befehl

```
cmdsep ,
```

eingeben musst, kannst du das Problem ein für allemal lösen, indem du den genannten Befehl in der *exaEdit*–Profildatei unterbringst, siehe dazu Abschnitt 3.1.26, *Die Profildateien*.

Bitte beachte, dass du den Befehlsseparator nicht ändern musst, wenn das Zeichen „;“ Bestandteil einer Zeichenkette in einem Befehl ist. Beispielsweise sucht `l/ab` nach der Zeichenkette `'ab'`. Willst du den `LOCATE`–Befehl mit einem weiteren Befehl verketten, so brauchst du nur `l/ab/;change ...` zu schreiben. Der Befehl `l/ab;` würde nach der Zeichenkette `'ab;'` suchen. Diesen Befehl mit einem anderen zu verketten, könnte so aussehen: `l/ab;/;change ...` Die Begrenzung von Zeichenketten hat also Vorrang vor der Erkennung des Befehlsseparators.

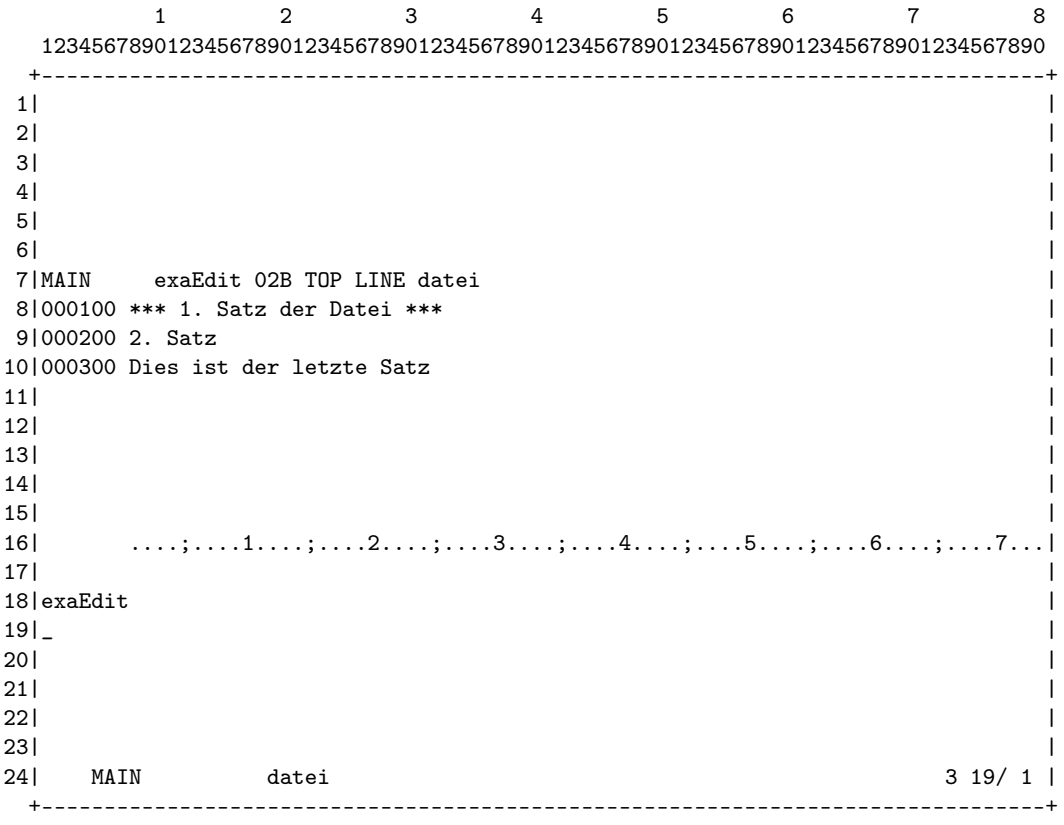
3.1.8 Fensteraufbau, aktuelle Zeile

Der Editor *exaEdit* operiert auf den Sätzen einer Datei und betrachtet sie dazu als Zeilen (Beispiel einer Datei mit 3 Sätzen):

```
*** 1. Satz der Datei ***
2. Satz
Dies ist der letzte Satz
```

exaEdit nutzt das ganze zur Verfügung stehende Fenster für seine Arbeit. Wenn du die Fenstergröße während des Arbeitens mit *exaEdit* veränderst (das geht leider nicht in allen Betriebssystemen), nimmt *exaEdit* von dieser Änderung Notiz und passt sich automatisch der neuen Größe an. Ist das Fenster zu klein oder sieht sich das Betriebssystem nicht in der Lage, den Fenstermodus zu verwenden (jeweils Beschreiben des kompletten Fensters), so arbeitet *exaEdit* im Zeilenmodus. Den Zeilenmodus, vgl. Abschnitt 3.1.20, kannst du auch explizit verlangen (*exaEdit*–Befehl `scope off`).

exaEdit zeigt im oberen Teil des Fensters, der sogenannten Datenzone, den Inhalt des workfiles bzw. (meist) Teile davon. Der untere Teil des Fensters dient im wesentlichen (Details folgen später) der Eingabe von Befehlen und der Ausgabe von Antworten des Editors und wird Dialogzone genannt. Das Fenster könnte also für den oben vorgestellten workfile wie folgt aussehen (wir nehmen ein 24–zeiliges Fenster an):

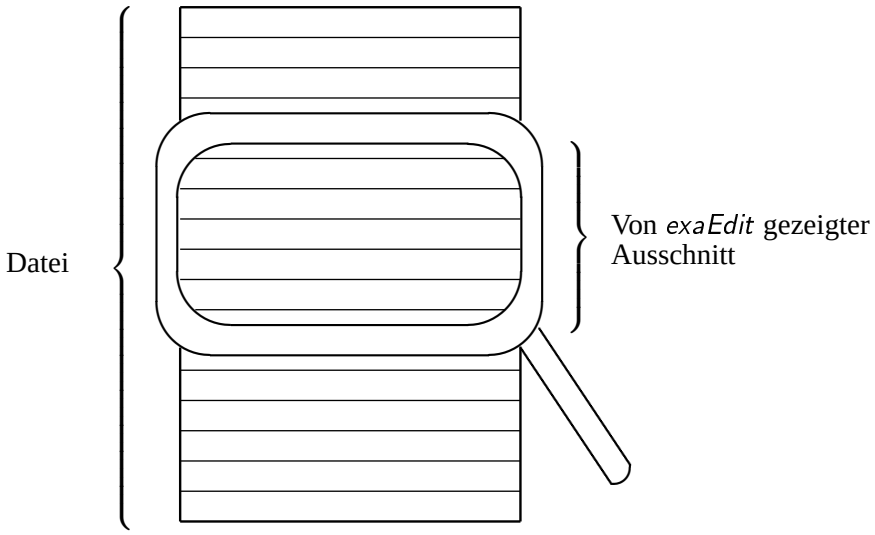


Die Numerierung oben (1 – 80) und links (1 – 24) und der Rahmen sind im Fenster nicht zu sehen, sie dienen nur der Verständigung in dieser Anleitung. Die Datenzone, in der die Sätze des workfiles oder Teile davon zu sehen sind, umfasst die Zeilen 1 bis 15. In Zeile 16 ist ein Lineal zu sehen, das das Abzählen von Spalten erleichtern soll. Zeile 17 bis 23 ist die Dialogzone. Zeile 24 ist die Statuszeile, in der ein sogenannter workfile–Name (`MAIN`), der Name der gerade bearbeiteten Datei (`datei`), sowie ganz rechts außen die Anzahl der Sätze des workfiles (3) und die Zeilen– und Spaltennummer der jeweiligen Cursorstellung (`19/ 1`) stehen.

Für die folgenden Überlegungen ist vorausgesetzt, dass jeder Satz im workfile genau eine Zeile in der Datenzone belegt, wenn er dort gezeigt wird. Dies muss nicht so sein, wie du später noch sehen wirst, die Vorstellung ist jedoch nützlich, weil sie es erlaubt, die Begriffe Zeile und Satz vorerst austauschbar zu verwenden.

Die Zeile 7 enthält (sozusagen als 0. Satz) einen zusätzlichen Satz des workfiles mit dem workfile–Namen (`MAIN`), dem Editornamen `exaEdit`, der Versionsnummer von *exaEdit* (`02B`), der Identifizierung `TOP LINE` und dem Namen der gerade bearbeiteten Datei (`datei`). Diese top line wird beim Laden eines workfiles aus einer Datei gebildet und beim Schreiben eines workfiles in eine Datei weggelassen.

Hat eine Datei mehr Sätze, als in das Fenster passen (das ist der Normalfall), so zeigt *exaEdit* jeweils einen bestimmten Ausschnitt im Fenster.



Dieser Ausschnitt wandert gemäß den verwendeten *exaEdit*–Befehlen über die Datei. Präge dir dieses Bild ein: Der Ausschnitt wandert über die Datei wie eine Leselupe über ein Blatt Papier. Dagegen ist die Vorstellung, der Ausschnitt sei wie ein feststehendes Fenster, hinter dem die Datei vorbeigezogen wird, falsch. Der *exaEdit*–Befehl `+5` zum Beispiel schiebt den Ausschnitt um 5 Sätze vorwärts, das heißt dann, 5 Sätze nach unten in Richtung Ende der Datei.

Der Ausschnitt enthält eine ungerade Anzahl von Zeilen. Es gibt daher eine mittlere Zeile, die optisch und logisch hervorgehoben ist. Sie wird als aktuelle Zeile (genauer: Zeile des aktuellen Satzes, current line) bezeichnet. Alle Befehle, in denen keine Zeilenangabe enthalten ist, beziehen sich auf die aktuelle Zeile. Zum Beispiel verlangst du mit dem Befehl `delete 3` („lösche 3 Zeilen“), dass der Satz der aktuellen Zeile und die beiden nächsten Sätze aus dem workfile gelöscht werden.

Die aktuelle Zeile ist zwar im Fenster an fester Stelle, nicht aber im workfile. Bei jeder Änderung im workfile wird anschließend die aktuelle Zeile auf die letzte bearbeitete Zeile gestellt, der vom workfile gezeigte Ausschnitt also immer so gelegt, dass die Stelle, die vermutlich zuletzt das Ziel deiner Aufmerksamkeit war, in der Mitte des Fensterausschnitts liegt. Bei der Beschreibung der einzelnen Befehle ist jeweils vermerkt, ob und gegebenenfalls wie die aktuelle Zeile verschoben wird. Mit ein wenig Übung weißt du irgendwann bei den meisten Befehlen, wo nach ihrer Ausführung die aktuelle Zeile stehen wird, und kannst diese Information beim Verketten von Befehlen (siehe 3.1.7) mit Vorteil nutzen.

Jeder Satz im workfile hat eine Satznummer. Diese Nummer wird beim Laden der Datei in den workfile festgelegt. Es handelt sich dabei um eine durchgezählte Nummer, die Standardzählung ist `100, 200, 300, ...`

Die Zeilen in der Datenzone, die Sätze aus dem workfile enthalten, sind standardmäßig so aufgebaut:

```
Spalte          11111   77778
        12345678901234...67890
Inhalt  nnnnnnfddddddd...ddddd
```

- Spalte 1 – 6 (`nnnnnn`) ist das Nummernfeld.
- Spalte 7 (`f`) ist das Flaggenfeld, in dem bestimmte Eigenschaften der Zeile markiert sind (Details später, im Normalfall ist das Feld leer).
- Spalte 8 – 80 (`dd...dd`) ist das Datenfeld.

Die Breite des Nummernfeldes (Voreinstellung: 6) kann geändert werden (0 bis 8 Zeichen), das Datenfeld passt sich in seiner Länge automatisch an.

Mit den genannten Zahlenwerten können also zunächst einmal Sätze bis 73 Zeichen Länge in der Datenzone gezeigt werden. Oft gibt es aber längere Sätze. Gibst du nichts anderes an, so werden die übrigen Teile des Satzes in Folgezeilen dargestellt. Diese Folgezeilen sind am leeren Nummernfeld zu erkennen:

```
000100 ------ZEHN---ZWANZIG--DREISSIG---VIERZIG...SIEBZIG---
       ACHTZIG---NEUNZIG---HUNDERT
000200 Ein Satz, der in eine Zeile passt.
```

In diesem Beispiel sind zwei Sätze zu sehen: der Satz mit der Nummer 200, dessen Inhalt in 1 Fensterzeile passt, und der Satz mit der Nummer 100, für dessen 100 Zeichen 2 Fensterzeilen benötigt werden.

Da eine solche Darstellung nicht immer erwünscht ist, gibt es die Möglichkeit, eine logische Fensterbreite (Befehl `LWWIDTH`) zu definieren. Wird diese etwa auf 110 gesetzt, so verwandelt sich das obige Beispiel in

```
000100 ------ZEHN---ZWANZIG--DREISSIG---VIERZIG...SIEBZIG---
000200 Ein Satz, der in eine Zeile passt.
```

Hier hast du dann (aber noch nicht in der gegenwärtigen Version von *exaEdit*) die Möglichkeit, das Sichtfenster nach rechts zu verschieben, um den hinteren Teil des Satzes 100 zu sehen (Befehl `szone`). Wird `szone` etwa auf 30 gesetzt, so siehst du im Fenster

```
000100 G---VIERZIG...NEUNZIG---HUNDERT
000200 asst.
```

also alle Sätze ab dem 30. Zeichen.

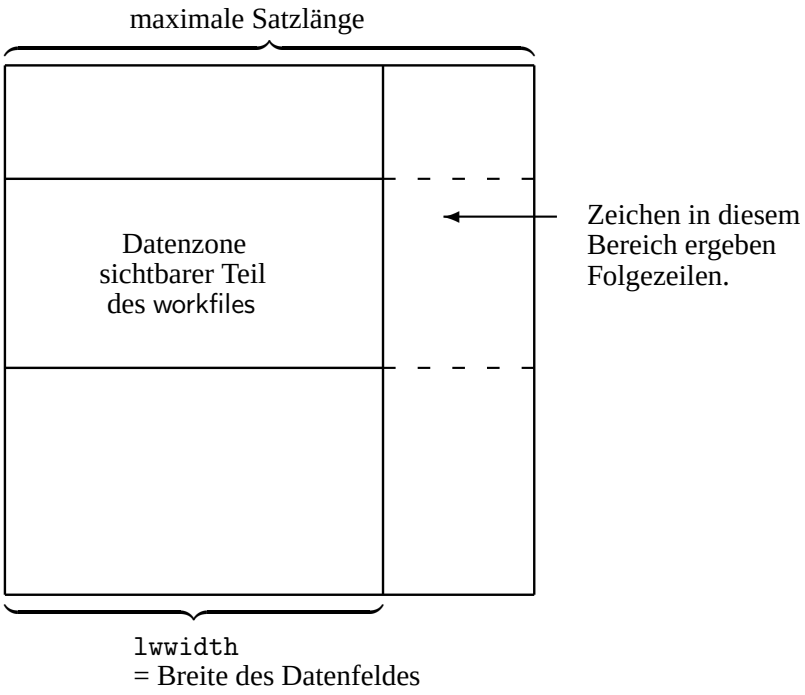
Was passiert, wenn `LWWIDTH` auf 90 gesetzt wird (und `szone` wieder die Normaleinstellung 1 hat)?

```
000100 ------ZEHN---ZWANZIG--DREISSIG---VIERZIG...SIEBZIG---
       ---HUNDERT
000200 Ein Satz, der in eine Zeile passt.
```

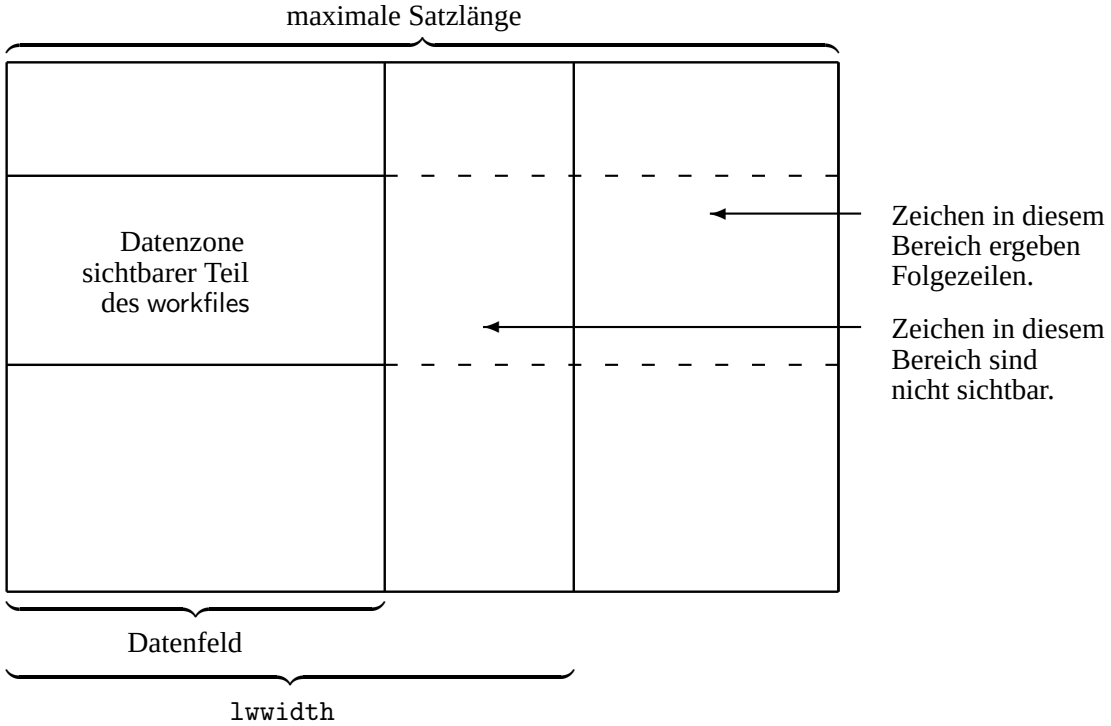
Es wird also immer der Teil, der den Wert von `LWWIDTH` übersteigt, in Folgezeilen dargestellt. Willst du gar keine Folgezeilen haben, so muss `LWWIDTH` $\geq$ längster Satz sein.

Noch mal eine schematische Darstellung

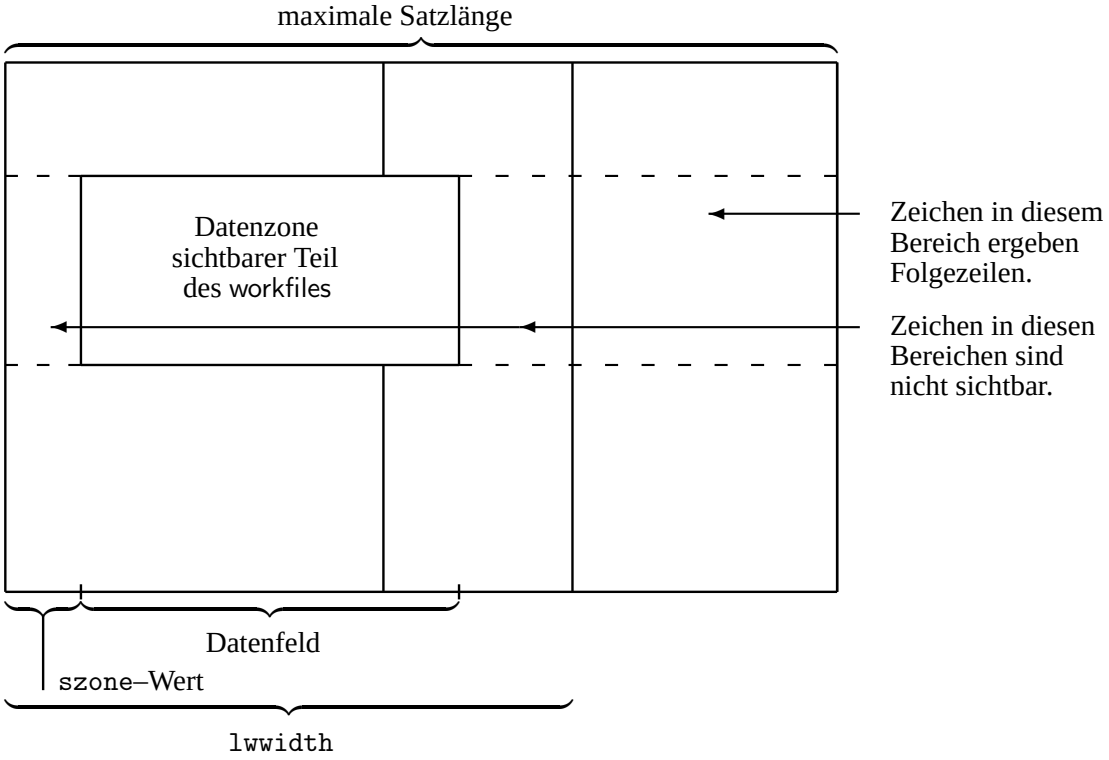
1. Fall: Logische Fensterbreite (`lwwidth`) = Breite des Datenfeldes,
linker Rand = 1. Zeichen (`szone 1`):



2. Fall: Logische Fensterbreite (`lwwidth`) > Breite des Datenfeldes,
linker Rand = 1. Zeichen (`szone 1`):



3. Fall: Logische Fensterbreite (`lwwidth`) > Breite des Datenfeldes,
linker Rand = n–tes Zeichen (`szone n` mit `n > 1`):
(Hinweis: Diesen Fall kann es in der gegenwärtigen Version von *exaEdit* noch nicht geben.)



Für die Folgezeilen gilt natürlich ebenso, dass durch die Angabe von `szone` ihr gezeigter Ausschnitt verändert werden kann, und dass Zeichen, die jenseits der Spalte `LWWIDTH` sind, erneut Folgezeilen ergeben.

3.1.9 Satznummern

Alle Sätze des workfiles haben eine Nummer, die beim Laden der Datei erzeugt, beim Abspeichern aber wieder weggelassen wird. Die Nummern werden von manchen Befehlen zur Angabe ihres Arbeits- oder Zielbereichs benötigt.

Die Satznummern liegen im Bereich von 0 bis 99999999. Normalerweise trägt der erste Satz die Nummer 100 und der Abstand zweier Nummern ist ebenfalls 100. Beim Editieren entfallen manchmal Zeilen, andere Zeilen kommen hinzu oder die Reihenfolge von Zeilen ändert sich. Die Numerierung bleibt dabei immer streng aufsteigend. Da am Anfang zwei Satznummern einen Abstand von 100 haben, passen Änderungen oft dazwischen. *exaEdit* ist bestrebt, Zeilennummern nach Möglichkeit beizubehalten. Geht es nicht mehr, so numeriert *exaEdit* teilweise neu. Die Nummernabstände liegen dann nur bei 20 (statt 100), so dass die Wahrscheinlichkeit groß ist, dass mit diesen kleinen Schritten schon sehr schnell wieder eine vorhandene Satznummer erreicht ist, die (wie alle folgenden) bestehen bleiben kann. *exaEdit* meldet ein Neunumerieren mit der Meldung

```
Neu numeriert
```

Möchtest du selbst neu numerieren, so verwendest du dafür den Befehl

```
REKEY
```

eventuell mit Parametern für einen Anfangswert und einen Abstand deiner Wahl. Näheres siehe bei `REKEY`.

Normalerweise werden die an sich 8–stelligen Satznummern mit 6 Ziffern ausgewiesen (`000100`, `000200` usw.). Wird eine Datei mit mehr als 10000 oder mehr als 100000 Zeilen geladen, so wird die Nummernbreite 7– oder 8–ziffrig.

Diese Breite kannst du mit jederzeit mit dem Befehl

```
SKEY
```

(`SKEY` steht für screen key) und geeignetem Parameter selbst einstellen (Näheres siehe bei `SKEY`).

Reicht die Breite des Nummernfeldes nicht aus, um alle signifikanten Ziffern einer Nummer darzustellen, so wird der optische Ausfall führender Ziffern durch einen * zwischen Nummer und Daten angezeigt, etwa

```
234500*Daten
```

wenn die Nummer eigentlich 1234500 wäre. Normalerweise kannst du durch geeignete Wahl von SKEY die *–Anzeigen verschwinden lassen, was immer zu empfehlen ist, damit zwischen dir und *exaEdit* keine Missverständnisse über die Bedeutung etwa der Zeilennummer 234500 auftreten. Notfalls musst du mittels REKEY ... die Numerierungsabstände verkleinern.

3.1.10 Löschen und Einfügen von Zeichen

Prinzipiell kannst du jedes Zeichen im *exaEdit*–Fenster löschen, ausgenommen sind nur die top line, die Statuszeile und die Nummernfelder in der Datenzone. Zum Löschen gibt es meist 2 Tasten.

Die Entf–Taste, oft auch mit Del oder anders beschriftet, löscht das Zeichen unter dem Cursor. Alles was rechts davon steht, wird um 1 Stelle nach links gerückt (manchmal auch der Inhalt der nächsten Zeilen bei mehrzeilig dargestellten Sätzen, was aber jetzt nicht weiter interessiert).

Die Backspace–Taste, oft durch einen Pfeil nach links dargestellt, löscht das Zeichen vor dem Cursor. Wie bei der anderen Taste wird alles, was rechts von dem gelöschten Zeichen steht, um 1 Stelle nach links gerückt.

Das Einfügen von Zeichen ist etwas komplizierter, weil es vom Überschreiben eines vorhandenen Zeichens zu trennen ist. Es gibt zwar heutzutage viele Programme, die auf die Möglichkeit des Überschreibens eines Zeichens verzichten und nur Löschen und Einfügen anbieten, aber nicht so *exaEdit*.

Die Einschränkungen für das Einfügen sind dieselben wie für das Löschen eines Zeichens.

Um im Editorfenster ein Zeichen einfügen zu können, muss *exaEdit* in den Einfügemodus (bitte vom Eingabemodus unterscheiden) versetzt werden. Dies geschieht im allgemeinen durch Drücken der Einfügetaste. Das Bestehen des Einfügemodus wird in der Statuszeile durch das Zeichen ∧ angezeigt. Solange der Einfügemodus gilt, dienen eingetippte Zeichen nicht dem Überschreiben am Ort des Cursors, sondern dem Einfügen an dieser Stelle. Alle rechts davon befindlichen Zeichen werden um 1 Stelle nach rechts verschoben.

In der Voreinstellung des Editors wird der Einfügemodus durch das Drücken der Enter–Taste (oder einer Funktionstaste) beendet. Soll er vorher beendet werden, weil zum Beispiel die nächsten Zeichen dem Überschreiben dienen sollen, so kannst du dafür die Einfügetaste erneut drücken, falls du nach dem Einschalten des Einfügemodus wenigstens 1 Zeichentaste gedrückt hast. Den hier beschriebene Einfügemodus nennen wir den gewöhnlichen Einfügemodus (im Unterschied zu dem im nächsten Absatz beschriebenen).

Nun also der dauerhafte Einfügemodus: Wenn du es vermmeiden möchtest, dass der Einfügemodus mit jeder Enter–Taste beendet wird, so kannst du das dadurch erreichen, dass du die Einfügetaste 2-mal nacheinander (ohne eine andere Taste dazwischen) drückst. Wenn du in diesem dauerhaften Einfügemodus die Einfügetaste wieder drückst, so schaltet er sich wieder aus.

Zusätzlich zum Drücken der Einfügetaste gibt es noch den Befehl

```
INSMODE
```

INSMODE ON schaltet in den dauerhaften Einfügemodus (wie das doppelte Drücken der Einfügetaste), INSMODE OFF schaltet den Einfügemodus aus (wie das einfache Drücken der Einfügetaste). Damit hast du die Möglichkeit, auch mit einer solchen Tastatur Zeichen einfügen zu können, auf der es keine Einfügetaste gibt. Der Befehl INSMODE und die Einfügetaste sind natürlich auch im Austausch verwendbar.

3.1.11 Setzen und Aufsuchen von Markierungen

Es kommt sicher öfters vor, dass du dir eine Stelle in der zu editierenden Datei merken möchtest, um sie später wieder aufzusuchen. Dazu kannst du zwar die Zeilennummer verwenden, aber das ist umständlich und kann danebengehen, wenn neu numeriert wurde.

Die elegantere Methode zum Markieren besteht in der Verwendung des Befehls

```
set
```

Die Bearbeitung dieses Befehls besteht darin, dass sich *exaEdit* den Satz der aktuellen Zeile merkt.

Mit dem Befehl

```
return
```

kannst du von jeder Stelle der Datei aus zu dem markierten Satz zurückkehren.

Gibst du den Befehl SET ein weiteres Mal, so wird die alte Markierung aufgehoben und erneut die aktuelle Zeile markiert.

Mit dem Befehl

```
set ?
```

kannst du dich informieren, zu welchem Satz ein folgendes RETURN zurückkehren würde.

Gibst du den Befehl RETURN, ohne vorher im selben workfile mit SET eine Markierung gesetzt zu haben, so erhältst du die Meldung

```
SET-Speicher unbenutzt
```

Hast du den Satz mit einer SET–Markierung gelöscht, so liefern dir die Befehle SET ? und RETURN den vorhergehenden Satz. Damit du darauf aufmerksam wirst, erhältst du in beiden Fällen die zusätzliche Meldung

```
SET-Speicher geändert, Rückkehr zum vorhergehenden Satz
```

Die Meldung wird erst dann nicht mehr erzeugt, wenn du eine neue SET–Markierung gemacht hast.

Den SET–Speicher gibt es für jeden workfile einzeln. Es ist also nicht möglich, mittels RETURN in einen anderen workfile zurückzukehren.

Die mit SET markierte Zeile hat die symbolische Zeilennummer s, die du in allen Befehlen verwenden kannst, in denen eine Zeilennummer anzugeben ist.

3.1.12 Positionieren

Mit Positionieren ist das Verschieben der aktuellen Zeile (siehe Abschnitt 3.1.8, *Fensteraufbau, aktuelle Zeile*) gemeint. Du kannst direkt oder indirekt positionieren.

Indirekt positionierst du durch direkte Datenänderung in der Datenzone oder durch Erteilen eines Befehls, der als Nebeneffekt die aktuelle Zeile verändert.

Änderst du Daten direkt, so wird nach dem Drücken der Enter–Taste diejenige geänderte Zeile zur aktuellen, die am weitesten unten war.

Bei Befehlen, die Daten ändern, wird die letzte geänderte Zeile zur aktuellen. Werden Zeilen gelöscht, so wird die vorhergehende zur aktuellen. Werden Zeilen eingefügt, so wird die letzte eingefügte zur aktuellen. Bei der Einzelbeschreibung der Befehle in Abschnitt 3.2.3 wird jedesmal genau beschrieben, ob bzw. wie sich die aktuelle Zeile verändert.

Für die direkte Positionierung gibt es eine Reihe von Befehlen. Vorwärts, das heißt, in Richtung Dateiende, positionierst du mit einem der Befehle

```
next
down
+
```

die alle drei identisch in Parametern und Ausführung sind. Die Anzahl der Sätze, um die du weiterschieben willst, ist entweder 1, wenn du keinen Parameter angibst, oder die im Befehl angegebene Zahl `n`. Schießt du über das Ziel hinaus, also zum Beispiel mit dem Befehl `NEXT 5`, wenn die aktuelle Zeile 3 Zeilen vor der letzten steht, so erhältst du die Fehlermeldung

```
Datenende
```

und die aktuelle Zeile bleibt stehen.

Willst du die aktuelle Zeile auf die letzte Datenzeile positionieren, so verwendest du den Befehl

```
bottom
```

Rückwärts, das heißt, in Richtung Dateianfang, positionierst du mit einem der Befehle

```
back
up
-
```

die ebenfalls alle drei identisch in Parameter und Ausführung sind und entsprechend den Befehlen für die Vorwärtspositionierung funktionieren, nur dass die Fehlermeldung

```
Datenanfang
```

heißt, wenn du versuchst, übers Ziel hinauszuschießen.

Zum Positionieren der aktuellen Zeile auf die top line verwendest du den Befehl

```
top
```

Möchtest du auf die erste Datenzeile positionieren, so kannst du das etwa so machen:

```
top; next
```

Mit dem Befehl `POINT` positionierst du die aktuelle Zeile auf den im Befehl angegebenen Satz, Beispiel

```
point 400
```

Die erste Datenzeile kannst du daher auch kurz so erreichen:

```
po f
```

Mit den Befehlen `LOCATE` und `RLOCATE` zum Aufsuchen bestimmter Daten kannst du ebenfalls direkt positionieren. Einzelheiten dazu sind in Abschnitt 3.1.14, *Suchen* beschrieben.

Mit dem Befehl `RETURN` kannst du auf einen vorher markierten Satz positionieren. Einzelheiten dazu sind in Abschnitt 3.1.11, *Setzen und Aufsuchen von Markierungen*, beschrieben.

Eine besondere Art des Positionierens ist das Blättern, welches im folgenden Abschnitt beschrieben wird.

3.1.13 Blättern

Verwandt mit dem im vorigen Abschnitt beschriebenen Positionieren der Datei ist das Blättern in der Datei. Im Kern ist es ebenfalls ein Positionieren, das aber zum Teil anderen Regeln gehorcht.

Du kannst um eine ganze oder eine halbe Seite vorwärts oder rückwärts blättern. Für dieses Blättern gibt es keine *exaEdit*–Befehle, sondern statt dessen *exaEdit*–Funktionen.

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

Diese (und andere, siehe Abschnitt 3.1.30) *exaEdit*–Funktionen sind beim Start von *exaEdit* mit bestimmten Tasten der Tastatur verbunden oder können mithilfe des Befehls PFK mit den F–Tasten verbunden werden.

Die Voreinstellung beim Start von *exaEdit* ist

Funktion	Tasten
'+page'	F8, Bild ↓
'-page'	F7, Bild ↑
'+half'	F11
'-half'	F10

Alle vier genannten Funktionen können mit dem Befehl PFK (siehe Abschnitt 3.1.23, *Programmierbare Funktionstasten*) mit beliebigen F–Tasten verbunden werden.

Wenn diese *exaEdit*–Funktionen im Fenster angezeigt oder ausgeführt werden, sind sie zur Verdeutlichung in die entsprechenden *exaEdit*–Befehle + bzw. - umgerechnet.

Da das Blättern so geht, dass etwa beim Vorwärtsblättern die zuvor letzte Zeile zur anschließend ersten Zeile des ganzen oder halben Fensters wird, ergibt sich bei einem 24–zeiligen Fenster mit einer deshalb 15–zeiligen Datenzone folgende Umrechnung

Funktion	Ergebnis
'+page'	+14
'-page'	-14
'+half'	+7
'-half'	-7

Das bedeutet zum Beispiel, dass das Drücken der Taste F10, die mit '+half' verbunden ist, das gleiche bewirkt, die die Eingabe des Befehls +7.

Allerdings gibt es doch einige Abweichungen von dieser Gleichsetzung. Die 1. Abweichung besteht darin, dass bei einem Hinausschießen über die top line oder die letzte Datenzeile bei der *exaEdit*–Funktion die aktuelle Zeile auf die top line bzw. die letzte Zeile gesetzt wird, während beim normalen Positionierungsversuch der Befehl mit der Meldung Datenanfang bzw. Datenende verweigert, also die aktuelle Zeile nicht verändert wird.

Die 2. Abweichung ist bisher nur geplant: Wenn es Sätze gibt, die wegen ihrer Länge in Folgezeilen dargestellt werden, so soll in späteren Versionen von *exaEdit* darauf Rücksicht genommen werden, also die Daten beim Blättern nahtlos gezeigt werden, während beim Positionieren mit beispielsweise +14 die aktuelle Zeile um 14 Sätze vorgeschoben werden soll, auch wenn dabei zwischen den beiden gezeigten Datenausschnitten ungezeigte Daten liegen.

Die 3. Abweichung besteht darin, dass beispielsweise der *exaEdit*–Befehl +14 unabhängig von der Fenstergröße ausgeführt wird, während die *exaEdit*–Funktion '+page' nur bei einem 24– oder 25–zeiligen Fenster in "+14" resultiert, aber bei anderen Zeilenzahlen entsprechend andere Werte hat.

3.1.14 Suchen

Mit „Suchen" ist das Aufsuchen von Sätzen gemeint, die bestimmte Zeichenfolgen enthalten bzw. nicht enthalten. Gewöhnlich verwendest du zum Suchen nach einer Zeichenkette den Befehl

```
locate /Zeichenkette/
```

der in Abschnitt 3.2 ausführlich beschrieben ist.

Von hinten nach vorne kannst du mit dem Befehl RLOCATE („reverse locate") suchen.

Mit den Befehlen NLOCATE („negative locate") und RNLOCATE („reverse negative locate") oder NRLOCATE („negative reverse locate") erhältst du die jeweils nächste Zeile, die die Suchkette nicht enthält.

Da die Minimalabkürzung für LOCATE nur L ist und *exaEdit* sich die zuletzt verwendete Suchkette merkt, ist das wiederholte Suchen nach der gleichen Kette sehr einfach: du brauchst nur den Befehl L zu geben. Bitte denke auch

daran, dass du den LOCATE–Befehl auf eine F–Taste legen kannst (vgl. Abschnitt 3.1.30), so dass du mit jeweils einem Tastendruck auch verschiedene Zeichenketten aufsuchen kannst.

Die gespeicherte Suchkette ist dieselbe für alle Formen des LOCATE–Befehls. Du brauchst also zum Rückwärtssuchen nach der Zeichenkette, die du zuvor vorwärts gesucht hast, nur den Befehl RLOCATE zu geben.

3.1.15 Ändern von Daten, Überblick

Es gibt drei verschiedene Methoden für die Änderung der Daten

1. direkte Änderungen
2. Befehle
3. Präfixbefehle

3.1.15.1 Direkte Änderungen

Damit ist gemeint, dass die in den Datenfeldern befindlichen Zeichen geändert, gelöscht oder neu eingefügt werden können. Alle diese Änderungen werden erst wirksam, das heißt, dass sie die Daten des workfiles ändern, wenn die Enter-Taste gedrückt wird. Mit direkten Änderungen können keine Zeilen hinzugefügt oder weggenommen werden, dafür brauchst du Befehle oder Präfixbefehle.

3.1.15.2 Befehle

Befehle werden in einer Befehlszeile eingegeben (normalerweise unterhalb der Linealzeile) und dienen

- dem Ändern des workfiles,
  Beispiele: DELETE, CHANGE,
- dem Ändern der Darstellung,
  Beispiele: LWWIDTH, SKEY,
- der Positionierung der aktuellen Zeile,
  Beispiele: NEXT, TOP, LOCATE,
- und noch manch anderem.

Befehle können in jeder Zeile, außer der Statuszeile (das ist die unterste Zeile) eingegeben werden, doch ist meist am sinnvollsten, sie in einer Zeile unterhalb der Linealzeile einzugeben, also dort, wo der Cursor nach dem Drücken der Enter–Taste hingestellt wird.

Einzelheiten findest du insbesondere im Abschnitt 3.2.

3.1.15.3 Präfixbefehle

Dies sind besondere Befehle, die im Nummern–(=Präfix–)Bereich einer Zeile eingegeben werden und in oder für diese Zeile wirksam sind. Beispiele sind u.a.

d[n]	zum Löschen (delete) von n Zeilen
dd	zum Löschen eines Bereiches von Zeilen
i	zum Einfügen (insert) einer Leerzeile
"	zum Verdoppeln einer Zeile

Einzelheiten findest du in Abschnitt 3.3.

3.1.15.4 Reihenfolge der Bearbeitung

Alle Änderungen, seien es direkte oder durch Befehle oder Präfixbefehle veranlasste, werden erst wirksam, wenn die Enter–Taste gedrückt wird.

Dabei dürfen beliebig viele Änderungen aus allen drei Klassen gemacht werden. Die Bearbeitungsreihenfolge ist diese: Das Fenster wird von oben nach unten überprüft, welche Zeilen geändert wurden. Dabei gilt eine Zeile auch dann als geändert, wenn ein Zeichen durch sich selbst ersetzt wurde. Aus der Stellung der Zeile im Fenster, aus ihrem vorigen und aus ihrem jetzigen Inhalt wird erschlossen, um welche Art von Änderung es sich handelt.

Eine geänderte Zeile, in der sich zuvor kein Satz des workfiles befunden hat (also jenseits der Datenzone oder in der Datenzone vor der ersten oder nach der letzten benutzten Zeile) wird immer als Befehl interpretiert.

Für alle anderen geänderten Zeilen gilt: Hat das Nummernfeld denselben Inhalt wie zuvor, handelt es sich um eine direkte Änderung. Ist im geänderten Nummernfeld ein gültiger Präfixbefehl erkennbar, so handelt es sich um einen Präfixbefehl. Zusätzlich wird vorher das eventuell geänderte Datenfeld als direkte Änderung behandelt. Ist dagegen im geänderten Nummernfeld kein gültiger Präfixbefehl, so wird die gesamte Zeile als Befehl behandelt.

Falls dir die vorstehenden Ausführungen zu abstrakt sind, solltest du sorgfältig die Beispiele im Kapitel *Erste Schritte* durcharbeiten.

3.1.16 Löschen von Zeilen

Zum Löschen von Zeilen gibt es verschiedene Möglichkeiten.

Der Befehl DELETE, Minimalabkürzung DE, löscht die aktuelle Zeile und eventuell die folgenden, wenn du dies entsprechend angibst (vgl. Abschnitt 3.2.3).

Der Befehl DELETEL, Minimalabkürzung DL, löscht die angegebenen Zeilen, die du durch ihre Zeilennummer bezeichnen musst. Symbolische Zeilennummern kannst du natürlich auch verwenden. Beispielsweise löscht

```
dl p n
```

die aktuelle Zeile und die unmittelbar oben und unten anschließenden Zeilen.

Zeilen können auch mit Präfixbefehlen gelöscht werden. Es gibt die Befehle d, d*n* und dd.

Alle Zeilen, in deren Nummernfeld du ein „d“ tippst, werden beim Drücken der Enter–Taste gelöscht.

Bei der Form d*n* werden, beginnend mit der Zeile, in der diese Markierung steht, *n* Zeilen gelöscht. Für die Schreibweise d*n* musst du einige Besonderheiten beachten, die in Abschnitt 3.3 beschrieben sind.

Der Präfixbefehl dd muss in zwei Zeilen stehen. Beim Drücken der Enter–Taste werden alle Sätze indem markierten Bereich gelöscht.

3.1.17 Einfügen von Zeilen

Zum Einfügen von Zeilen gibt es verschiedene Möglichkeiten. Zunächst werden die beschrieben, bei denen Zeilen unmittelbar nach der aktuellen Zeile eingefügt werden. Für diese Fälle musst du also den workfile vorher entsprechend positionieren.

Mit dem Befehl INPUT versetzt du *exaEdit* in den Eingabemodus. Er ist daran erkenntlich, dass in der Statuszeile das Zeichen „I“ steht und das Lineal zwischen Daten– und Dialogzone in Spalte 1 beginnt. Alle jetzt eingegebenen Zeilen werden oben in der Datenzone eingefügt. Sie erhalten zunächst noch keine Zeilennummer, sondern erst dann, wenn du den Eingabemodus wieder beendet hast. Das Beenden des Eingabemodus geschieht durch Drücken der Enter–Taste, ohne dass du vorher etwas eingetippt hast.

Da *exaEdit* die Eingabe von Zeichen an jeder Fensterstelle erkennt, kannst du deine Tipparbeit vereinfachen, wenn du nacheinander ähnliche Zeilen einfügen möchtest. Ein Beispiel: Du möchtest die beiden Zeilen

```
23-950320 Neuer Befehl CASE.
22-950320 Neuer Befehl PFK.
```

einfügen. Wenn du im Eingabemodus die erste dieser beiden Zeilen eingetippt und mit Enter in die Datenzone geschafft hast, hast du bei einem Fenster von 24 Zeilen folgende Situation:

- Die erste eingefügte Zeile ist in Zeile 17 des Fensters wiederholt.
- Der Cursor steht eingabebereit am Anfang der Zeile 18.

Statt jetzt in der Zeile 18 die zweite einzufügende Zeile komplett einzutippen, gehst du mit dem Cursor in die Zeile 17 und änderst sie so, wie die zweite einzufügende Zeile aussehen müsste, was in diesem Beispiel sehr viel weniger aufwendig ist. Dann schickst du die Eingabe ganz normal mit der Enter–Taste ab.

Andere Befehle, mit denen du nach der aktuellen Zeile Einfügungen machen kannst, sind COPY, LOAD und MOVE, die in Abschnitt 3.2.3 ausführlich beschrieben sind.

Statt nach der aktuellen Zeile kannst du auch unmittelbar an jeder Stelle Zeilen einfügen, indem du den Nummerbefehl verwendest. Der Nummerbefehl sieht formal wie eine Zeile aus der Datenzone aus:

```
nummer daten
```

wobei das Leerzeichen zwischen den beiden Teilen nicht erforderlich ist. Hast du beispielsweise einen workfile mit den Zeilennummern 100, 200, 300, ... vor dir und gibst den Befehl

```
120 qwert
```

so fügst du damit die Zeile „000120 qwert“ ein. Dasselbe Ergebnis hättest du mit „120qwert“ erreicht, während „120 qwert“ die Zeile „000120 qwert“ ergeben hätte.

Besonders effektiv wird der Nummerbefehl, wenn du ihn nicht vollständig zu tippen brauchst, sondern auf Vorhandenes zurückgreifen kannst. Stell dir vor, du hast den workfile

```
000100 Neuer Befehl CASE.
000200 irgendwas
```

und möchtest dazwischen als 2. Zeile den Text „Neuer Befehl PFK.“ haben. Am einfachsten ist es, wenn du mit dem Cursor in die Datenzone in die Zeile 100 fährst, dort die Nummer in 000150 und das Wort CASE in PFK abänderst und das ganze mit der Enter–Taste abschickst. Keine Angst, die übertippte Zeile 100 erscheint wieder und nach ihr die gewünschte Zeile 150. Du hast hier zwei *exaEdit*–Eigenschaften auf einmal verwendet: einmal den Nummerbefehl, zum andern die Tatsache, dass du Eingabe an beliebiger Stelle (also auch in der Datenzone) machen darfst.

Du musst natürlich aufpassen, dass du die Nummer richtig wählst. Verwendest du nämlich eine schon vorhandene Zeilennummer, so wird diese Zeile überschrieben, was vielleicht von dir gewollt ist, aber nicht mehr in den Abschnitt *Einfügen von Zeilen* gehört.

Falls du den Nummerbefehl anwenden möchtest, aber die Numerierungsabstände schon zu klein sind, kannst du mit dem Befehl REKEY die 100er–Abstände wiederherstellen.

3.1.18 Eigenschaften des Eingabemodus

Wie du in Abschnitt 3.1.17, *Einfügen von Zeilen*, gesehen hast, kannst du *exaEdit* in den Eingabemodus versetzen und damit neue Zeilen in den workfile einsetzen. Im Eingabemodus, den du mit dem Befehl INPUT startest, hat der Editor weitere Fähigkeiten, die insbesondere bei der Eingabe von Programmen oder Texten vorteilhaft sind.

3.1.18.1 Automatisches Einrücken

Nach jeder eingegebenen Zeile steht der Cursor wieder bereit für die nächste. Die Voreinstellung des Editors ist dabei nicht immer die erste Spalte, sondern diejenige Spalte, die in der Zeile zuvor die erste nicht leere war. Die „Zeile

zuvor“ ist zu Beginn des Eingabemodus die aktuelle Zeile und im weiteren Verlauf die gerade zuvor eingegebene. Die Einrückspalte, die sich der Editor gemerkt hat, übersteht auch die Eingabe von Leerzeilen. Das automatische Einrücken ist insbesondere bei der Eingabe von Programmen vorteilhaft.

Zum Steuern des Einrückverhaltens gibt es den Befehl

```
INDENT
```

Er gibt an, ob automatisches Einrücken stattfinden soll oder nicht. Mit ihm kannst du weiter angeben, ob die Einrückspalte wie beschrieben von der vorherigen Zeile abhängen soll oder ob sie einen festen Wert haben soll. Weitere Einzelheiten zur Syntax findest du im entsprechenden Teil des Abschnitts 3.2, *Die Befehle im einzelnen*.

3.1.18.2 Automatischer Zeilenumbruch

Wenn die Eingabe von Texten mehrere Zeilen umfasst, ist es nützlich, wenn der Editor eine volle Zeile selbständig abschließt und automatisch eine neue beginnt, so dass du dich beim Eintippen voll auf den Text konzentrieren kannst. Ab der Version 02 von *exaEdit* ist dies das Standardverhalten des Editors. Voreingestellt sind Zeilen mit der maximalen Länge `LWWIDTH`. `LWWIDTH` ist die sichtbare Datenbreite in der Datenzone, also etwa 73 bei einem Editorfenster von 80 Zeichen Breite und einem Nummernfeld von 6 Zeichen.

Sobald mehr als 73 Zeichen eingetippt sind, sucht *exaEdit* von hinten her nach der ersten Gruppe von Leerzeichen und schlägt alle Zeichen bis vor dieser Gruppe zur ersten Eingabezeile, die damit als fertig getippt eingegeben wird. Alle Zeichen nach dieser Gruppe werden an den Anfang der nächsten Eingabezeile gestellt, die mit den in der Folge eingetippten Zeichen ergänzt wird.

Zum Steuern des automatischen Zeilenumbruchs gibt es den Befehl

```
INLENGTH
```

Er gibt an, ob der automatische Zeilenumbruch erfolgen soll oder nicht. Mit ihm kannst du weiter angeben, ob die Umbruchspalte wie beschrieben am Ende der sichtbaren Zeile liegen soll, oder ob sie einen angebbaren festen Wert haben soll. Weitere Einzelheiten zur Syntax des Befehls findest du im entsprechenden Teil des Abschnitts 3.2, *Die Befehle im einzelnen*.

3.1.19 Blöcke editieren

exaEdit kennt drei Befehle (`CCOPY`, `CMOVE`, `CDELETE`), mit denen Spalten kopiert, verschoben oder gelöscht werden können (`CCOPY` = column copy).

Da diese Befehle sich auf die gleichen Spalten in mehreren aufeinanderfolgenden Zeilen erstrecken können, ergibt sich die Möglichkeit, in einem workfile „Rechtecke“ von Zeichen mit jeweils einem Befehl zu kopieren, zu verschieben oder zu löschen. Ein Beispiel: Sieht der workfile etwa so aus:

```
000100 Hier ist
000200 ein Text
000300 im Block
000400 ========
```

so sieht er nach dem Ausführen des Befehls

```
ccopy (f l) 1 8 column 10 line 400
```

folgendermaßen aus:

```
000100 Hier ist
000200 ein Text
000300 im Block
000400 ======== Hier ist
000500          ein Text
```

```
000600          im Block
000700          ========
```

Der Befehl bedeutet: Kopiere von der ersten bis zur letzten Zeile des workfiles die Spalten 1 bis 8 in die Spalten ab 10 in die Zeile ab 400.

Ausführlichere Angaben findest du bei der Beschreibung der drei Befehle im Abschnitt 3.2.3.

3.1.20 Der Zeilenmodus

Bezüglich der Benutzung des Fensters kennt *exaEdit* zwei Arten, den Fenstermodus und den Zeilenmodus. Der Normalzustand ist der Fenstermodus, in dem *exaEdit* die Eingabe vom ganzen Fenster erkennt und als Ausgabe immer den ganzen Fenster beschreibt.

Im Zeilenmodus dagegen wird das Fenster wie ein Schreibmaschinenterminal verwendet, wo du und *exaEdit* immer nur neue Zeilen ans Ende des bisher Geschriebenen setzen könnt.

Der Zeilenmodus wird von *exaEdit* nur verwendet, wenn der Fenstermodus nicht möglich ist oder du es mit dem Befehl

```
scope off
```

verlangst. Zurück in den Fenstermodus geht es mit dem Befehl

```
scope on
```

wobei `scope` alleine die gleiche Wirkung hat. Der Zeilenmodus kann für dich von Vorteil sein, wenn du etwas anzeigen lassen möchtest, das in den 7 oder 8 Zeilen der Dialogzone keinen Platz findet, etwa die gesamte Ausgabe des Befehls `HELP`. In einem solchen Fall kannst du die Befehle `scope off; help` geben und später mit `scope on` in den vertrauten Fenstermodus gehen.

Eine andere Anwendung besteht darin, mit `scope off; scope on` den Fenstermodus zu beenden und wieder zu starten, wenn das Betriebssystem mit der Fenstersteuerung durcheinander gekommen ist (was in manchen Systemen leider gelegentlich vorkommt).

Ein nützlicher Befehl für den Zeilenmodus ist

```
display ...
```

mit dem du verlangst, dass *exaEdit* die angegebene Anzahl von Zeilen, beginnend mit der aktuellen Zeile, zeigt.

3.1.21 Programmieren des Editors

Unter dem Programmieren des Editors verstehen wir alle die Möglichkeiten, die es gibt, komplizierte Befehle oder Folgen von Befehlen mit wenigen Tastendrucken zu veranlassen. *exaEdit* hat eine ganze Reihe von Eigenschaften, die dieses Programmieren ermöglichen. Weitere Funktionen werden im Laufe der Zeit noch hinzukommen. Die Flexibilität einer echten Programmiersprache wird *exaEdit* jedoch nicht erreichen, das wäre auch unvernünftig, weil es dafür ja schon genügend andere, auch sehr brauchbare, Programme gibt. Was bietet *exaEdit* auf diesem Gebiet?

- Die programmierbaren Funktionstasten `F1`, `F2`, ..., siehe Abschnitt 3.1.23.
- Die Befehlsspeicher `X` und `Y`, siehe Abschnitt 3.1.22.
- Der Befehlsfolgen–workfile `EXEC`, siehe Abschnitt 3.1.24.
- Die Parametervariablen, siehe Abschnitt 3.1.25.

3.1.22 Befehlsspeicher

exaEdit kann Befehle oder mit dem Befehlsseparator verkettete Befehle (siehe Abschnitt 3.1.7) auch speichern, damit du sie jederzeit bequem abrufen kannst.

exaEdit kennt 2 Arten von Befehlsspeichern. Das sind einmal die F–Tasten, die in Abschnitt 3.1.23 beschrieben sind, zum anderen die beiden *exaEdit*–Befehle `X` und `Y`, die hier näher erläutert werden. Der wesentliche Unterschied ist der, dass du die in `X` oder `Y` gespeicherte Befehlsfolge selbst wieder als Befehl `X` bzw. `Y` in einer Befehlsfolge verwenden kannst, während du die F–Tasten immer nur separat verwenden kannst.

Den Befehlsspeicher `X` setzt du mit dem Befehl

```
x befehlsfolge
```

beispielsweise

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

Zum Ausführen des Befehls genügt dann die Eingabe des Befehls `X`. Oft möchtest du die Befehlsfolge wiederholt ausführen. Dazu brauchst du nur die gewünschte Anzahl dahinterzusetzen.

```
x 17
```

führt die Befehlsfolge 17 mal aus.

Kann ein Befehl der Befehlsfolge nicht ausgeführt werden, beispielsweise wenn das Ende des workfiles erreicht wird, so wird die Ausführung von `X` abgebrochen. Du kannst also etwa das oben definierte `X` mit `X 9999` auch in einem workfile aufrufen, der nicht so viele Sätze enthält, wie auf den ersten Blick notwendig wären.

Mit dem Befehl

```
x ?
```

erreichst du, dass *exaEdit* die in `X` abgelegte Befehlsfolge im Fenster zeigt. Dies ist besonders nützlich, wenn du etwa eine komplizierte Befehlsfolge definiert hast, die du nur leicht verändern möchtest: dann gibst du `X ?` ein, änderst die gezeigte Zeile entsprechend ab und drückst die `Enter`–Taste. Damit hast du `X` erneut (und verändert) definiert.

Der Befehl `Y` ist identisch mit `X`. In der Definition von `X` darfst du `Y` verwenden und umgekehrt. Eine direkt oder indirekt rekursive Definition wird von *exaEdit* beim Ausführen der Befehle erkannt und mit einer der Meldungen

```
Abbruch bei rekursivem X
Abbruch bei rekursivem Y
```

geahndet. Weitere Einzelheiten findest du im Abschnitt 3.2.3, *Die Befehle im einzelnen*, unter `X` oder `Y`.

3.1.23 Programmierbare Funktionstasten

exaEdit erlaubt es, die `F`–Tasten (beschriftet mit `F1`, `F2`, ...) mit Befehlen oder mit Funktionen zu belegen.

Auf den üblichen Tastaturen befinden sich die `F`–Tasten `F1` bis `F12`. Ganz gleich jedoch, wieviele solcher `F`–Tasten vorhanden sind, muss für ihre Nutzung durch *exaEdit* noch die Bedingung erfüllt sein, dass ihr Niederdrücken auch als diese Aktion an das Programm *exaEdit* weitergegeben wird. Dies ist leider nicht immer der Fall. Du kannst prüfen, ob die `F`–Tasten verfügbar sind. Für die `F`–Tasten `F7`, `F8`, `F10` und `F11` muss einer der *exaEdit*–Befehle zum Blättern um eine ganze oder halbe Seite erzeugt werden (etwa `-7` für `F7` in einem Fenster mit 24 Zeilen). Die übrigen `F`–Tasten bringen die Meldung

```
F-Taste ist nicht belegt
```

Erfolgt beim Niederdrücken der `F`–Tasten keine der beiden genannten Reaktionen, so steht sie *exaEdit* nicht zur Verfügung.

Zunächst ein einfaches Beispiel: Du möchtest in einem workfile in jeder 10. Zeile die Ziffer 3 in eine 7 umändern. Zu diesem Zweck könntest du die Befehlszeile

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

entsprechend oft eingeben. Eine Arbeitserleichterung dafür sind die Befehlsspeicher `X` und `Y` (vgl. vorigen Abschnitt und Abschnitt 3.2.3, *Die Befehle im einzelnen*). Noch einfacher ist es, diese Befehlsfolge auf eine der `F`–Tasten, etwa die Taste `F1` zu legen, indem du die Zeile

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

tippst, dann aber nicht die `Enter`–Taste, sondern die Taste `F1` drückst. Als Erfolgsmeldung erscheint

```
F-Taste wurde belegt
```

Jedesmal, wenn du dann die Taste `F1` drückst, wird die gespeicherte Befehlszeile ausgeführt.

Die Belegung der `F`–Taste ist zunächst nicht besonders geschützt. Dies bedeutet, dass die `F`–Taste eine neue Belegung erhält, wenn du etwas eingibst und die `F`–Taste wieder drückst.

Die Belegungen der `F`–Tasten sind für alle workfiles deiner *exaEdit*–Sitzung dieselben. Beim Beenden von *exaEdit* gehen sie verloren.

Neben der oben beschriebenen einfachen Benutzung der `F`–Tasten gibt es noch weitere Möglichkeiten, für die du den Befehl `PFK` („program function key“) brauchst. `PFK` ist ausführlich in Abschnitt 3.2.3, *Die Befehle im einzelnen*, beschrieben, hier nur das Wichtigste.

Mit

```
pfk n ?
```

siehst du die Belegung der `F`–Taste `n`. Lässt du die Zahl `n` weg, siehst du die Belegung aller belegten `F`–Tasten. Das Fragezeichen ist ebenfalls entbehrlich.

Mit

```
pfk n lock
```

schützt du die Belegung der `F`–Taste `n`. Dies bedeutet, dass sich die Taste nicht mehr durch Eingabe von Befehlen und Drücken der Taste belegen lässt. Dadurch ist die `F`–Taste vor dem versehentlichen Belegen geschützt. In den ungeschützten Zustand zurückversetzen kannst du die Taste durch

```
pfk n unlock
```

Das Belegen der `F`–Taste `n` kannst du ebenfalls mit dem Befehl `PFK` machen.

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

Zwischen den Schrägstrichen, die du wie immer durch andere Begrenzungszeichen ersetzen kannst, steht die gewünschte Belegung der Taste. Bei der Belegung mittels `PFK SET` wird die Taste automatisch in den geschützten Zustand versetzt.

Die Belegung von `F`–Tasten mittels `PFK SET` ist die einzige Möglichkeit, *exaEdit*–Funktionen (siehe Abschnitt 3.1.30) mit `F`–Tasten zu verbinden. Fehlt beispielsweise die `Entf`– oder `Del`–Taste auf der Tastatur, oder wird sie nicht an *exaEdit* weitergereicht oder hat sie eine andere Bedeutung, dann kannst du ihre Funktion, nämlich das Zeichen unter dem Cursor zu löschen, mittels

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

auf die Taste `F1` legen.

3.1.24 Befehlsfolgen im Workfile: `EXEC`

Wie du vielleicht schon weißt, kannst du in *exaEdit*–Profildateien *exaEdit*–Befehle ablegen, die dann beim Start von *exaEdit* oder dem Start in einen neuen workfile ausgeführt werden (siehe Abschnitt 3.1.26). Allerdings erfolgt diese Ausführung nur ein einziges Mal und dazu noch vor dem Laden der eventuell zu editierenden Datei. Möchtest du vorbereitete Folgen von *exaEdit*–Befehlen zu beliebiger Zeit ausführen lassen, so gibt es dafür zwar die Befehlsspeicher `X`

und Y und die programmierbaren Funktionstasten F1, F2, ..., aber die Menge der damit ausführbaren *exaEdit*–Befehle ist doch immer sehr begrenzt. Eine Erweiterung dieser Möglichkeit besteht in folgendem:

1. Lege einen workfile namens EXEC an.
2. Fülle diesen workfile mit denjenigen Befehlen, die du ausführen lassen möchtest.
3. Gehe zurück in den workfile, in welchem du eigentlich editieren möchtest.
4. Gib den Befehl EXEC.

Dann werden im aktuellen workfile alle Zeilen des workfiles EXEC der Reihe nach ausgeführt.

Folgende Besonderheiten sind zu beachten:

- Nach dem Befehl EXEC darf derzeit nichts anderes in der *exaEdit*–Befehlszeile stehen (diese Einschränkung wird in späteren *exaEdit*–Versionen fallen).
- Gibt es keinen workfile EXEC, so erhältst du nur die Fehlermeldung

  ```
  Workfile nicht gefunden
  ```

- Kommt im workfile EXEC der Befehl FILE vor, so entfallen bei seiner Ausführung die sonst üblichen Rückfragen der Art

  ```
  Alte Datei, drücke J oder Y, um sie zu ersetzen:
  ```

  usw. Du solltest in diesem Fall also besondere Sorgfalt walten lassen und die verwendeten Dateinamen genau prüfen.

Weitere Einzelheiten findest du im Abschnitt 3.2.3, *Die Befehle im einzelnen*, unter EXEC.

3.1.25 Die Parametervariablen

Die Parameter in *exaEdit*–Befehlen sind zunächst Konstanten, wie zum Beispiel 6 oder /abc/ in den Befehlen NEXT 6 bzw. LOCATE /abc/. Anstelle solcher Konstanten können aber oft auch Variablen verwendet werden, die Parametervariablen genannt werden.

Es gibt drei Typen von Parametervariablen:

- Typ N (numerical). Der Wert einer solchen Variablen ist eine ganze Zahl.
- Typ L (line number). Der Wert ist eine Zeilennummer.
- Typ S (string). Der Wert ist eine Zeichenkette.

Einige Parametervariablen sind von vornherein definiert. Ihr Wert wird durch die Ausführung bestimmter Befehle gesetzt:

- &Col ist vom Typ N und wird von den Befehlen LOCATE und RLOCATE gesetzt, indem die Variable die Spalte angibt, in der das Suchargument gefunden wurde.
- &Count ist vom Typ N und enthält das Ergebnis der Ausführung des Befehls COUNT.
- &Line ist vom Typ L und wird von den Befehlen der LOCATE–Familie gesetzt, indem die Variable die Zeile angibt, in der die Suche endet.
- &Loc ist vom Typ S und wird von den Befehlen der LOCATE–Familie mit dem Suchargument belegt.

Neben diesen immer vorhandenen Parametervariablen kannst du beliebig eigene definieren. Wie das geht, findest du im Abschnitt 3.2.3, *Die Befehle im einzelnen*, unter &.

Mit den Parametervariablen kannst du in gewissem Umfang auch „rechnen“, etwa Konstanten addieren oder Teilketten bilden. Auch diese Möglichkeiten sind in dem genannten Abschnitt beschrieben.

3.1.26 Die Profildateien

exaEdit hat zwar für die meisten Parameter Voreinstellungen, die alles in allem genommen die „besten“ sind, es gibt aber immer wieder Fälle, wo du andere Voreinstellungen besser findest. Für dieses Problem gibt es Profildateien, d.h., Dateien mit *exaEdit*–Befehlen, die jedesmal ausgeführt werden, wenn du *exaEdit* startest. Genaugenommen werden die Profildateibefehle sogar schon vor dem Laden der zu editierenden Datei ausgeführt.

Es gibt 2 Arten von Profildateien für *exaEdit*:

- die Installationsprofildatei
- die private Profildatei

Das Installationsprofil heißt so, weil es für den installierten Editor gilt, ganz gleich, wer ihn (bei Mehrbenutzersystemen) aufruft, während das private Profil an die aufrufende Benutzeridentifikation gebunden ist. Im einzelnen geht *exaEdit* so vor:

exaEdit findet sein Installationsprofil über die Umgebungsvariable

```
EXAEDITIP
```

Ob diese vorhanden ist und welchen Wert sie gegebenenfalls aufweist, kannst du in Unix–Systemen mit dem Unix-Befehl

```
echo $EXAEDITIP
```

in Windows–Systemen mit dem Windows-Befehl

```
set exaeditip
```

feststellen. Beginnt der Wert von `EXAEDITIP` mit dem Zeichen „-“, so bedeutet dies, dass kein Installationsprofil vorhanden ist.

Ist jedoch die Umgebungsvariable `EXAEDITIP` gar nicht vorhanden, so verwendet *exaEdit* statt dessen die Datei `.exaEditip` in dem Verzeichnis, von dem aus *exaEdit* aufgerufen wurde. Gibt es eine solche Datei nicht, so arbeitet *exaEdit* ohne das Installationsprofil.

Die private Profildatei erstellst du bei Bedarf selbst. Sie heißt

```
.exaeditpp
```

und wird von *exaEdit* zuerst in dem Verzeichnis gesucht, von dem aus *exaEdit* aufgerufen wurde. Gibt es dort keine solche Datei, so sucht *exaEdit* sie in deinem Home–Verzeichnis. Gibt es dort auch keine, so arbeitet *exaEdit* ohne privates Profil.

exaEdit findet das Home-Verzeichnis in Unix-Systemen über die Umgebungsvariable `HOME`, in Windows-Systemen über die beiden Umgebungsvariablen `HOMEDRIVE` und `HOMEPATH`. Deren Werte kannst du wie oben beschrieben mit den Befehlen `echo` bzw. `set` feststellen.

exaEdit bearbeitet zuerst das Installationsprofil und dann das private, so dass du in deiner eigenen Profildatei Befehle, die dir im Installationsprofil nicht gefallen, durch entsprechende eigene überschreiben kannst.

Ein nützliches Beispiel für einen Eintrag in der Profildatei ist der Befehl

```
cmdsep ,
```

der den Befehlsseparator für die Verkettung von *exaEdit*–Befehlen von „`;`“ auf „`,`“ setzt. Dieses Umsetzen ist von Vorteil, wenn das Zeichen „`;`“ nur über die `Shift`–Taste, das Zeichen „`,`“ dagegen ohne die `Shift`–Taste eingegeben werden kann.

Wenn *exaEdit* gestartet wird und – wie oben beschrieben – auf verschiedenen Wegen die Profildateien sucht, so hält es die vorgefundenen Verhältnisse fest, ohne dich direkt zu informieren: die Profilbearbeitung soll „lautlos“ erfolgen. Du kannst dich jedoch jederzeit während der *exaEdit*–Sitzung informieren, wie das mit den Profilen beim Start von *exaEdit* war, indem du den Befehl

```
profile ?
```

aufrufst. Die von *exaEdit* präsentierten Meldungen sollten selbsterklärend sein. Die Kurzfassung der von *exaEdit* im allgemeinen Fall versuchten Aktionen kannst du über den Befehl

```
help profilex
```

erhalten. `profilex` ist kein Befehl, sondern nur ein Hilfetext. Die Funktionen des Befehls `PROFILE` sind weiter unten beschrieben.

Es gibt bestimmte Einstellungen von *exaEdit*, die workfile–spezifisch sind, beispielsweise die Breite des Nummernfeldes, die du mit dem Befehl `SKEY` setzt. Du kannst einen solchen Befehl jederzeit in ein *exaEdit*–Profil stellen, damit er beim Start von *exaEdit* automatisch ausgeführt wird. Wenn du jedoch einen weiteren workfile einrichtest, gilt wieder die Voreinstellung für `SKEY`. Um auch in solchen Fällen vom Profil zu profitieren, kannst du im Profil *exaEdit*–Befehlszeilen mit dem Zeichen

```
!
```

beginnen lassen. So markierte Zeilen werden beim Bearbeiten des Profils am Anfang der *exaEdit*–Sitzung wie die Zeilen ohne Ausrufezeichen bearbeitet. Wenn du aber einen neuen workfile anlegst, oder einen workfile mit dem Befehl

```
delete all
```

vollständig leer machst, werden aus dem Profil alle Zeilen, die mit einem Ausrufezeichen beginnen, erneut bearbeitet.

Mit dem Befehl `profile` und geeigneten Parametern kannst du die Befehlszeilen im Profil

- anzeigen
- ausführen
- in den aktiven workfile laden

Dabei kannst du angeben, ob du alle Zeilen meinst oder nur diejenigen, die mit dem Ausrufezeichen beginnen. Einzelheiten, wie du den Befehl `profile` dafür aufrufen musst, findest du durch `help profile` oder bei der ausführlichen Präsentation aller *exaEdit*–Befehle im Abschnitt 3.2.3.

Wie bereits an den entsprechenden Stellen gesagt, werden die Profildateibefehle vor dem Laden der zu editierenden Datei oder beim Start in einen neuen workfile ausgeführt. Manchmal wäre es jedoch vorteilhaft, einen Satz von *exaEdit*–Befehlen zu beliebiger Zeit ausführen zu lassen. Dies ist möglich, jedoch nicht mit einem *exaEdit*–Profil, sondern mit dem *exaEdit*–Befehl `EXEC`.

3.1.27 Online–Hilfen

exaEdit kennt zwei Arten von Online–Hilfe: einmal die kurzen Hilfetexte, die du mit dem *exaEdit*–Befehl

```
help
```

bekommst (siehe Abschnitt 3.2.3, *Die Befehle im einzelnen*), zum andern enthält *exaEdit* Hilfsmittel zur Anzeige der vorliegenden Benutzeranleitung am Bildschirm. Nur von dieser zweiten Hilfeart ist im vorliegenden Abschnitt die Rede.

Die *exaEdit*–Benutzeranleitung gibt es in vier verschiedenen Formaten:

DVI
HTML
PDF
Postscript

Falls du sie benötigst, aber nicht findest, frage diejenige Person, die *exaEdit* auf deinem Rechner installiert hat oder wende dich an den Autor von *exaEdit*. Du findest sie natürlich auch von der *exaEdit*–Startseite im WWW aus.

Da die gesamte Benutzeranleitung von der Größe her nicht in das Programm *exaEdit* passt, handelt es sich um eine separate Datei, die daher nicht notwendig auf jedem Rechner, der *exaEdit* enthält, zur Verfügung steht (beispielsweise wenn vergessen wurde, sie in ein passendes Verzeichnis zu stellen). Ein weiteres Problem ergibt sich aus der Tatsache, dass zum Ansehen der Anleitung am Bildschirm Programme verwendet werden müssen, die ebenfalls nicht zu *exaEdit* gehören, die nicht auf allen Rechnern vorhanden sind und die verschiedene Darstellungsformen der Anleitung benötigen.

exaEdit hat für das Ansehen seiner Benutzeranleitung am Bildschirm den Befehl

```
manual
```

und kennt dann dazu ein externes Programm, eventuell einen Satz von Parametern und eine Datei. Das Programm wird dann mit den Parametern auf die Datei losgelassen.

Da, wie bereits gesagt, verschiedene Programme, Dateien und Parameter möglich sind, versieht *exaEdit* einen solchen zusammengehörigen Satz mit einem Namen.

In *exaEdit* gibt es immer einen (schon einprogrammierten) solcher `manual`–Sätze. Bei seiner Verwendung greifst du auf die Benutzeranleitung zu, die *exaEdit* im WWW zur Verfügung stellt. Bei gewissen Arten der Installation von *exaEdit* wird im Installationsprofil ein weiterer `manual`–Satz bereitgestellt, mit dem eine lokale Kopie der Benutzeranleitung angesprochen wird.

Die so bereitgestellten Aufrufe der Online–Darstellung der Benutzeranleitung kannst du jederzeit im *exaEdit*–Profil oder in der laufenden *exaEdit*–Sitzung verändern oder ergänzen. Mit dem Befehl

```
manual * ?
```

erhältst du eine Liste der gerade vorhandenen `manual`–Sätze. Weitere ausführliche Informationen findest du bei der Beschreibung des Befehls `MANUAL`.

3.1.28 Die Tastatur

exaEdit verwendet keine Tastenkombinationen für Editorfunktionen. Neben den normalen Zeichentasten haben die folgenden Tasten eine Bedeutung:

- Die Cursor–Tasten ($\uparrow \rightarrow \leftarrow \downarrow$):
  Sie bewegen den Cursor durch das Fenster.
- Die `Backspace`–Taste ($\triangleleft$—):
  Sie dient dem Entfernen eines Zeichens.
- Die `Enter`–Taste ($\triangleleft$—$\lrcorner$, `Enter`):
  Sie dient der Eingabe von Befehlen, Präfixbefehlen und Direktänderungen.
- Die Taste `Einfg` (oder `INS` oder â):
  Sie dient zum Ein– und Ausschalten des Einfügemodus.
- Die Taste `Entf` (oder `DEL` oder á):
  Sie dient dem Entfernen eines Zeichens.
- Die Taste `Pos 1` (oder `HOME`):
  Sie dient dem Rückgängigmachen aller Tipp–Änderungen seit dem letzten Drücken der `Enter`–Taste.
- Die Taste `Bild`$\downarrow$:
  Sie dient dem Vorwärtsblättern im workfile um 1 Fensterseite.
- Die Taste `Bild`$\uparrow$:
  Sie dient dem Rückwärtsblättern im workfile um 1 Fensterseite.

- Die Fn–Tasten:

  Von diesen haben nur die folgenden eine voreingestellte Bedeutung:

  - F7: Sie dient dem Rückwärtsblättern im workfile um eine ganze Seite.
  - F8: Sie dient dem Vorwärtsblättern im workfile um eine ganze Seite.
  - F10: Sie dient dem Rückwärtsblättern im workfile um eine halbe Seite.
  - F11: Sie dient dem Vorwärtsblättern im workfile um eine halbe Seite.

- Die Tasten im rechten Tastenblock:

  Sie werden bei manchen Tastaturen als Alternative zu den Fn–Tasten verwendet.

Oftmals gibt es Tastaturen, auf denen die oben genannten Tasten nicht alle vorhanden sind. Manchmal sind sie zwar physisch vorhanden, aber logisch mit solchen Werten versehen, die *exaEdit* nicht erkennen kann oder erst gar nicht erhält. In solchen Fällen kannst du einige der oben bezeichneten Funktionen mit den vorhandenen und funktionierenden F–Tasten verknüpfen.

Beispielsweise ist ein vernünftiges Editieren ohne die Zeicheneinfüge– oder Zeichenentferntaste (Einfg/INS bzw. Entf/DEL) nicht möglich. Wenn nun wenigstens die F–Tasten funktionieren, kannst du mithilfe des Befehls PFK die beiden Tastenfunktionen auf F–Tasten legen. Näheres siehe Abschnitt 3.1.23, *Programmierbare Funktionstasten*, und beim Befehl PFK, Abschnitt 3.2.3, *Die Befehle im einzelnen*. Denke übrigens daran, dass du solche Festlegungen auch in der *exaEdit*–Profildatei (siehe Abschnitt 3.1.26) machen kannst.

Welche Tastenfunktion mit F–Tasten verknüpft werden können, steht im Abschnitt 3.1.30, *exaEdit–Funktionen*.

3.1.29 Tastaturtest

Dieser Abschnitt gilt für die Verhältnisse in Unix–Betriebssystemen, die Informationen zum Befehl KEYBOARD TEST gelten aber auch für Windows–Systeme.

Tastaturen sind eines der trübsten Kapitel in Unix–Betriebssystemen. In dem Bestreben, flexibel auf alle möglichen und unmöglichen Wünsche zu reagieren, wurde die Zuordnung von Tasten zu Zeichen oder Funktionen auf mehreren verschiedenen Ebenen variabel gestaltet. Damit wurde zwar nur die Möglichkeit geschaffen, ein gigantisches Chaos zu produzieren, aber leider wurde dies gelegentlich wenigstens teilweise erreicht.

Wenn du eine Taste der Tastatur drückst und *exaEdit* dann auf eine bestimmte Weise darauf reagiert, gibt es neben den beiden natürlichen Ebenen des Bedeutungswechsels, nämlich Tastaturbeschriftung am einen Ende und Interpretation der Taste durch *exaEdit* am anderen, 7 (= sieben) weitere Ebenen, auf denen sich die Bedeutung aller oder auch nur einiger Tasten verändern lässt.

Damit du dich vergewissern kannst, dass die von dir gedrückten Tasten auch die von dir gewünschte Bedeutung haben, gibt es den Befehl KEYBOARD mit dem Parameter TEST.

Allerdings musst du beachten, dass es Tasten gibt, deren Drücken nicht an *exaEdit* weitergeleitet wird. Wenn es sich dabei um Tasten handelt, von denen du meinst, dass sie *exaEdit* zur Verfügung stehen sollten, musst du dich an die Personen wenden, die für das Betriebssystem deines Rechners zuständig sind.

Der Test einer Taste geht so: Du gibst den Befehl

```
keyboard test
```

(Minimalabkürzung KEYB T) ein, worauf *exaEdit*

```
Drücke Taste:
```

in das Fenster schreibt. Dann drückst du die zu testende Taste. Antwortet *exaEdit* mit einer Zeile, so ist der Test beendet. Siehst du dagegen keine Reaktion von *exaEdit*, so musst du zusätzlich die Enter–Taste drücken. Falls du mehrere Tasten testen möchtest, könnte es vorteilhaft sein, den Befehl KEYBOARD TEST mit X oder Y abzukürzen (siehe die entsprechenden Befehle) oder mittels PFK (siehe dort) auf eine F–Taste zu legen.

Um die Informationen, die KEYBOARD TEST liefert, richtig deuten und gegebenenfalls Verbesserungen an der Tastatur vornehmen zu können, musst du zunächst folgendes wissen:

Im Betriebssystem Unix verwendest du normalerweise kein physisches Terminal, sondern, als sogenannte Terminalemulation, ein logisches Terminal. Welche Terminalemulation du gerade verwendest, steht in der Umgebungsvariablen TERM, die du dir beispielsweise mit dem Unixbefehl `echo $TERM` ansehen kannst.

Jede physisch vorhandene Taste kannst du einzeln oder zusammen mit der Shift–, Alt–, Strg–, ...Taste drücken. Im folgenden bedeutet „Taste“ eine Einzeltaste oder eine solche Kombinationstaste. Die verwendete Terminalemulation legt dann fest, was die Tasten bedeuten. Bitte beachte, dass es Tasten gibt, deren Drücken nicht an die Terminalemulation geht, weil sie schon vorher abgefangen werden.

Die an die Terminalemulation weitergereichten Tasten werden von ihr auf zweierlei Art weitergegeben: Als normale Tasten (auf denen irgendein Zeichen liegt) und als Folgen von Tasten (oft Escape–Folgen oder –Sequenzen genannt, da sie meist mit der Esc–Taste beginnen). Beispiel: Wenn du die mit F1 beschriftete Taste drückst, behauptet die Terminalemulation xterm, du hättest die 5 Tasten

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

in dieser Reihenfolge gedrückt (\E soll die Esc–Taste bedeuten). Andere Terminalemulationen liefern andere Tastenfolgen.

Um diesen Unsinn wieder wettzumachen, gibt es im Unix–Betriebssystem eine sogenannte Terminfo–Datei, in der für jede Terminalemulation, die an dem betreffenden Rechner verwendet wird, angegeben ist, wie die Tastenfolgen wieder zu den Originalinformationen zusammenzusetzen sind. Beispielsweise steht in der Terminfo–Datei für die Terminalemulation xterm, dass die Tastenfolge \E[11~ bedeuten soll, dass die Taste F1 gedrückt wurde.

exaEdit verwendet zum Schreiben in das und zum Lesen vom Fenster den Betriebssystemzusatz Curses (vgl. Abschnitt 3.1.1, *Aufnahme einer exaEdit–Sitzung*). Curses gibt an *exaEdit* diejenigen Informationen weiter, die es von Terminfo erhält. Nun bist du so weit, dass du die von KEYBOARD TEST gelieferten Informationen verstehen kannst. Die Antwort von *exaEdit* hat immer die Gestalt

```
Curses: ..., Zeichen: ..., Escape: ..., Funktion: ...
```

wobei für ... jeweils Einträge gemacht sind. Diese Einträge haben folgende Bedeutung:

Curses: Hier steht der Zahlenwert einer (Zeichen–)Taste oder die Funktion, die Curses von Terminfo mitgeteilt bekommt. Steht nach Curses: aber ein Strich, so bedeutet dies, dass *exaEdit* eine Escape–Folge gelesen hat, die in Terminfo nicht vorkommt, also als Folge von Einzelzeichen von Curses an *exaEdit* durchgereicht wurde. In diesem Fall ist die 3. Rubrik „Escape:“ ausgefüllt.

Zeichen: Hier steht das Zeichen, das mit der Taste verknüpft ist, wenn es ein solches gibt. Ansonsten das Wort „keins“.

Escape: Hier steht die Escape–Folge (vergleiche die Erläuterungen zu „Curses:“), wenn eine solche geschickt wurde.

Funktion: Hier steht das Wort „keine“, wenn es sich um eine Zeichentaste handelt. Handelt es sich um eine andere Taste, so hat sie entweder (meist) eine Funktion in *exaEdit*, die dann genannt wird, oder es wird „?“ ausgegeben.

Falls du mit KEYBOARD TEST arbeiten willst, empfiehlt es sich, den Befehl für einige bekannte und funktionierende Tasten aufzurufen, bevor du dich an die Erkundung unbekannter Tasten wagst.

Die Personen, die für die Betreuung deines Rechners zuständig sind, können die Terminfo–Datei verändern und auf diese Weise fehlende oder falsche Zusammenfassungen von Escape–Folgen zu Funktionstasten ergänzen bzw. verbessern.

3.1.30 *exaEdit*–Funktionen

Wie in Abschnitt 3.1.28, *Die Tastatur*, ausgeführt wurde, gibt es einige Tastenfunktionen, die (mittels Befehl PFK) mit F–Tasten verknüpft werden können. Ursprünglich waren dies alles Funktionen, die auf manchen Tastaturen real als

Tasten vorhanden waren, aber nicht auf allen, so dass der Wunsch nach einem Ersatz entstand. Da in manchen Fällen jedoch der Bezug zu realen Tasten nicht mehr offensichtlich ist, werden diese Funktionen jetzt

exaEdit–Funktionen

genannt.

In der folgenden Tabelle sind die *exaEdit*–Funktionen beschrieben. In der Spalte „Name“ steht die Form, die du im Befehl `PFK` für die Funktion verwenden musst. In der Spalte „Taste“ stehen Beschriftungen von Tasten, die üblicherweise die entsprechende Funktion haben sollten. Stehen in der Spalte `F`–Tasten, so handelt es sich um die Belegungen, die *exaEdit* in jeder neuen *exaEdit*–Sitzung am Anfang macht.

Name	Taste(n)	Funktion
'del'	Entf	Lösche Zeichen unter dem Cursor
'ins'	Einfg	Einfügemodus an oder aus
'+page'	F8, Bild ↓	ganze Seite vorwärts
'-page'	F7, Bild ↑	ganze Seite rückwärts
'+half'	F11	halbe Seite vorwärts
'-half'	F10	halbe Seite rückwärts
'cleft'	←	Cursor nach links
'cright'	→	Cursor nach rechts
'cup'	↑	Cursor nach oben
'cdown'	↓	Cursor nach unten
'pos1'	Pos1, Home	Restauriere Bild nach dem letzten Enter

Mit dem Befehl

```
help function
```

erhältst du einen kurzen Hilfetext in der üblichen Form, so dass du die *exaEdit*–Funktionen auch ohne Benutzeranleitung findest.

3.1.31 Einfügen von Satznummern

Wie in Abschnitt 3.1.9, *Satznummern*, bereits erwähnt, macht *exaEdit* für Sätze, die zwischen zwei vorhandene Sätze kommen, eine Satznummer, die zwischen den beiden Satznummern liegt, damit auf diese Weise die Satznummern immer streng aufsteigend bleiben.

Gehen wir zunächst vom Normalfall aus, dass zwei Sätze Nummern im Abstand von 100 haben, die auch durch 100 teilbar sind, also etwa

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

Wenn jetzt 1 Satz zwischen die beiden Sätze kommt, erhält er die Nummer 820:

```
800 ...
820 neuer Satz
900 ...
```

Folgen nach diesem neuen Satz weitere neue Sätze, so erhalten sie die Nummern `840`, `860` und `880`. Folgen danach noch weitere neue Sätze, so erhalten sie geeignete Nummern zwischen 880 und 900.

Sind die Normalabstände nicht 100, dann sind die ersten Einfügeschritte auch nicht 20, sondern dem Normalabstand angepasst. Auf jeden Fall verfolgt *exaEdit* beim Einfügen diese Ziele:

- Verwende eher „glatte“ Zahlen als „krumme“.
- Belege nicht sofort die Mitte eines Intervalls, sondern bevorzuge zunächst die untere Hälfte. Der Grund ist die Vermutung, dass es eher weitere Sätze nach dem gerade eingesetzten gibt, als vor diesem.

- Greife erst dann zum Neunumerieren, wenn keine Zwischennummer mehr möglich ist.
- Verwende beim erzwungenem Neunumerieren nicht die voreingestellte Schrittweite, sondern eine kleinere, damit nicht alle folgenden alten Sätze neu numeriert werden müssen (also z.B. 20 statt 100 oder 100 statt 500).
- Würden sich beim Neunumerieren Satznummern größer als das mögliche Maximum 99999999 ergeben, so wird zunächst die Schrittweite so verkleinert, bzw. danach der Anfangspunkt des Neunumerierens so weit nach vorne verlegt, dass die größte benötigte Satznummer das Maximum nicht übersteigt (in Version 01 von *exaEdit* erst teilweise realisiert).

3.1.32 *exaEdit*–Fehler

exaEdit ist, wie jedes größere Programm, nur schwer fehlerfrei zu schreiben. Leider muss gegebenenfalls sogar mit unvorhergesehenem Programmabbruch gerechnet werden. Um in einem solchen Fall möglichst viel zu retten, versucht *exaEdit*, den Inhalt aller geänderten workfiles in neue Dateien zu schreiben. Im einzelnen geschieht folgendes:

exaEdit verlässt den Fenstermodus und führt seine restlichen Arbeiten im Zeilenmodus aus. Als erstes kommt eine der Meldungen

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

Dann wird eine Datei `exaEdit.dmp` geöffnet, in die *exaEdit* Informationen schreiben soll, die für die Fehlersuche nützlich sein können. Das Gelingen oder Misslingen führt zu der Meldung

```
exaEdit.dmp [nicht] geöffnet
```

Der Inhalt der Datei `exaEdit.dmp` ist nicht vorhersagbar. Er hängt von der Stelle ab, an der das Programm *exaEdit* abstürzte.

Um die geänderten workfiles zu retten, überprüft *exaEdit* alle workfiles daraufhin, ob du sie geändert, aber die Änderungen noch nicht gesichert hast. Für jeden solchen workfile wird eine Datei mit einem bestimmten Namen geöffnet, in die alle Sätze des workfiles (ohne die top line) hineingeschrieben werden sollen. Das Gelingen oder Misslingen dieses Öffnens führt zu der Meldung

```
exaEdit.jjjj.mm.tt-hh.mm.ss.wfn.dsn [nicht] geöffnet
```

Dabei bedeutet

`jjjj`	das Jahr,
`mm`	den Monat,
`tt`	den Tag,
`hh`	die Stunde,
`mm`	die Minute,
`ss`	die Sekunde,
`wfn`	den Namen des *workfiles*,
`dsn`	den Namen der Datei.

Falls die Sätze des workfiles erfolgreich gerettet werden konnten, erfolgt die Meldung

```
Daten gesichert
```

Nachdem alle workfiles überprüft und gegebenenfalls in Dateien geschrieben wurden, wird die Protokolldatei `exaEdit.dmp` wieder geschlossen (falls sie geöffnet werden konnte), was durch die Meldung

```
exaEdit.dmp geschlossen
```

angezeigt wird.

3.1.33 *exaEdit*–Tests

In *exaEdit* sind einige Mechanismen eingebaut, die es dem *exaEdit*–Entwickler ermöglichen, Informationen über Fehler zu sammeln. Diese Testmöglichkeiten werden mit dem Befehl `TEST` aktiviert oder deaktiviert.

Die Syntax des `TEST`–Befehls ist im Abschnitt 3.2.3, *Die Befehle im einzelnen* (Seite 114), beschrieben. Was die einzelnen Parameter bedeuten, wird jedoch nicht mitgeteilt, da du als *exaEdit*–Benutzer wenig damit anfangen kannst.

3.2 Die Befehle

3.2.1 Notation

Im folgenden werden alle Befehle in alphabetischer Reihenfolge einzeln und im Detail vorgestellt. Die Präfixbefehle sind im Abschnitt 3.3 beschrieben.

Die Beschreibung eines Befehls beginnt links oben mit dem Befehlsnamen und der Syntax des Befehls. In derselben Zeile steht rechts außen die Minimalabkürzung für den Befehl.

Gibt es zu einem Befehl andere mit genau der gleichen Syntax und Bedeutung, so werden diese anderen Befehlsnamen, durch einen senkrechten Strich („|“) abgetrennt, mit aufgeführt. Beispiel

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

Es handelt sich also um den Befehl `BACK`, der identisch ist mit den Befehlen `UP` und `-` (ein Minuszeichen). Die Minimalabkürzung von `BACK` ist `BA`, die von `UP` ist `U`, `-` kann nicht weiter verkürzt werden.

Bei der Beschreibung der Parameter (oder Operanden) eines Befehls wird die folgende Notation verwendet:

eckige Klammern ([ ]): Diese schließen Parameter ein, die du angeben oder weglassen kannst. Im letzteren Fall tritt eine Standardannahme in Kraft, die jeweils genau beschrieben wird.

Beispiel: `BACK [n]` besagt, dass du entweder „`BACK`“ oder „`BACK n`“ angeben kannst. Die Standardannahme ist in diesem Fall 1. Weiteres siehe Beschreibung des Befehls `BACK`.

senkrechter Strich (|): Dieses Zeichen trennt alternative Parameter voneinander ab, von denen du höchstens einen angeben darfst.

Beispiel: `SKEY [n|?]` besagt, dass du entweder „`SKEY n`“ oder „`SKEY ?`“ (oder nur „`SKEY`“), aber nicht „`SKEY n ?`“ eingeben darfst. Weiteres siehe Beschreibung des Befehls `SKEY`.

Kleinbuchstaben: Mit Kleinbuchstaben werden Parameter bezeichnet, für die du Werte einzusetzen hast.

Beispiel: `UP [n]` gibt an, um wieviele Sätze die aktuelle Zeile nach oben zu schieben ist. Bei diesem Befehl musst du also für `n` eine Zahl einsetzen (wenn du einen Parameter angibst).

Großbuchstaben: Mit Großbuchstaben werden Parameter bezeichnet, die du genau so wie beschrieben eingeben musst. Dabei kannst du diese natürlich auch mit Kleinbuchstaben schreiben. Die hier verwendete Unterscheidung dient nur der Information, für welche Parameter Werte einzusetzen sind und welche Parameter unverändert einzutippen sind.

Bitte denke daran, dass Parameter ebenfalls abgekürzt werden können, wenn die Abkürzung eindeutig ist. Die Minimalabkürzung von Parametern wird hier nicht extra ausgewiesen.

3.2.2 Meldungen

Bei jedem Befehl sind im Prinzip alle Fehlermeldungen und Meldungen aufgeführt, die bei seiner Bearbeitung auftreten können. Alle diese Meldungen erscheinen im Stichwortverzeichnis dieser Anleitung in der Form

> `Meldung`, *exaEdit*–Meldung, ...

Für `Meldung` steht natürlich die tatsächliche Meldung.

Zusätzlich gibt es das Kapitel 6, *Die exaEdit–Meldungen*, das sozusagen einen Auszug aus dem Stichwortverzeichnis darstellt, der sich auf die Meldungen beschränkt.

Von den vorstehenden Regeln gibt es Ausnahmen, nämlich Meldungen, die du mit sehr vielen Befehlen erzeugen kannst und deren entsprechend häufige Aufnahme in Stichwortverzeichnis und Meldungskapitel keine wesentliche Information bringen würde. Sie werden im Meldungskapitel ohne Seitenhinweise erläutert, und das Stichwortverzeichnis verweist für sie auf den vorliegenden Abschnitt und auf die entsprechende Seite des Meldungskapitels.

```
Anfangsspalte größer als Endspalte
Datenanfang
Datenende
Die Spalte 0 gibt es nicht
Die top line kannst du nicht ...
Es gibt keinen folgenden (n) Satz
Es gibt keinen vorigen (p) Satz
Parametervariable keine Zeichenkette
Parametervariable nicht definiert
Parametervariable nicht numerisch
SET-Speicher unbenutzt
SET-Speicher ungültig
Zahl zu groß
Zeichenkette zu lang
```

Ebenfalls weggelassen wurden einige Fehlermeldungen, deren Ursache so offensichtlich ist, dass sie vermutlich niemand bei der Beschreibung der Befehle aufsuchen möchte.

```
Anzahl 0 nicht erlaubt
Diesen Befehl gibt es nicht
Fehlerhafter Befehl: ...
Operand fehlt in ...
Parameter fehlt
Ungültiger Parameter
```

3.2.3 Die Befehle im einzelnen

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach unten, zum Ende des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach oben.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die nach dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

> `Datenende`

in die Dialogzone und belässt die aktuelle Zeile.

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach oben, zum Anfang des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach unten.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die (einschließlich der top line) vor dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

```
Datenanfang
```

in die Dialogzone und belässt die aktuelle Zeile.

`_|CALL externbefehl` `_|CAL`

Der Unix– oder Shell– oder MSDOS–Befehl „`externbefehl`“ wird nach außen zur Ausführung weitergegeben.

Ist der externe Befehl fertig, so meldet sich *exaEdit* mit

```
exaEdit: Drücke die Eingabetaste, wenn du alles gesehen hast
```

wenn es vorher im Fenstermodus war. Wenn du die Taste gedrückt hast, befindet sich *exaEdit* wieder im gleichen Zustand wie zuvor.

Falls sich *exaEdit* im Zeilenmodus befand, wird das Beenden des externen Befehls mit

```
exaEdit: Externer Befehl beendet
```

angezeigt.

Es sind alle Befehle möglich, die du auch sonst im Grundzustand der Unix–Sitzung oder in der Windows–Eingabeaufforderung aufrufen kannst.

Möchtest du in Unix einen externen Befehl ablaufen lassen, während du *exaEdit* weiterbenutzt, so kannst du in Unix wie üblich den externen Befehl mit dem Zeichen `&` enden lassen.

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

Der Befehl `&` dient der Definition und der Handhabung von Parametervariablen. Was Parametervariablen sind und was du damit machen kannst, ist in Abschnitt 3.1.25, *Die Parametervariablen* beschrieben.

Der Befehl muss immer mit einem Variablennamen gegeben werden, `&` alleine ist falsch. Für die Parameter `?` und `-` kann anstelle eines Namens ein `*` angegeben werden, mit der Bedeutung, dass alle Parametervariablen gemeint sind.

Gibst du gar keinen Parameter an, so ist das gleichbedeutend mit der Angabe `?`: Die angegebene Parametervariable wird angezeigt. Die Informationen sind

- Der Typ: Einer der Buchstaben `N`, `L` oder `S` für den Typ (Vergleiche Abschnitt 3.1.25.)
- Der Name der Parametervariablen.
- Der Wert der Parametervariablen.

Mit dem Parameter `+NUMBER` definierst du eine neue Parametervariable für numerische Werte. Der Name muss aus 1 bis 8 Buchstaben bestehen. Dabei wird zwischen großen und kleinen Buchstaben unterschieden. Beachte, dass die Namen der vordefinierten Standardparametervariablen einen großen Anfangsbuchstaben und sonst kleine Buchstaben aufweisen. Um der neu definierten Parametervariablen einen Wert zuzuweisen, muss der Befehl `&` erneut mit dem Parameter `=value` aufgerufen werden.

Die Parameter `+STRING` und `+LINE` dienen in gleicher Weise der Definition von Parametervariablen für Zeichenketten bzw. für Satznummern.

Mit dem Parameter `=value` wird einer bereits bestehenden Parametervariablen ein (neuer) Wert zugewiesen. Ein solcher Wert ist entweder eine (zum Typ der Variablen) passende Konstante oder ein bestimmter einfacher Ausdruck.

Erlaubte Ausdrücke für numerische Parametervariablen sind Summe und Differenz von Konstanten. Auch bei Parametervariablen, die Satznummern darstellen, können numerische Konstanten addiert oder subtrahiert werden, was dann in einer neuen Satznummer resultiert, die entsprechend viele Sätze weiter unten oder oben ist. Bei Parametervariablen, die Zeichenketten darstellen, können im Ausdruck mit einem Additionszeichen Zeichenfolgen verkettet werden oder mit der Parameterfolge `SUBSTR /kette/ anfang laenge` eine Teilkette gebildet werden.

Da bei allen *exaEdit*–Befehlen Konstanten durch geeignete Parametervariablen ersetzt werden können, gilt dies auch für den Befehl `&`, sodass auf diese Weise ein relativ mächtiges „Rechnen“ mit Parametern möglich ist.

Mit dem Parameter `-` wird die angegebene Parametervariable gelöscht. Schreibst du einen `*`, so werden alle Parametervariablen gelöscht. Beachte aber, dass die Standardparametervariablen nicht gelöscht werden können.

Ein paar Beispiele (die Satznummern der Datei seien 100, 200, ...):

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

Dann ergibt der Befehl `&*?` die folgende Ausgabe:

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

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

Dieser Befehl verschiebt die betroffenen Sätze horizontal.

Zeilen können explizit angegeben werden: von Zeile `l1` bis Zeile `l2` oder von Zeile `l1` bis zum Datenende. Sie können aber auch implizit angegeben werden: beginnend mit der aktuellen Zeile `n` Zeilen oder alle Zeilen des workfile.

In der ersten Form werden von den betroffenen Zeilen alle diejenigen verschoben, die die Zeichenkette `kette` enthalten. Sie werden so ausgerichtet, dass `kette` jeweils untereinander steht. Das Ergebnis richtet sich nach derjenigen Zeile, in der `kette` am weitesten rechts steht, weil Zeilen alle nach rechts, aber im allgemeinen nicht nach links verschoben werden können. Diejenigen Zeichen, die vor `kette` stehen, bleiben an ihrem Ort. Hast du zum Beispiel die Zeilen

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

so lauten sie nach dem Befehl `ALIGN /[/ ALL`

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

Zusätzlich hast du die Möglichkeit, die Zeilen als ganzes zu verschieben, indem die Zeichen vor `kette` mit verschoben werden, wenn du den Parameter `MOVEALL` angibst. Der Befehl `ALIGN /[/ MOVEALL ALL` hätte also folgendes ergeben:

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

Bei der zweiten Form geschieht die Ausrichtung der betroffenen Zeilen anhand der angegebenen Spalte. Steht in dieser Spalte ein Leerzeichen, so werden ohne Angabe weiterer Parameter alle Zeichen nach dieser Spalte so weit nach links verschoben, dass das erste Nicht-Leerzeichen in diese Spalte gerät. Gibst du aber den Parameter `RIGHT` an, so werden alle Zeichen vor dieser Spalte so weit nach rechts verschoben, dass das erste Nicht-Leerzeichen in diese Spalte gerät. Gibst du bei dieser Verschiebung zusätzlich den Parameter `MOVEALL` an, so wird die Zeile insgesamt nach rechts verschoben, so dass also auch die Zeichen nach der Spalte an der Verschiebung teilnehmen. Wenn du etwa folgende Zeilen hast:

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

so hat der Befehl `ALIGN 7 ALL` das Ergebnis

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

der Befehl `ALIGN 7 RIGHT ALL` das Ergebnis

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

und der Befehl `ALIGN 7 RIGHT MOVEALL` das Ergebnis

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

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach oben, zum Anfang des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach unten.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die (einschließlich der top line) vor dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

```
Datenanfang
```

in die Dialogzone und belässt die aktuelle Zeile.

`BOTTOM` `B`

Der Zeiger auf die aktuelle Zeile wird auf den letzten Satz des workfiles gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach oben.

`CALL|_ externbefehl` `CAL|_`

Der Unix– oder Shell– oder MSDOS–Befehl „`externbefehl`“ wird nach außen zur Ausführung weitergegeben.

Ist der externe Befehl fertig, so meldet sich *exaEdit* mit

```
exaEdit: Drücke die Eingabetaste, wenn du alles gesehen hast
```

wenn es vorher im Fenstermodus war. Wenn du die Taste gedrückt hast, befindet sich *exaEdit* wieder im gleichen Zustand wie zuvor.

Falls sich *exaEdit* im Zeilenmodus befand, wird das Beenden des externen Befehls mit

```
exaEdit: Externer Befehl beendet
```

angezeigt.

Es sind alle Befehle möglich, die du auch sonst im Grundzustand der Unix–Sitzung oder in der Windows–Eingabeaufforderung aufrufen kannst.

Möchtest du in Unix einen externen Befehl ablaufen lassen, während du *exaEdit* weiterbenutzt, so kannst du in Unix wie üblich den externen Befehl mit dem Zeichen `&` enden lassen.

`CASE [?|U|M|L] | [?|I|S]` `CA`

Der Befehl `CASE` wird für zwei verschiedene Zwecke verwendet. Zunächst zu `CASE` mit den Parametern `U`, `M` und `L`.

Normalerweise verwendet *exaEdit* die von dir eingetippten Buchstaben so wie sie sind: kleine Buchstaben bleiben klein, große bleiben groß. Dies entspricht der (Vor–)Einstellung

```
case m      oder      case l
```

(`m` = „mixed“, `l` = „lower“). Stellst du dagegen *exaEdit* mit

```
case u
```

(`u` = „upper“) um, so werden alle eingegebenen Kleinbuchstaben in Großbuchstaben übersetzt (a bis z, nicht die Umlaute).

Alle Zeilen, die du durch Verändern eines Zeichens „anfasst“, werden ebenfalls von klein nach groß übersetzt. Während die Einstellung `U` gilt, wird dies durch ein „`U`“ in der Statuszeile von *exaEdit* angezeigt.

Nun zu `CASE` mit den Parametern `I` und `S`: Durch Eingabe von

```
case i
```

verlangst du, dass die Befehle `CHANGE`, `SSPLIT` und die der `LOCATE`–Familie so ablaufen, als wäre bei ihnen der Parameter `I` angegeben. Dies bedeutet, dass die Befehle groß/klein-insensibel arbeiten, also zwischen großen und kleinen Buchstaben nicht unterscheiden. Durch Eingabe von

```
case s
```

stellst du das Verhalten der genannten Befehle wieder auf die Voreinstellung groß/klein-sensibel um.

```
case ?
```

zeigt die Einstellung beider Schalter durch die Ausgabe je einer Meldung aus den beiden folgenden Meldungspaaren an:

```
Mixed (lower): ohne Übersetzung in Großbuchstaben
Upper: mit Übersetzung in Großbuchstaben (ohne Umlaute)

Groß/klein-sensibel
Groß/klein-insensibel
```

Gibst du den Befehl CASE ohne Parameter ein, so werden beide Schalter wieder in die Ausgangsstellung gebracht: es gilt dann also `case m` und `case s`.

`CCOPY [(l1 [l2])] c1 [c2] COLUMN [+|-]col [LINE [+|-]num][n]` CC

CCOPY steht für column copy, kopiert also Spalten einer Zeile und sollte nicht mit dem Befehl COPY verwechselt werden, der Zeilen kopiert.

```
ccopy 5 column 20
```

kopiert im Satz der aktuellen Zeile die Spalte 5 in die Spalte 20

```
ccopy 5 10 col 20
```

kopiert die Spalten 5 bis 10 in die Spalten 20 bis 25. Beachte, dass bei dem Ziel nur die Anfangsspalte angegeben wird, weil die Anzahl ja bereits bei den Quellspalten festgelegt wurde. Das Kopieren geht so vor sich, dass zunächst die Spalten jenseits des Zielbereichs um den nötigen Betrag nach rechts geschoben werden und dann die Quellspalten in den Zielbereich kopiert werden. Das ist dasselbe Verfahren wie bei dem Befehl COPY, der auch keine Zeilen überschreibt, sondern dazwischen einfügt.

Quell– und Zielbereich dürfen sich nicht überlappen. Falls doch, so erhältst du die Meldung

```
Quell- und Zielbereich überlappen
```

Anstelle einer absoluten Zielspalte kannst du auch eine relative angeben, indem du die Spaltenangabe bei COLUMN mit einem Vorzeichen versiehst:

```
ccopy 5 10 c +15
```

ist dasselbe wie oben.

Bisher beschränkten sich alle Beispiele auf die aktuelle Zeile. Durch Angabe einer Anzahl als letztem Parameter kannst du wie üblich den Befehl in n aufeinanderfolgenden Zeilen ab der aktuellen ausführen lassen:

```
ccopy 5 10 colu 20 7
```

führt den Befehl in 7 Zeilen aus. Hat der workfile zu wenig Zeilen, so erfolgt die Meldung

```
Datenende
```

Mit der beschriebenen Methode beginnt die Ausführung immer in der aktuellen Zeile. Alternativ kannst du die gewünschten Zeilen auch in Klammern als ersten Parameter angeben:

```
ccopy (500 1200) 5 10 c20
```

Dies führt CCOPY in den Zeilen 500 bis 1200 aus, unabhängig davon, wo die aktuelle Zeile ist. Die angegebenen Zeilennummern dürfen auch symbolische sein (t, f, p, *, n, l, b, s). Gibst du nur 1 Zeilennummer an, so wird die zweite gleich der ersten angenommen. Bitte achte auf die Klammern.

Nun zum letzten Parameter von CCOPY. Bisher waren die Zielspalten in der selben Zeile wie die Quellspalten. Du kannst aber auch die Zielzeilen separat angeben:

```
cc5 10c20 line 700
```

Dies kopiert die Spalte 5 bis 10 der aktuellen Zeile in die Spalten 20 bis 25 der Zeile 700.

Wie bei COLUMN kannst du auch bei LINE durch Voransetzens eines Vorzeichens eine relative Zielzeile angeben:

```
cc5 10c20 l-6
```

Gibt es die angegebene Zeile nicht, so erhältst du die Meldung

```
Zielsatz nicht gefunden
```

Selbstverständlich kannst du auch noch die (beiden alternativen) Angaben über die Anzahl der betroffenen Zeilen hinzufügen:

```
ccopy (500 b) 8 column 9 line 000400
```

Bitte beachte, dass du keine Zielzeile angeben kannst, die es (noch) nicht gibt. Wenn du also etwas hinter die letzte workfile–Zeile kopieren willst, musst du die Zielzeile vorher (leer) erstellen.

Mit CCOPY lassen sich ganze Rechtecke des workfiles an eine andere Stelle kopieren.

`CDELETE [(l1[l2])] c1 c2 [n|ALL]` CD

CDELETE steht für `column delete`, löscht also Spalten einer Zeile.

```
cdelete 5 5
```

löscht die Spalte 5 der aktuellen Zeile,

```
cdelete 5 10
```

Löscht die Spalten 5 bis 10 der aktuellen Zeile. Das Löschen geht so vor sich, dass die Zeichen rechts vom Löschbereich nach links in die entstehende Lücke geschoben werden. Beachte, dass du auch beim Löschen nur einer Spalte Anfangs– und Endspalte des Löschbereichs angeben musst.

Durch Angabe einer Anzahl `n` als letztem Parameter verlangst du das Spaltenlöschen in `n` Zeilen ab der aktuellen. Hat der workfile nicht genügend Sätze, so endet der Befehl mit der Meldung

```
Datenende
```

Anstelle einer Anzahl `n` kannst du auch den Parameter `ALL` angeben, der das Spaltenlöschen in allen Sätzen des workfile veranlasst.

Anstelle einer Anzahl im letzten Parameter kannst du den Zeilenbereich, in welchem CDELETE auszuführen ist, auch in Klammern als ersten Parameter angeben. Du kannst explizite und symbolische Zeilennummern angeben:

```
cd (f 1200) 5 10
```

Damit werden von der ersten Zeile bis zu der mit der Nummer 1200, die Grenzen jeweils eingeschlossen, die Spalten 5 bis 10 gelöscht.

Die Angabe der Zeilennummern und die Angabe der Zeilenanzahl schließen sich gegenseitig aus.

Gibst du in der Zeilenklammer nur 1 Zeile an, so wird die zweite gleich der ersten angenommen.

`CHANGE [sp1 [sp2]] /alt[/[neu[/ [n][A][D][H][I]]]]` C

Verwandter Befehl: `REPLACE`

In der aktuellen Zeile wird die Zeichenkette „`alt`“ durch die Zeichenkette „`neu`“ ersetzt.

Die Grundform des `CHANGE`–Befehls ist

```
CHANGE /alt/neu/
```

Dabei musst du für „`alt`“ und „`neu`“ die gewünschten Zeichenketten einsetzen. Die Zeichenketten sind dabei durch einen Kettenbegrenzer einzurahmen, der hier als „/“ wiedergegeben wird. Als Kettenbegrenzer eignet sich jedes Zeichen außer Ziffern und außer dem Leerzeichen. Als praktisch hat sich erwiesen, das rechts unten auf der Tastatur befindliche Sonderzeichen zu nehmen, und nur, wenn dieses in der alten oder neuen Zeichenkette vorkommt, ein anderes zu wählen.

Abkürzende Schreibweisen:

```
CHANGE /alt/neu
```

kannst du schreiben, wenn du keinen der nachfolgenden Parameter benötigst.

```
CHANGE /alt/
```

kannst du schreiben, wenn du die alte Zeichenkette durch nichts ersetzen, also löschen willst, und keine weiteren Parameter benötigt werden. Ist letzteres nicht der Fall, musst du `CHANGE /alt//...` schreiben.

```
CHANGE /alt
```

ist eine weitere Verkürzung von `CHANGE /alt/`.

Einschränkung auf Spalten:

Normalerweise darf sich die zu ändernde Zeichenkette an beliebiger Stelle des Satzes befinden. Dies kannst du jedoch etwa mit dem Befehl `ZONE` (siehe dort) einschränken. Eine Beschränkung des Suchbereichs kannst du aber auch direkt im `CHANGE`–Befehl vornehmen, indem du die Spalten angibst. Zum Beispiel ändert

```
CHANGE 10 20 /alt/neu/
```

nur dann, wenn die Zeichenkette „`alt`“ zwischen den Spalten 10 und 20, beide eingeschlossen, vorkommt. Bei dieser Angabe werden die Grenzen des `ZONE`–Bereichs ignoriert. Gibst du nur 1 Spalte an, so erstreckt sich der mögliche Änderungsbereich von der angegebenen Spalte bis zum jeweiligen Satzende.

Erweiterung auf mehrere Zeilen (Sätze):

Ohne weitere Angabe wird nur in der aktuellen Zeile geändert. Durch die Angabe einer Zeilenzahl `n`, etwa

```
CHANGE ... /alt/neu/ n
```

verlangst du, dass die aktuelle und die `n-1` folgenden Zeilen durchsucht werden und dort gegebenenfalls die Änderung gemacht wird.

Wird die Zeichenkette „`alt`“ in den angegebenen Zeilen nicht gefunden, so bleibt die aktuelle Zeile bestehen, und es erfolgt die Meldung

```
Zeichenkette nicht gefunden: alt
```

Wird sie dagegen `m`–mal gefunden, so rutscht die aktuelle Zeile um `n` Zeilen nach unten (also die Daten im sichtbaren Fenster um `n` Zeilen nach oben), und es erfolgt die Meldung

```
m mal geändert
```

Normalerweise wird die zu ändernde Zeichenkette in jedem Satz nur 1–mal gesucht. Kommt sie in einem Satz mehrfach vor, wird nur die am weitesten links stehende geändert. Durch Angabe des zusätzlichen Parameters

```
A
```

verlangst du aber, dass alle Vorkommnisse in der Zeile geändert werden.

Durch Angabe des Parameters

```
D
```

(= „display“) verlangst du, dass alle geänderten Zeilen in der Dialogzone angezeigt werden.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal editieren. Zu diesem Zweck müssen die (ein oder zwei) angegebenen Zeichenketten hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um in einer Datei alle Carriage–Return–Zeichen (hexadezimal `0d`) zu beseitigen, gibst du, am Anfang des workfiles stehend, den Befehl

```
change /0d// 999 h
```

(Hier wurde vorausgesetzt, dass die Datei weniger als 1000 Zeilen hat und dass `0d` in jedem Satz nur 1–mal vorkommt.)

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Suchen nach der zu ändernden Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise kann

```
change /ab/12/ i
```

`ab` oder `Ab` oder `aB` oder `AB` ändern.

Wenn du den Parameter `I` öfters benutzen musst, kannst du auch mit dem Befehl `CASE` erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter `CASE`.

Die Parameter `H` und `I` darfst du nicht gleichzeitig angeben. Tust du es doch, so wird der Parameter `I` mit der Meldung

```
I wird ignoriert, da H angegeben
```

ignoriert, der Befehl aber weiter bearbeitet.

`CMDSEP [?|x]` `CMD`

Normalerweise kannst du mehrere *exaEdit*–Befehle auf einmal abgeben, indem du sie mit einem Semikolon („;“) voneinander trennst. Mit dem Befehl `CMDSEP` kannst du nun ein anderes Zeichen zum Trennzeichen machen (`CMDSEP x`) oder das Trennzeichen abfragen (`CMDSEP ?`).

Gibst du nur „`CMDSEP`“ an (also ohne Parameter), so stellt dies die Voreinstellung wieder her; dies ist gleichwertig mit der Eingabe des Befehls

```
CMD ;
```

Als Trennzeichen sind alle Zeichen außer dem Fragezeichen („?“) und den Buchstaben erlaubt.

Da du bei intensivem Gebrauch von *exaEdit* das Trennzeichen vermutlich recht häufig benötigst, sollte es nach Möglichkeit auf einer leicht erreichbaren Taste liegen, die ohne Umschaltung (Taste ⇧) benutzt wird. Es empfiehlt sich also, bei der Verwendung einer Tastatur, bei der das Semikolon nur über die Umschalttaste erreicht werden kann, ein anderes Trennzeichen, zum Beispiel das Komma, zu verwenden:

```
CMD ,
```

Eventuell solltest du die Abänderung des Trennzeichens in deine *exaEdit*–Profildatei aufnehmen.

`CMOVE [(l1[l2])] c1[c2] COLUMN [+|-]col [LINE [+|-]num] [n]` CM

CMOVE steht für column move, verschiebt also Spalten einer Zeile und sollte nicht mit dem Befehl MOVE verwechselt werden, der Zeilen verschiebt.

```
cmove 5 column 20
```

verschiebt im Satz der aktuellen Zeile die Spalte 5 in die Spalte 20

```
cmove 5 10 col 20
```

verschiebt die Spalten 5 bis 10 in die Spalten 20 bis 25. Beachte, dass beim Ziel nur die Anfangsspalte angegeben wird, weil die Anzahl ja bereits bei den Quellspalten festgelegt wurde.

Das Verschieben kannst du dir so vorstellen: Die Zeichen ab der Zielspalte werden um die Anzahl der zu verschiebenden Zeichen nach rechts verschoben. In die aufgegangene Lücke werden die zu verschiebenden Zeichen geschrieben, die am Originalplatz dann gelöscht werden. Die sich dadurch ergebende Lücke wird durch Linksverschiebung der Zeichen rechts von dieser Lücke wieder geschlossen. Im Beispiel `cmove 5 10 col 20` befinden sich die zu verschiebenden Zeichen danach in Spalte 14 bis 19, weil sie „zwischen“ die ursprünglichen Spalten 19 und 20 verschoben wurden. Das ist dasselbe Verfahren wie bei dem Befehl MOVE, der auch keine Zeilen überschreibt, sondern dazwischen einfügt und an der Verschiebungsquelle keine Lücke hinterlässt.

Quell– und Zielbereich dürfen sich nicht überlappen. Falls doch, so erhältst du die Meldung

```
Quell- und Zielbereich überlappen
```

Anstelle einer absoluten Zielspalte kannst du auch eine relative angeben, indem du die Spaltenangabe bei COLUMN mit einem Vorzeichen versiehst:

```
cmove 5 10 c +15
```

ist dasselbe wie oben.

Bisher beschränkten sich alle Beispiele auf die aktuelle Zeile. Durch Angabe einer Anzahl als letztem Parameter kannst du wie üblich den Befehl in n aufeinanderfolgenden Zeilen ab der aktuellen ausführen lassen:

```
cmove 5 10 colu 20 7
```

führt den Befehl in 7 Zeilen aus. Hat der workfile zu wenig Zeilen, so erfolgt die Meldung

```
Datenende
```

Mit der beschriebenen Methode beginnt die Ausführung immer in der aktuellen Zeile. Alternativ kannst du die gewünschten Zeilen auch in Klammern als ersten Parameter angeben:

```
cmove (500 1200) 5 10 c20
```

Dies führt CMOVE in den Zeilen 500 bis 1200 aus, unabhängig davon, wo die aktuelle Zeile ist. Die angegebenen Zeilennummern dürfen auch symbolische sein (t, f, p, *, n, l, b, s). Gibst du nur 1 Zeilennummer an, so wird die zweite gleich der ersten angenommen. Bitte achte auf die Klammern.

Nun zum letzten Parameter von CMOVE. Bisher waren die Zielspalten in der selben Zeile wie die Quellspalten. Du kannst aber auch die Zielspalten separat angeben:

```
cm5 10c20 line 700
```

Dies verschiebt die Spalte 5 bis 10 der aktuellen Zeile in die Spalten 20 bis 25 der Zeile 700.

Wie bei COLUMN kannst du auch bei LINE durch Voransetzen eines Vorzeichens eine relative Zielzeile angeben:

```
cm5 10c20 l-6
```

Gibt es die angegebene Zeile nicht, so erhältst du die Meldung

```
Zielsatz nicht gefunden
```

Selbstverständlich kannst du auch noch die (beiden alternativen) Angaben über die Anzahl der betroffenen Zeilen hinzufügen:

```
cmove (500 b) 8 column 9 line 000400
```

Bitte beachte, dass du keine Zielzeile angeben kannst, die es (noch) nicht gibt. Wenn du also etwas hinter die letzte workfile–Zeile verschieben willst, musst du die Zielzeile vorher (leer) erstellen.

Mit CMOVE lassen sich ganze Rechtecke des workfiles an eine andere Stelle verschieben.

`CODEPAGE [?|DOS|WIN]` COD

In den 32Bit–Windowssystemen gibt es verschiedene Darstellungen für die deutschen Sonderzeichen.

Die Darstellung, die vorzugsweise von „echten“ Windowsprogrammen verwendet wird (Beispiel `notepad`), verwendet *exaEdit* unter `WIN`. Die Darstellung, die vorzugsweise von `DOS`–Programmen verwendet wird (Beispiel `type`), verwendet *exaEdit* unter `DOS`.

Die Voreinstellung des Schalters ist `WIN`. Mit dem Parameter „?“ kannst du die Schalterstellung abfragen. Die entsprechenden Meldungen sind

```
DOS      bzw.      WIN
```

Rufst du `CODEPAGE` unter Unix auf, erhältst du die Meldung

```
CODEPAGE ist nur für Windows-Systeme
```

`COMPRESS [?|#n|n|ALL]` COM

Umgekehrter Befehl: `EXPAND`

Dieser Befehl komprimiert Sätze, indem geeignete Folgen von Leerzeichen durch Tabulatorzeichen ersetzt werden. Ein Tabulatorzeichen bedeutet: Setze von hier bis zum nächsten Tabulatorstop Leerzeichen zwischen das vorige und das nächste Zeichen. Die Tabulatorstops sind die Spalten 1, 9, 17, 25, ... Enthält eine Zeile eine Folge von Leerzeichen, in der ein Tabulatorstop liegt, so wird das 1. Leerzeichen durch ein Tabulatorzeichen (`x09`) ersetzt und die restlichen Zeichen der Zeile ab dem Tabulatorstop nach links bis zum Tabulatorzeichen verschoben. Dies wird für die Zeile solange wiederholt, wie es geht.

Ohne weitere Angabe wird nur die aktuelle Zeile komprimiert. Du kannst aber auch die Anzahl `n` der (ab der aktuellen) zu komprimierenden Zeilen angeben oder mit `ALL` verlangen, dass alle Sätze des workfiles komprimiert werden.

Da solche Komprimierungen oft nicht besonders sinnvoll sind, wenn kurze Folgen von Leerzeichen komprimiert werden, komprimiert *exaEdit* gemäß der Voreinstellung nur Folgen von mindestens 4 Leerzeichen. Diese Anzahl kannst du mit dem Befehl

```
compress #n
```

entsprechend verändern. Der Befehl

```
compress ?
```

zeigt die gültige Einstellung. Die Antwort ist die Meldung

```
compress #...
```

Du kannst sie verwenden, um den Wert zu verändern.

Falls es aufgrund des `COMPRESS`–Befehls zu Komprimierungen gekommen ist, erstattet *exaEdit* eine Erfolgsmeldung

```
n mal in m Sätzen um k Leerzeichen komprimiert
```

Ist m = 1, so wird „Sätzen“ durch „Satz“ ersetzt.

Gibt es nichts zu komprimieren, so bleibt die aktuelle Zeile stehen. Im Normallfall stellt *exaEdit* die aktuelle Zeile auf den zuletzt komprimierten Satz. Hast du so viele Sätze angegeben, dass *exaEdit* bis über das workfile–Ende hinaus gehen wollte (Meldung Datenende), so ist der letzte workfile–Satz die aktuelle Zeile. Hast du den Parameter ALL verwendet, so bleibt die aktuelle Zeile erhalten, egal ob komprimiert wurde oder nicht.

CONCAT [/kette/|n] CON

Umgekehrter Befehl: SSPLIT

Mit diesem Befehl verkettest du den Satz der aktuellen Zeile mit dem folgenden Satz, der damit als selbständiger Satz verschwindet.

Gibst du keinen Parameter an, so wird der anzuhängende Satz unmittelbar hinter das letzte Nicht–Leerzeichen der aktuellen Zeile gesetzt.

Gibst du als Parameter eine Zeichenkette an, so wird diese zwischen die beiden zusammenzuhängenden Sätze gesetzt. Eine häufige Anwendung ist es zum Beispiel, ein Leerzeichen zwischen die beiden Teile zu setzen.

Anstelle einer Zeichenkette kannst du auch eine Spalte angeben. Diese bezeichnet die Stelle im Satz der aktuellen Zeile, an der der Inhalt der Folgezeile eingesetzt werden soll. Befindet sich die Spalte nach dem letztem Nicht–Leerzeichen, so werden entsprechend Leerzeichen eingesetzt. Befindet sich die Spalte vor dem Ende der aktuellen Zeile, so überschreiben die Zeichen des folgenden Satzes die entsprechenden Zeichen der aktuellen Zeile. Ist in diesem Fall die „anzuhängende“ Zeile kurz genug, so bleiben die restlichen Zeichen der aktuellen Zeile erhalten. CONCAT verhält sich hier ähnlich wie der Befehl REPLACE, nur dass die einzusetzenden Zeichen nicht im Befehl anzugeben sind, sondern vom Folgesatz genommen werden.

Als Beispiel sollen die folgenden beiden Zeilen dienen, die erste ist die aktuelle Zeile:

```
Mit freundlichen
Grüßen
```

Befehl	Ergebnis
concat	Mit freundlichenGrüßen
concat / /	Mit freundlichen Grüßen
concat 20	Mit freundlichen Grüßen
concat 5	Mit Grüßenlichen

Ist beim Aufruf von CONCAT die Zeile nach der aktuellen leer, so wirkt der Befehl

```
CONCAT /kette/
```

wie das Verlängern der aktuellen Zeile um die Zeichen „kette“. (Mit dem Befehl SSPLIT (siehe dort) kannst du übrigens eine Leerzeile erzeugen, wenn du den beschriebenen Effekt zum Verlängern eines Satzes verwenden möchtest.)

`COPY [num1[num2][wfname]]` `CO`

Der Befehl `COPY` kopiert einen oder mehrere Sätze hinter den Satz der aktuellen Zeile des workfiles. Die zu kopierenden Sätze werden durch ihre Nummer oder eine symbolische Nummer bezeichnet.

Die Satznummer kann ohne führende Nullen angegeben werden. Symbolische Satz- bzw. Zeilennummern sind:

- `*` für die aktuelle Zeile
- `p` (= previous) für die Zeile vor der aktuellen Zeile
- `n` (= next) für die Zeile nach der aktuellen Zeile
- `f` (= first) für die erste Zeile des workfiles
- `l` (= last) für die letzte Zeile des workfiles
- `t` (= top) für die top line des workfiles
- `b` (= bottom) für die letzte Zeile des workfiles
- `s` (= set) für die mit dem Befehl `SET` markierte Zeile

Gibst du nur 1 Nummer an, so wird nur dieser Satz kopiert; gibst du 2 Nummern an, so werden alle Sätze von der ersten bis zur zweiten Nummer kopiert. Zu diesem Zweck muss natürlich `num1` $\leq$ `num2` gelten. Ist dies nicht der Fall, so erfolgt die Meldung

```
Erste Nummer größer als zweite
```

Gibst du eine Nummer an, die es im workfile nicht gibt, so erfolgt die Meldung

```
Nummer ... nicht gefunden
```

wobei für ... die Nummer eingesetzt wird.

Außerdem musst du darauf achten, dass der Kopierbereich nicht das Kopierziel enthält: Das Kopierziel ist sozusagen die Stelle zwischen aktuellem und dem darauffolgenden Satz. Der Kopierbereich darf also nicht die aktuelle Zeile und gleichzeitig die darauffolgende enthalten, wie es zum Beispiel bei `COPY * num2` der Fall ist, wenn `num2` hinter der aktuellen Zeile liegt. Bei Verstoß gegen diese Bedingung erhältst du die Meldung

```
Ziel im COPY-Bereich
```

Einige Beispiele:

- `co *` verdoppelt die aktuelle Zeile.
- `co b` kopiert die letzte Zeile.
- `co f l` kopiert die gesamten Daten hinter die aktuelle Zeile (geht nur, wenn die aktuelle Zeile die letzte Zeile oder die top line ist).
- `co500*` kopiert von Zeile 500 bis zur aktuellen Zeile.
- `co p *` kopiert von der Zeile vor der aktuellen bis zur aktuellen, verdoppelt also 2 Sätze an der Stelle der aktuellen Zeile, macht also aus den beiden Zeilen `a b` die 4 Zeilen `a b a b`, wenn `b` die aktuelle Zeile war.

Ohne Angabe eines 3. Parameters werden die zu kopierenden Sätze aus demselben workfile genommen, in den sie eingesetzt werden sollen. Du kannst aber auch als 3. Parameter einen workfile angeben, aus dem die Sätze zum Kopieren geholt werden sollen. Gibt es diesen nicht, so erhältst du die Meldung

```
Workfile nicht gefunden
```

Beispielsweise kopiert

```
copy 1000 1500 abc
```

die Sätze mit den Nummern `1000` bis `1500` aus dem workfile `abc` hinter die aktuelle Zeile des aktiven workfiles.

Ein weiteres Beispiel: Du möchtest den gesamten Inhalt des Hauptworkfiles (`main`) in einem neuen workfile namens `abc` haben. Dann geht das am einfachsten mit

```
wf abc;co f l main
```

Beachte bitte, dass `COPY T B` die top line mitkopiert, was meistens nicht erwünscht sein dürfte.

Bei der Angabe des workfile–Namens musst du den 2. Parameter (die 2. Nummer) angeben (auch wenn er mit dem 1. identisch ist), wenn der workfile–Name mit einer symbolischen Zeilennummer verwechselt werden könnte. Beispielsweise kopiert:

```
co 500 a
```

den Satz mit der Nummer 500 aus dem workfile a, während

```
co 500b
```

alle Sätze im aktiven workfile ab der Nummer 500 meint, so dass du bei Kopieren des Satzes 500 aus dem workfile b

```
co500 500b
```

schreiben musst.

Die beim Kopieren eingesetzten Sätze erhalten Nummern, die sich an die vorhandenen Nummern der beiden Sätze, zwischen die eingesetzt wird, möglichst gut anpassen. Da das Verfahren, wie die neuen Nummern bestimmt werden, auch bei anderen Befehlen benutzt wird, ist es nur einmal beschrieben und zwar im Abschnitt 3.1.31, *Einfügen von Satznummern*.

`COUNT num1 [num2]` COU

Der Befehl COUNT zählt Sätze im workfile. Die Parameter sind die Satznummern, zwischen denen die Sätze gezählt werden sollen. Die beiden Grenzsätze werden mitgezählt.

Die Satznummer(n) kannst du ohne führende Nullen eingeben. Satz- bzw. Zeilennummern dürfen auch symbolische sein:

- `*` für die aktuelle Zeile
- `p` (= previous) für die Zeile vor der aktuellen Zeile
- `n` (= next) für die Zeile nach der aktuellen Zeile
- `f` (= first) für die erste Zeile des workfiles
- `l` (= last) für die letzte Zeile des workfiles
- `t` (= top) für die top line des workfiles
- `b` (= bottom) für die letzte Zeile des workfiles
- `s` (= set) für die mit dem Befehl SET markierte Zeile

Wenn du COUNT nur mit 1 Parameter aufrufst, ergänzt *exaEdit* als 2. Parameter den Satz der aktuellen Zeile.

Als Ergebnis schreibt *exaEdit* die gefundene Anzahl in die Dialogzone. Ist die erste Satznummer größer als die zweite, so wird das Ergebnis negativ, also mit einem vorangehenden Minuszeichen, ausgewiesen.

Das Ergebnis der Zählung wird außerdem in die Parametervariable `&Count` geschrieben.

`DELETE [n|ALL]` DE

Verwandte Befehle: DELETEL und Präfixbefehl DELETE.

Der Befehl DELETE löscht den Satz der aktuellen Zeile und die `n - 1` folgenden Sätze, falls n angegeben wird.

Bei der Angabe von ALL werden sämtliche Sätze des workfiles gelöscht.

Nach dem Löschen steht der letzte Satz vor den gelöschten Sätzen in der aktuellen Zeile.

`DELETEL|DL num1 [num2]` `DL`

Verwandte Befehle: `DELETE` und Präfixbefehl `DELETE`.

Der Befehl `DELETEL` (seine Abkürzung ist nur `DL`) löscht bei der Angabe von nur einem Parameter den Satz mit der angegebenen Nummer, bei der Angabe von beiden Parametern die Sätze von der Nummer `num1` bis zur Nummer `num2` (jeweils einschließlich).

Die Satznummer(n) können ohne führende Nullen angegeben werden.

Als Satznummern sind auch symbolische Satz- bzw. Zeilennummern erlaubt. Diese können sein:

- `*` für die aktuelle Zeile
- `p` (= previous) für die Zeile vor der aktuellen Zeile
- `n` (= next) für die Zeile nach der aktuellen Zeile
- `f` (= first) für die erste Zeile des workfiles
- `l` (= last) für die letzte Zeile des workfiles
- `t` (= top) für die top line des workfiles
- `b` (= bottom) für die letzte Zeile des workfiles
- `s` (= set) für die mit dem Befehl `SET` markierte Zeile

Beispiel:

```
dl f l
```

löscht alle Sätze des workfiles und ist daher mit `DELETE ALL` identisch.

Nach dem Löschen steht der letzte Satz vor den gelöschten Sätzen in der aktuellen Zeile.

`DISPLAY [n|ALL]` `D`

Der Befehl `DISPLAY` zeigt den Satz der aktuellen Zeile in der Dialogzone.

Gibst du zusätzlich eine Zahl `n` oder den Parameter `ALL` an, so werden `n` Sätze (ab der aktuellen Zeile) oder alle Sätze des workfiles aufgelistet.

Die Hauptanwendung dieses Befehls liegt im Zeilenmodus von *exaEdit*. In diesem werden keine Sätze des workfiles im Fenster gezeigt, wenn du es nicht mit dem Befehl `DISPLAY` verlangst.

Daneben kannst du `DISPLAY` auch im (normalen) Fenstermodus verwenden, um zum Beispiel eine Kopie des Satzes der aktuellen Zeile in der Dialogzone zu erhalten, etwa, um sie für den Nummerbefehl (Ändern einer Zeile, Erstellen einer neuen aus einer alten) zu verwenden.

Beachte bitte, dass von einem Satz, der in der Datenzone mehrere Zeilen zur Darstellung benötigt, bei `DISPLAY` nur die erste Zeile gezeigt wird.

`DL|DELETEL num1 [num2]` `DL`

Verwandte Befehle: `DELETE` und Präfixbefehl `DELETE`.

Der Befehl `DL` (das ist die einzige Abkürzung für `DELETEL`) löscht bei der Angabe von nur einem Parameter den Satz mit der angegebenen Nummer, bei der Angabe von beiden Parametern die Sätze von der Nummer `num1` bis zur Nummer `num2` (jeweils einschließlich).

Die Satznummer(n) können ohne führende Nullen angegeben werden.

Als Satznummern sind auch symbolische Satz- bzw. Zeilennummern erlaubt. Diese können sein:

- `*` für die aktuelle Zeile
- `p` (= previous) für die Zeile vor der aktuellen Zeile
- `n` (= next) für die Zeile nach der aktuellen Zeile
- `f` (= first) für die erste Zeile des workfiles
- `l` (= last) für die letzte Zeile des workfiles
- `t` (= top) für die top line des workfiles
- `b` (= bottom) für die letzte Zeile des workfiles
- `s` (= set) für die mit dem Befehl `SET` markierte Zeile

Beispiel:

```
dl f l
```

löscht alle Sätze des workfiles und ist daher mit `DELETE ALL` identisch.

Nach dem Löschen steht der letzte Satz vor den gelöschten Sätzen in der aktuellen Zeile.

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach unten, zum Ende des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach oben.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die nach dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

```
Datenende
```

in die Dialogzone und belässt die aktuelle Zeile.

`END | QUIT` `E | Q`

Der Befehl `END` beendet den Editor. Vorher wird festgestellt, ob du in den vorhandenen workfiles Änderungen vorgenommen hast. Falls ja, so wirst du darauf hingewiesen und erhältst die Möglichkeit, die *exaEdit*–Sitzung weiterzuführen, beispielsweise, um die geänderten workfiles auf einen Datenträger zu schreiben.

Gibt es in der *exaEdit*–Sitzung nur den workfile `MAIN`, so erhältst du die Meldung

```
Änderungen nicht gesichert
Drücke J oder Y, um aufzuhören:
```

Im Zeilenmodus lautet die zweite Zeile

```
Gib J oder Y ein, um aufzuhören
```

Gibt es noch andere workfiles, so lautet die erste Zeile der Meldung statt dessen

```
Nicht gesicherte workfiles: ...
```

wobei anstelle von ... alle betroffenen workfiles aufgeführt sind.

Falls du dich entschließt, die Änderungen nicht zu sichern, drückst du einfach die Taste „`j`“ oder „`y`“ (klein oder groß), und der Editor beendet sofort seine Arbeit. Drückst du dagegen irgendeine andere Taste, ist die Bearbeitung des Befehls `END` beendet, und du kannst ganz normal weitereditieren.

Bitte beachte, dass beim Beantworten der Frage nach dem Aufhören das bloße Drücken einer Taste genügt: anschließendes Drücken der `Enter`–Taste ist nicht erforderlich.

EXEC EX

Der Befehl EXEC setzt voraus, dass es einen workfile namens EXEC gibt. In diesem workfile müssen lauter *exaEdit*–Befehle stehen.

Wird der Befehl EXEC aufgerufen, so werden in demjenigen workfile, in welchem der Aufruf erfolgt, die im workfile EXEC stehenden *exaEdit*–Befehle ausgeführt.

Gibt es den workfile EXEC nicht, so erhältst du die Meldung

```
Workfile nicht gefunden
```

Nach dem Befehl EXEC in einer Befehlszeile darf nichts weiter stehen.

Die Befehlszeilen im workfile EXEC dürfen höchstens so lang sein wie die Fensterbreite in demjenigen workfile, in welchem der EXEC–Befehl gegeben wird. Bei einem Verstoß erhältst du die Meldung

```
n. EXEC-Zeile länger als Fensterbreite ...
```

worin für n die absolute Zeilennummer und für die Punkte die aktuelle Fensterbreite steht.

Du kannst EXEC zum Beispiel dafür verwenden, immer wieder benötigte Belegungen von F–Tasten als *exaEdit*–Befehle (vergleiche PFK) in einer Datei abzuspeichern, sie bei Bedarf in einen workfile namens EXEC zu laden und sie dann mittels entsprechender EXEC–Befehle für alle workfiles deiner *exaEdit*–Sitzung wirksam zu machen. Dieses Verfahren ist eine Alternative zur Verwendung des *exaEdit*–Profils.

Sämtliche *exaEdit*–Befehle werden unter EXEC so ausgeführt wie sonst auch, mit einer Ausnahme: Beim Befehl FILE entfallen die sonst üblichen Rückfragen der Art

```
Alte Datei, drücke J oder Y, um sie zu ersetzen
```

usw. Besondere Vorsicht ist also bei FILE unter EXEC angezeigt.

EXPAND [n|ALL] EXP

Umgekehrter Befehl: COMPRESS

Dieser Befehl löst Tabulatorzeichen in den Daten auf. Gibst du keine Parameter an, so wird der Satz der aktuellen Zeile bearbeitet. Die Angabe n besagt, dass, beginnend mit dem Satz der aktuellen Zeile, n Sätze bearbeitet werden sollen. Gibst du ALL an, so werden alle Sätze des workfiles bearbeitet.

exaEdit nimmt derzeit an, dass sich Tabulatorstops an den Spalten 9, 17, 25, ... befinden; eine Änderung ist nicht möglich.

Das Expandieren eines Satzes geht so: Wird, von links beginnend, ein Tabulatorzeichen gefunden (hexadezimaler Wert x09), so wird es durch ein Leerzeichen ersetzt, der nach dem Tabulatorzeichen befindliche Text zum nächsten Tabulatorstop nach rechts verschoben und die eventuell entstehende Lücke mit Leerzeichen aufgefüllt. Dies geschieht für alle Tabulatorzeichen des Satzes.

Falls es aufgrund des EXPAND–Befehls zu Expansionen gekommen ist, erstattet *exaEdit* eine Erfolgsmeldung

```
n mal in m Sätzen um k Leerzeichen expandiert
```

Ist m = 1, so wird „Sätzen“ durch „Satz“ ersetzt.

Gibt es nichts zu expandieren, so bleibt die aktuelle Zeile stehen. Im Normalfall stellt *exaEdit* die aktuelle Zeile auf den zuletzt expandierten Satz. Hast du so viele Sätze angegeben, dass *exaEdit* bis über das workfile–Ende hinaus gehen wollte (Meldung Datenende), so ist der letzte workfile–Satz die aktuelle Zeile. Hast du den Parameter ALL verwendet, so bleibt die aktuelle Zeile erhalten, egal ob expandiert wurde oder nicht.

`FILE [ filename ]` `FIL`

Dieser Befehl schreibt den Inhalt des workfiles in eine Datei. `FILE` ist ausführlich in Abschnitt 3.1.4, *Speichern einer Datei*, beschrieben.

`FILL [ kette ]` `FILL`

Dieser Befehl füllt Sätze auf, indem so viele Wörter aus dem nächsten Satz in den bearbeiteten übernommen werden, wie in der sichtbaren Zeilenbreite (Wert von `LWWIDTH`) Platz haben. Das gleiche geschieht für den nächsten Satz usw.

Das Auffüllen beginnt immer mit dem Satz der aktuellen Zeile.

Das Auffüllen endet entweder vor der nächsten Leerzeile oder vor demjenigen Satz, der mit `kette` beginnt, oder am Dateiende. Gibst du keinen Parameter an, so ist der Stoppsatz die nächste Leerzeile.

Wenn in einem Text Absätze durch Leerzeilen oder durch eine eindeutige Markierung am Ende des Absatzes (aber am Anfang eines Satzes) gekennzeichnet sind, kannst du so den Text absatzweise auffüllen. Allerdings solltest du darauf achten, dass die Leerzeilen oder die Stoppketten vorhanden sind, bzw. richtig eingegeben werden, da sonst das Auffüllen ungewollt bis zum Dateiende durchlaufen kann.

`HELP [ cmd | PREFIX ]` `H`

Wird dieser Befehl ohne Parameter eingegeben, so schreibt er eine Liste aller Befehle, Sonderhilfetexte und Präfixbefehle in das Fenster. Die Befehlsnamen und Sonderhilfetexte werden in dieser Liste mit Groß– und Kleinbuchstaben geschrieben. Die Großbuchstaben am Anfang eines solchen Begriffes geben an, wie weit er abgekürzt werden darf. Zum Beispiel besagt

```
COpy
```

dass der `COPY`–Befehl `COPY`, `COP` oder `CO` geschrieben werden kann, aber nicht `C`. Die Liste der Befehle ist nach diesen Minimalabkürzungen sortiert, also nicht exakt alphabetisch, was die ausgeschriebenen Namen angeht.

Gibst du bei `HELP` als Parameter einen Befehlsnamen an, so erhältst du Syntax und Bedeutung dieses Befehls. In der ersten Zeile der Ausgabe steht links die vollständige Syntax des Befehls. Dabei werden senkrechter Strich und eckige Klammern in derselben Bedeutung verwendet, wie sie in Abschnitt 3.2.1, *Notation*, beschrieben ist. In der ersten Zeile rechts steht eine kurze Charakterisierung des Befehls in Englisch. Die restlichen Zeilen der Ausgabe beschreiben den Befehl, so gut es in maximal 5 Zeilen geht. Diese Beschränkung wurde deshalb gewählt, damit du sowohl den gegenwärtigen workfile–Ausschnitt, als auch den abgerufenen Hilfetext gleichzeitig vor Augen haben kannst, wenn du einen dir nicht geläufigen Befehl eingeben willst.

Gibst du zum Befehlsnamen den Parameter `PREFIX` an, erhältst du den Hilfetext zu diesem Präfixbefehl. Der Parameter ist notwendig für Befehle, die es als Zeilenbefehl und als Präfixbefehl gibt, z.B. `D` oder `I`.

Der ausführliche Hilfetext ist nur in der vorliegenden Anleitung zu finden, die auch in Zweifelsfällen ausschlaggebend ist.

Mit dem Befehl `HELP` kannst du dir auch einige Sonderhilfetexte im Fenster zeigen lassen. Formal geht das so wie bei Hilfetexten zu Befehlen, nur dass es sich eben nicht um Befehle handelt.

- `function` zeigt alle *exaEdit*–Funktionen, vgl. Abschnitt 3.1.30.
- `profilex` gibt Informationen über die Profildateien.
- `symbolic` führt alle symbolischen Zeilennummern (für die Befehle `COPY`, `DL`, `MOVE` usw.) auf.

HEXA HEX

Gibst du diesen Befehl zum ersten Mal in einer *exaEdit*–Sitzung, so ändert sich die Darstellung der aktuellen Zeile von ASCII in hexadezimal.

Gibst du den Befehl erneut, wird die aktuelle Zeile wieder in ASCII (also normal) dargestellt usw.

Der Befehl HEXA dient nur der Darstellung. Für hexadezimales Editieren benötigst du den Parameter H der Befehle ALIGN, CHANGE, SSPLIT und der Befehle der LOCATE–Familie.

INDENT [? | n | AUTO | ON | OFF] IND

Dieser Befehl steuert das Verhalten von *exaEdit* beim automatischen Einrücken im Eingabemodus.

Das Einrücken wird mit ON bzw. OFF ein- bzw. ausgeschaltet. Der Parameter n sorgt für das Einrücken um n Spalten. Dies bedeutet, dass zu Beginn des Eintippens einer jeden Zeile der Cursor in Spalte n + 1 steht.

Gilt stattdessen der Parameter AUTO, so wird jede neue Zeile so weit eingerückt, wie die vorige Zeile eingerückt war. Der Einrückbetrag, den sich *exaEdit* von jeder Zeile merkt, wird duch die Eingabe einer Leerzeile nicht gestört.

Mit INDENT ? werden die gültigen Parameterwerte ausgewiesen.

Die Voreinstellung ist

```
INDENT AUTO ON
```

INLENGTH [? | n | AUTO | SOFT | HARD] INL

Dieser Befehl steuert das Verhalten von *exaEdit* beim automatischen Umbruch im Eingabemodus. Unter Umbruch ist zu verstehen, dass du bei der Eingabe im Eingabemodus Text für mehrere Zeilen eingeben kannst, ohne dich darum kümmern zu müssen, wieviele Zeichen in eine Zeile (einen Satz) passen. Einzelheiten darüber findest du im Abschnitt 3.1.18.2, *Automatischer Zeilenumbruch*.

Als Voreinstellung gilt, dass Sätze im Eingabemodus umgebrochen werden, wenn sie länger als LWWIDTH Spalten werden würden. Dies entspricht auch der Angabe des Parameters AUTO. Du kannst aber durch Angabe des Parameters n eine feste Spalte dafür angeben.

Ebenso gilt als Voreinstellung, dass der Umbruch am letzten Leerzeichen vor der angegebenen oder angenommenen Spalte erfolgt. Dies entspricht auch der Angabe des Parameters SOFT. Du kannst aber auch durch Angabe des Parameters HARD verlangen, dass der Umbruch genau an der Spalte, also unabhängig von eventuellen Wortgrenzen, erfolgt.

Mit INLENGTH ? werden die gültigen Parameterwerte ausgewiesen.

Die Voreinstellung ist

```
INLENGTH AUTO SOFT
```

INPUT [/kette/] I

Dieser Befehl hat zwei Bedeutungen, je nachdem ob du ihn mit oder ohne Parameter aufrufst.

Wird INPUT ohne Parameter eingegeben, so schaltet *exaEdit* vom Befehlsmodus in den Eingabemodus um. Willst du umgekehrt vom Eingabemodus in den Befehlsmodus zurückschalten, so musst du eine Leereingabe machen, d.h. die Enter–Taste drücken, ohne vorher eine andere Taste außer der Enter–Taste gedrückt zu haben.

Beim Einschalten des Eingabemodus wird

```
Eingabe
```

in das Fenster geschrieben (sozusagen als Aufforderung). Diese Ausgabe wird mit der nächsten Eingabe überschrieben. Als dauerhafte Anzeige für den Eingabemodus wird in der Statuszeile ein

```
I
```

angezeigt. Außerdem rutscht die Linealzeile nach links, so dass du für die Eingabe bei Bedarf Spalten abzählen kannst.

Du verwendest den Eingabemodus, um neue Sätze in den workfile zu bringen. Eingegeben werden diese Sätze in der Dialogzone (also da, wo du normalerweise die *exaEdit*–Befehle eingibst), eingefügt werden diese Sätze nach dem Satz in der aktuellen Zeile.

Jede eingegebene Zeile wird in der 1. Zeile der Dialogzone wiederholt, die 2. Zeile dieser Zone gelöscht und der Cursor für die nächste Eingabe an den Anfang dieser 2. Zeile gestellt. Du kannst jedoch deine Eingabe, wie bei *exaEdit* üblich, in jeder beliebigen Zeile des Fensters machen, insbesondere die in der 1. Zeile der Dialogzone stehende vorige Eingabezeile mitverwenden.

Solange sich *exaEdit* im Eingabemodus befindet, erhalten die eingegebenen und nach oben in die Datenzone gewanderten Eingabesätze keine Numerierung. Diese wird erst dann hinzugefügt, wenn der Eingabemodus wieder beendet wird, weil erst dann feststeht, wieviele Sätze eingefügt wurden. Wie diese Nummern für die eingefügten Sätze festgelegt werden, ist in Abschnitt 3.1.31, *Einfügen von Satznummern*, beschrieben.

Nun zur Verwendung des Befehls mit Parameter: Damit fügst du nach der aktuellen Zeile einen neuen Satz ein, den du in dem Parameter `kette` angegeben hast.

`INSMODE [? | ON | OFF]` INS

Dieser Befehl steuert das Verhalten von *exaEdit* bezüglich des Einfügemodus, siehe dazu auch den Abschnitt 3.1.10, *Löschen und Einfügen von Zeichen*. Er dient zum Hin- und Herschalten zwischen dem gewöhnlichen Einfügemodus (`INSMODE OFF`) und dem dauerhaften Einfügemodus (`INSMODE ON`). Die genannten Befehle können auch durch einfaches bzw. doppeltes Drücken der `Einfg`–Taste ersetzt werden (die es aber manchmal nicht auf der Tastatur gibt).

Mit `INSMODE ?` wird der Stand des Schalters angezeigt.

Die Voreinstellung ist

```
INSMODE OFF
```

`KEYBOARD [?|EXAEDIT|ALL|TEST]` KEYB

Dieser Befehl hat 2 verschiedene Funktionen. Zum einen änderst du damit die Antwort auf das Drücken bestimmter Tasten auf der Tastatur. Neben den Tasten, die mit normalen Zeichen beschriftet sind, gibt es solche, mit denen bestimmte Funktionen verbunden sind. Je nach der (Hard– oder Software–)Umgebung, in der *exaEdit* benutzt wird, können diese besonderen Tasten mit verschiedenen Informationen belegt sein. Aus diesem Grund gibt es Tasten, die nur derzeit, vielleicht aber auch künftig, für *exaEdit* keine Bedeutung haben. Sie werden, wenn gedrückt, von *exaEdit* ignoriert. Dies entspricht der Voreinstellung

```
keyboard exaedit
```

Falls jedoch

```
keyboard all
```

gilt, wird beim Drücken einer solchen Taste eine Meldung erzeugt, aus der hervorgeht, welche Information *exaEdit* mit dieser Taste erhalten hat, und dass diese Information ignoriert wird. Allerdings kann es vorkommen, dass die für die Taste erzeugte Meldung deine normale *exaEdit*–Benutzung stört, etwa wenn du eine solche Taste mitten in einer Eingabe drückst. Aus diesem Grund ist das schweigende Ignorieren solcher Tasten die Voreinstellung.

Mit dem Fragezeichen als Parameter kannst du den Stand des Schalters abfragen. Die Antwort ist dann `EXAEDIT` bzw. `ALL`.

Die andere Funktion des Befehls KEYBOARD verwendest du durch die Angabe des Parameters

```
TEST
```

exaEdit schreibt dann

```
Drücke Taste:
```

in das Fenster und erwartet zunächst das Drücken genau einer Taste. Falls *exaEdit* darauf antwortet, ist der Aufruf von KEYBOARD TEST fertig bearbeitet. Tut sich jedoch nichts, musst du noch die Enter–Taste drücken, um die gewünschte Reaktion von *exaEdit* zu erhalten.

Wie du die Meldung von *exaEdit* interpretieren musst, ist im Abschnitt 3.1.29, *Tastaturtest*, ausführlich beschrieben.

LANGUAGE [?|DEUTSCH|UDEUTSCH|ENGLISH] LA

Mit diesem Befehl stellst du die Sprache ein, in der *exaEdit* Meldungen und Hilfetexte ausgibt.

Die Voreinstellung von *exaEdit* für diesen Parameter ist UDEUTSCH oder ENGLISH, je nach installierter Version. UDEUTSCH bedeutet die deutsche Sprache und die Verwendung von Umlauten und Es–Zet. Wenn du diese Sonderzeichen nicht haben willst, etwa weil sie in deinem Fenster verstümmelt oder gar nicht erscheinen, so kannst du mittels

```
language deutsch
```

zu den Umschreibungen ae, ..., ss übergehen.

Beachte bitte, dass über eine Profildatei auch eine andere Sprache voreingestellt sein kann.

Mit dem Parameter ? zeigt *exaEdit* die Spracheinstellung an.

LOAD filename LOA

Verwandter Befehl: Aufruf von *exaEdit*

LOAD dient dem Laden einer Datei von der Platte in den aktuellen workfile. LOAD ist ausführlich im Abschnitt 3.1.3, *Laden einer Datei*, beschrieben.

LOCATE [sp1[sp2]] [/kette/[H][I]] L

Verwandte Befehle: RLOCATE, auch NLOCATE und RNLOCATE/NRLOCATE

LOCATE dient dem Suchen von Zeichenketten. Als Parameter gibst du eine Zeichenkette an. Diese muss normalerweise von zwei Begrenzern umgeben sein. Als Begrenzer wurde oben der Schrägstrich angegeben („/“), es ist aber jedes beliebige Zeichen erlaubt. Als praktisch hat sich erwiesen, immer dasjenige Sonderzeichen zu nehmen, das rechts unten auf der Tastatur liegt und nur dann ein anderes Zeichen, wenn das üblicherweise verwendete Bestandteil der zu suchenden Zeichenkette ist. Den Begrenzer am Ende der Suchkette darfst du weglassen, wenn du den Rest der Eingabezeile frei lässt.

```
locate /abc/
```

sucht nach der Zeichenkette „abc“,

```
l -a/b-
```

sucht nach der Zeichenkette „a/b“,

```
l abc
```

sucht nach der Zeichenkette „bc“, weil „a“ der Anfangsbegrenzer ist und der Endbegrenzer fehlt. `l abca` würde ebenfalls nach der Zeichenkette „bc“ suchen.

Bitte achte darauf, dass du beim Verketten mit anderen folgenden Befehlen den Endbegrenzer schreiben musst:

```
l /abc/;-2
```

sucht die Kette „abc“ und führt dann den Befehl `-2` aus, während

```
l /abc ;-2
```

die Kette „`abc ;-2`“ suchen würde.

Die Suche beginnt in dem Satz nach dem mit der aktuellen Zeile.

Wird die Zeichenkette gefunden, so erscheint der Satz, in der sie vorkommt, in der aktuellen Zeile; der vom workfile gezeigte Ausschnitt wird also entsprechend verschoben.

Wird die Zeichenkette nicht gefunden, so erscheint die Meldung

```
Zeichenkette nicht gefunden: ...
```

wobei für ... die gesuchte Kette eingetragen wird, und die aktuelle Zeile bleibt unverändert.

Ist *exaEdit* beim Suchen nach der Zeichenkette auch im letzten Satz des workfiles nicht fündig geworden, so wird normalerweise die Suche ab dem ersten Satz wieder aufgenommen, bis entweder ohne Erfolg der Ausgangssatz erreicht oder die Zeichenkette gefunden wurde. „Normalerweise“ bedeutet, dass der mit dem Befehl `WRAP` (siehe dort) bediente Schalter auf `ON` steht.

Um anzuzeigen, dass die Suche ab dem Anfang des workfiles weitergegangen ist, wird die Meldung

```
Suche ab Anfang (wrap)
```

erzeugt. Dieser Meldung folgt dann entweder die Positionierung des gezeigten Ausschnitts beim Finden der Zeichenkette oder die oben genannte Misserfolgsmeldung beim Nicht–Finden.

Gilt dagegen `WRAP OFF`, so endet die Suche spätestens im letzten workfile–Satz. War sie erfolglos, so werden die Meldungen

```
Datenende
Zeichenkette nicht gefunden: ...
```

erzeugt.

Oft ist es erforderlich, ein und dieselbe Zeichenkette mehrfach zu suchen. In diesem Fall genügt die Eingabe von `LOCATE` ohne Parameter: Dann wird automatisch die zuletzt verwendete Suchkette genommen.

Die Befehle `LOCATE`, `RLOCATE`, `NLOCATE` und `RNLOCATE`/`NRLOCATE` verwenden alle dieselbe Suchkette. Sie ist auch in allen workfiles dieselbe.

Normalerweise erstreckt sich die Suche auf den gesamten Bereich eines Satzes. Durch die Angabe von zwei Spalten als erste Parameter kannst du jedoch die Suche auf den angegebenen Bereich beschränken. Gefunden wird eine Zeichenkette nur dann, wenn sie vollständig im angegebenen Bereich liegt. Gibst du nur 1 Spalte an, so erstreckt sich der Suchbereich von dieser Spalte bis zum jeweiligen Satzende. Spalteneinschränkungen kannst du auch mit dem Befehl `ZONE` vornehmen, verwendest du beide Einschränkungen, so geht die im `LOCATE`–Befehl vor.

Hast du `LOCATE` schon benutzt und rufst es dann ohne Parameter auf, so gelten auch etwaige Spalteneinschränkungen weiter. Gibst du aber beim Aufruf eine neue Suchkette an, so gelten die in einem früheren Aufruf gemachten Spaltenbeschränkungen nicht mehr. Gibst du nach einem `LOCATE`–Aufruf eines neues `LOCATE` mit einer (neuen) Spaltenbeschränkung ein, so gilt weiter die alte Suchkette.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal suchen. Zu diesem Zweck muss die angegebene Zeichenkette hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um den nächsten Satz aufzusuchen, der ein Tabulatorzeichen (hexadezimal `09`) enthält, gibst du den Befehl

```
locate /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Suchen nach der angegebenen Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise findet

```
locate /ab/ i
```

`ab` oder `Ab` oder `aB` oder `AB`.

Wenn du den Parameter `I` öfters benutzen musst, kannst du auch mit dem Befehl `CASE` erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter `CASE`.

`LWWIDTH [n|?]` `LW`

Verwandter Befehl: `SZONE`

`LWWIDTH` (= logical window width) legt die logische Fensterbreite fest. Darunter ist die maximale Zeilenlänge zu verstehen, die gezeigt werden würde, wenn das physische Fenster breit genug wäre. Alle Sätze, die länger als `LWWIDTH` sind, werden in Folgezeilen dargestellt. Ist beispielsweise ein Satz 200 Bytes lang und hat `LWWIDTH` den Wert 60, so werden 4 Zeilen benötigt, wovon die letzte Zeile die letzten 20 Bytes des Satzes enthält.

Willst du die Folgezeilen im Fenster nicht haben, so kannst du mit einem entsprechend großen Wert von `LWWIDTH` dafür sorgen, dass die Daten der Folgezeilen nach rechts hinaus „verschwinden“.

Die Voreinstellung von `LWWIDTH` ist so gewählt, dass, gegebenenfalls unter Verwendung von Folgezeilen, alle Daten des workfiles sichtbar sind, bzw. beim vertikalen Blättern sichtbar werden.

Der voreingestellte Wert von `LWWIDTH` hängt daher von der physischen Fensterbreite und dem Wert von `SKEY` (siehe Befehl `SKEY`) ab:

`lwwidth` = physische Fensterbreite − `skey` − 1

Änderst du den Wert von `SKEY`, so passt sich `LWWIDTH` automatisch so an, dass immer die volle Breite des Fensters genutzt wird: Bei einer physischen Fensterbreite von 80 hat für `SKEY = 6` `LWWIDTH` den Wert 73, für `SKEY = 0` den Wert 79.

Gibst du den Befehl `LWWIDTH` ohne Parameter, so wird der voreingestellte Wert, jedoch unter Berücksichtigung des aktuellen Wertes von `SKEY`, angenommen. Beispielsweise ergibt bei einer physischen Fensterbreite von 80 und `SKEY = 0` die Eingabe von `LWWIDTH` alleine einen Wert von 79.

Der Parameter „?“ zeigt den aktuellen Wert von `LWWIDTH` an.

`MANUAL [name|*][?|DEFAULT|DELETE|SET /kette/]` `MAN`

Dieser *exaEdit*–Befehl dient dem Erstellen, Verändern, Löschen, Listen und Ausführen von Unix– oder DOS–Befehlen zum Ansehen der *exaEdit*–Benutzeranleitung.

Mit dem Befehl

```
manual * ?
```

erhältst du die Auflistung aller definierten `manual`–Parametersätze. Falls er nicht im Installationsprofil oder von dir selbst geändert oder gelöscht wurde, gibt es mindestens den in *exaEdit* fest einprogrammierten Parametersatz, der einem der folgenden ähnlich ist:

```
MAN    0!   start iexplore exaedit.de/dok/de/
MAN    0!   konqueror exaedit.de/dok/de/&
MAN    0!   netscape exaedit.de/dok/de/
```

Zeilen der ersten Art erscheinen in Windows–Systemen, Zeilen der zweiten Art in Linux–Systemen, Zeilen der dritten Art in anderen Unix–Systemen. Die einzelnen Teile haben die folgende Bedeutung:

`MAN` ist der (abgekürzte) *exaEdit*–Befehlsname, damit du die gesamte Zeile bei Bedarf nach entsprechender Veränderung als neuen Befehl abschicken kannst.

`0` ist der Name des `manual`–Parametersatzes. Solche Namen können aus 1 bis 4 Buchstaben oder Ziffern bestehen.

`!` markiert den `manual`–Parametersatz als denjenigen, der genommen wird, wenn beim `manual`–Aufruf fürs Ausführen (oder Listen) kein `manual`–Name angegeben wird.

`start iexplore` bzw. `netscape` usw.ist der Name des DOS– bzw. Unix–Befehls, der ausgeführt werden soll, um die *exaEdit*–Benutzeranleitung online zu zeigen.

`exaedit.de/dok/de/` ist die WWW–Datei, die der Befehl für den gewünschten Zweck benötigt.

`&` am Ende des Unix–Befehls sorgt dafür, dass der Befehl asynchron zu *exaEdit* arbeitet.

Mit dem Befehl

```
manual ?
```

wird nur der mit `!` markierte `manual`–Parametersatz gelistet.

Mit dem Befehl

```
manual name ?
```

wird nur der angegebene `manual`–Parametersatz gelistet.

Mit dem Befehl

```
manual
```

wird der mit `!` markierte `manual`–Parametersatz genommen und der darin enthaltene Unix– oder DOS–Befehl ausgeführt.

Mit dem Befehl

```
manual name
```

wird der angegebene `manual`–Parametersatz genommen und der darin enthaltene Unix– oder DOS–Befehl ausgeführt.

Mit dem Befehl

```
manual name default
```

wird der angegebene `manual`–Parametersatz mit `!` markiert, ein eventuell schon vorhandener verliert dabei seine Markierung.

Mit dem Befehl

```
manual name delete
```

wird der angegebene `manual`–Parametersatz gelöscht. Mit `*` statt `name` löschst du alle `manual`–Parametersätze.

Mit dem Befehl

```
manual name set /zeichenkette/
```

wird ein neuer `manual`–Parametersatz definiert. Wie schon gesagt, muss `name` aus 1 bis 4 Buchstaben oder Ziffern bestehen. Die `zeichenkette` muss ein vollständiger Unix– oder DOS–Befehl sein. Da dieser für unsere Zwecke im allgemeinen Schrägstriche enthält, musst du für die Begrenzung der `zeichenkette` vermutlich ein anderes Zeichen als den Schrägstrich verwenden.

Die Parameter `DELETE` und `SET` darfst du wie üblich abkürzen. Möchtest du einen Parametersatz definieren, der einem bereits vorhandenen ähnelt, so kannst du ihn dir mittels `manual name ?` anzeigen lassen und dann die Zeile der Anzeige nach Wunsch abändern und mit dem Drücken der `Enter`–Taste besiegeln.

Bitte denke auch daran, dass du die von dir gewünschten Änderungen oder Ergänzungen der vorhandenen `manual`–Parametersätze in deine private *exaEdit*–Profildatei stecken kannst.

`MOVE num1 [ num2 ]` `M`

Der Befehl `MOVE` verschiebt einen oder mehrere Sätze hinter den Satz der aktuellen Zeile des workfiles. Die zu verschiebenden Sätze werden durch ihre Nummer oder eine symbolische Nummer bezeichnet.

Die Satznummer kann ohne führende Nullen angegeben werden. Symbolische Satz- bzw. Zeilennummern sind:

- `*` für die aktuelle Zeile
- `p` (= previous) für die Zeile vor der aktuellen Zeile
- `n` (= next) für die Zeile nach der aktuellen Zeile
- `f` (= first) für die erste Zeile des workfiles
- `l` (= last) für die letzte Zeile des workfiles
- `t` (= top) für die top line des workfiles
- `b` (= bottom) für die letzte Zeile des workfiles
- `s` (= set) für die mit dem Befehl `SET` markierte Zeile

Gibst du nur 1 Nummer an, so wird nur dieser Satz verschoben; gibst du 2 Nummern an, so werden alle Sätze von der ersten bis zur zweiten Nummer (die Grenzen jeweils eingeschlossen) verschoben. Zu diesem Zweck muss `num1` $\leq$ `num2` gelten. Ist dies nicht der Fall, so erfolgt die Meldung

```
Erste Nummer größer als zweite
```

Gibst du eine Nummer an, die es im workfile nicht gibt, so erfolgt die Meldung

```
Nummer ... nicht gefunden
```

wobei für ... die Nummer eingesetzt wird.

Außerdem musst du darauf achten, dass der Verschiebebereich nicht das Ziel enthält: Das Ziel ist sozusagen die Stelle zwischen aktuellem und darauffolgendem Satz. Der Verschiebebereich darf also nicht den Satz der aktuellen Zeile und gleichzeitig den darauffolgenden enthalten, wie es zum Beispiel bei `MOVE` $*$ `num2` der Fall ist, wenn `num2` hinter der aktuellen Zeile liegt. Bei Verstoß gegen diese Bedingung erhältst du die Meldung

```
Ziel im MOVE-Bereich
```

Einige Beispiele:

`m400 700`	verschiebt die Sätze mit den Nummern 400 bis 700 hinter den Satz der aktuellen Zeile.
`m b`	verschiebt den letzten Satz.
`m p`	verschiebt den Satz, der vor dem Satz der aktuellen Zeile steht, hinter diesen Satz, vertauscht also zwei Sätze miteinander.

Ohne Angabe eines 3. Parameters werden die zu verschiebenden Sätze aus demselben workfile genommen, in den sie eingesetzt werden sollen. Du kannst aber auch als 3. Parameter einen workfile angeben, aus dem die Sätze zum Verschieben geholt werden sollen. Gibt es diesen nicht, so erhältst du die Meldung

```
Workfile nicht gefunden
```

Beispielsweise verschiebt

```
move 1000 1500 abc
```

die Sätze mit den Nummern 1000 bis 1500 aus dem workfile `abc` hinter die aktuelle Zeile des aktiven workfiles.

Ein weiteres Beispiel: Du möchtest den gesamten Inhalt des Hauptworkfiles (`main`) in einen neuen workfile namens `abc` verschieben. Dann geht das am einfachsten mit

```
wf abc;m f l main
```

Beachte bitte, dass die top line nicht verschoben werden kann, der Befehl `move t b` erzeugt die Fehlermeldung

```
Die top line kannst du nicht verschieben
```

Bei der Angabe des workfile–Namens musst du den 2. Parameter (die 2. Nummer) angeben (auch wenn er mit dem 1. identisch ist), wenn der workfile–Name mit einer symbolischen Zeilennummer verwechselt werden könnte. Beispielsweise verschiebt

```
m 500 a
```

den Satz mit der Nummer 500 aus dem workfile a, während

```
m 500b
```

alle Sätze im aktiven workfile ab der Nummer 500 meint, so dass du bei Verschieben des Satzes 500 aus dem workfile `b`

```
m500 500b
```

schreiben musst.

Die verschobenen Sätze erhalten Nummern, die sich an die vorhandenen Nummern der beiden Sätze, zwischen die eingesetzt wird, möglichst gut anpassen. Da das Verfahren, wie die neuen Nummern bestimmt werden, auch bei anderen Befehlen benutzt wird, ist es nur einmal beschrieben und zwar im Abschnitt 3.1.31, *Einfügen von Satznummern*.

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach unten, zum Ende des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach oben.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die nach dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

```
Datenende
```

in die Dialogzone und belässt die aktuelle Zeile.

`NLOCATE [sp1[sp2]] [/kette/[H][I]]` NL

Verwandte Befehle: RNLOCATE/NRLOCATE, auch LOCATE und RLOCATE

NLOCATE dient dem Aufsuchen der nächsten Zeile, die die angegebene Zeichenkette nicht enthält. Als Parameter gibst du eine Zeichenkette an. Diese muss normalerweise von zwei Begrenzern umgeben sein. Als Begrenzer wurde oben der Schrägstrich angegeben („/“), es ist aber jedes beliebige Zeichen erlaubt. Als praktisch hat sich erwiesen, immer dasjenige Sonderzeichen zu nehmen, das rechts unten auf der Tastatur liegt und nur dann ein anderes Zeichen, wenn das üblicherweise verwendete Bestandteil der zu suchenden Zeichenkette ist. Den Begrenzer am Ende der Suchkette darfst du weglassen, wenn du den Rest der Eingabezeile frei lässt.

```
nlocate /abc/
```

sucht nach der jeweils nächsten Zeile, die die Zeichenkette „abc“ nicht enthält,

```
nl -a/b-
```

sucht nach der jeweils nächsten Zeile, die die Zeichenkette „a/b“ nicht enthält,

```
nl abc
```

sucht nach der jeweils nächsten Zeile, die die Zeichenkette „bc“ nicht enthält, weil „a“ der Anfangsbegrenzer ist und der Endbegrenzer fehlt. `nl abca` würde ebenfalls nach der jeweils nächsten Zeile suchen, die die Zeichenkette „bc“ nicht enthält.

Bitte achte darauf, dass du beim Verketten mit anderen folgenden Befehlen den Endbegrenzer schreiben musst:

```
nl /abc/;-2
```

sucht die jeweils nächste Zeile, die die Kette „abc“ nicht enthält und führt dann den Befehl -2 aus, während

```
nl /abc ;-2
```

die jeweils nächste Zeile suchen würde, die die Kette „abc ;-2“ nicht enthält.

Die Suche beginnt in dem Satz nach dem mit der aktuellen Zeile.

Wird die jeweils nächste Zeile gefunden, die die angegebene Zeichenkette nicht enthält, so erscheint diese in der aktuellen Zeile; der vom workfile gezeigte Ausschnitt wird also entsprechend verschoben.

Wird keine Zeile ohne die Zeichenkette gefunden, so erscheint die Meldung

```
Zeichenkette in allen Zeilen: ...
```

wobei für ... die angegebene Zeichenkette eingetragen wird, und die aktuelle Zeile bleibt unverändert.

Hat *exaEdit* beim Suchen nach der Zeichenkette diese auch im letzten Satz des workfiles gefunden, so wird normalerweise die Suche ab dem ersten Satz wieder aufgenommen, bis entweder ohne Erfolg der Ausgangssatz erreicht oder eine Zeile ohne die Zeichenkette gefunden wurde. „Normalerweise“ bedeutet, dass der mit dem Befehl WRAP (siehe dort) bediente Schalter auf ON steht.

Um anzuzeigen, dass die Suche ab dem Anfang des workfiles weitergegangen ist, wird die Meldung

```
Suche ab Anfang (wrap)
```

erzeugt. Dieser Meldung folgt dann entweder die Positionierung des gezeigten Ausschnitts beim Nicht–Finden der Zeichenkette oder die oben genannte Misserfolgsmeldung.

Gilt dagegen `WRAP OFF`, so endet die Suche spätestens im letzten workfile–Satz. War sie erfolglos, also die Zeichenkette in allen durchsuchten Zeilen vorhanden, so werden die Meldungen

```
Datenende
Zeichenkette in allen Zeilen: ...
```

erzeugt.

Oft ist es erforderlich, ein und dieselbe Zeichenkette mehrfach zu suchen. In diesem Fall genügt die Eingabe von NLOCATE ohne Parameter: Dann wird automatisch die zuletzt verwendete Suchkette genommen.

Die Befehle NLOCATE, RNLOCATE/NRLOCATE, LOCATE und RLOCATE verwenden alle dieselbe Suchkette. Sie ist auch in allen workfiles dieselbe.

Normalerweise erstreckt sich die Suche auf den gesamten Bereich eines Satzes. Durch die Angabe von zwei Spalten als erste Parameter kannst du jedoch die Suche auf den angegebenen Bereich beschränken. Gefunden wir eine Zeile nur dann, wenn sich die angegebene Suchkette nicht vollständig im angegebenen Bereich befindet. Gibst du nur 1 Spalte an, so erstreckt sich der Suchbereich von dieser Spalte bis zum jeweiligen Satzende. Spalteneinschränkungen kannst du auch mit dem Befehl ZONE vornehmen, verwendest du beide Einschränkungen, so geht die im NLOCATE–Befehl vor.

Hast du NLOCATE schon benutzt und rufst es dann ohne Parameter auf, so gelten auch etwaige Spalteneinschränkungen weiter. Gibst du aber beim Aufruf eine neue Suchkette an, so gelten die in einem früheren Aufruf gemachten Spaltenbeschränkungen nicht mehr. Gibst du nach einem NLOCATE–Aufruf eines neues NLOCATE mit einer (neuen) Spaltenbeschränkung ein, so gilt weiter die alte Suchkette.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal suchen. Zu diesem Zweck muss die angegebene Zeichenkette hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um den nächsten Satz aufzusuchen, der kein Tabulatorzeichen (hexadezimal 09) enthält, gibst du den Befehl

```
nl /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Suchen nach der angegebenen Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise findet

```
nl /ab/ i
```

die nächste Zeile, in der weder ab noch Ab noch aB noch AB vorkommen.

Wenn du den Parameter I öfters benutzen musst, kannst du auch mit dem Befehl CASE erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter CASE.

NRLOCATE | RNLOCATE [sp1[sp2]] [/kette/[H][I]] NRL | RNL

Verwandte Befehle: NLOCATE, auch RLOCATE und LOCATE

NRLOCATE sucht rückwärts die jeweils nächste Zeile, die die angegebene Zeichenkette nicht enthält. Als Parameter gibst du eine Zeichenkette an. Diese muss normalerweise von zwei Begrenzern umgeben sein. Als Begrenzer wurde oben der Schrägstrich angegeben („/“), es ist aber jedes beliebige Zeichen erlaubt. Als praktisch hat sich erwiesen, immer dasjenige Sonderzeichen zu nehmen, das rechts unten auf der Tastatur liegt und nur dann ein anderes Zeichen, wenn das üblicherweise verwendete Bestandteil der angegebenen Zeichenkette ist. Den Begrenzer am Ende der Zeichenkette darfst du weglassen, wenn du den Rest der Eingabezeile frei lässt.

```
nrlocate /abc/
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „abc“ nicht enthält,

```
nrl -a/b-
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „a/b“ nicht enthält,

```
nrl abc
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „bc“ nicht enthält, weil „a“ der Anfangsbegrenzer ist und der Endbegrenzer fehlt. `nrl abca` würde ebenfalls rückwärts nach der jeweils nächsten Zeile suchen, die die Zeichenkette „bc“ nicht enthält.

Bitte achte darauf, dass du beim Verketten mit anderen folgenden Befehlen den Endbegrenzer schreiben musst:

```
nrl /abc/;-2
```

sucht rückwärts die jeweils nächste Zeile, die die Kette „abc“ nicht enthält und führt dann den Befehl `-2` aus, während

```
nrl /abc ;-2
```

rückwärts die jeweils nächste Zeile suchen würde, die die Kette „`abc ;-2`“ nicht enthält.

Die Suche beginnt in dem Satz vor dem mit der aktuellen Zeile.

Wird rückwärts die jeweils nächste Zeile gefunden, die die angegebene Zeichenkette nicht enthält, so erscheint diese in der aktuellen Zeile; der vom workfile gezeigte Ausschnitt wird also entsprechend verschoben.

Wird keine Zeile ohne die Zeichenkette gefunden, so erscheint die Meldung

```
Zeichenkette in allen Zeilen: ...
```

wobei für ... die angegebene Zeichenkette eingetragen wird, und die aktuelle Zeile bleibt unverändert.

Hat *exaEdit* beim Suchen nach der Zeichenkette diese auch im ersten Satz des workfiles gefunden, so wird normalerweise die Suche ab dem letzten Satz wieder aufgenommen, bis entweder ohne Erfolg der Ausgangssatz erreicht oder eine Zeile ohne die Zeichenkette gefunden wurde. „Normalerweise“ bedeutet, dass der mit dem Befehl `WRAP` (siehe dort) bediente Schalter auf `ON` steht.

Um anzuzeigen, dass die Suche ab dem Ende des workfiles weitergegangen ist, wird die Meldung

```
Suche ab Ende (wrap)
```

erzeugt. Dieser Meldung folgt dann entweder die Positionierung des gezeigten Ausschnitts beim Nicht–Finden der Zeichenkette oder die oben genannte Misserfolgsmeldung.

Gilt dagegen `WRAP OFF`, so endet die Suche spätestens im ersten workfile–Satz. War sie erfolglos, also die Zeichenkette in allen durchsuchten Zeilen vorhanden, so werden die Meldungen

```
Datenanfang
Zeichenkette in allen Zeilen: ...
```

erzeugt.

Oft ist es erforderlich, ein und dieselbe Zeichenkette mehrfach zu suchen. In diesem Fall genügt die Eingabe von `NRLOCATE` ohne Parameter: Dann wird automatisch die zuletzt verwendete Suchkette genommen.

Die Befehle `NRLOCATE`, `RNLOCATE`, `NLOCATE`, `LOCATE` und `RLOCATE` verwenden alle dieselbe Suchkette. Sie ist auch in allen workfiles dieselbe.

Normalerweise erstreckt sich die Suche auf den gesamten Bereich eines Satzes. Durch die Angabe von zwei Spalten als erste Parameter kannst du jedoch die Suche auf den angegebenen Bereich beschränken. Gefunden wir eine Zeile nur dann, wenn sich die angegebene Suchkette nicht vollständig im angegebenen Bereich befindet. Gibst du nur 1 Spalte an, so erstreckt sich der Suchbereich von dieser Spalte bis zum jeweiligen Satzende. Spalteneinschränkungen kannst du auch mit dem Befehl `ZONE` vornehmen, verwendest du beide Einschränkungen, so geht die im `NRLOCATE`–Befehl vor.

Hast du NRLOCATE schon benutzt und rufst es dann ohne Parameter auf, so gelten auch etwaige Spalteneinschränkungen weiter. Gibst du aber beim Aufruf eine neue Suchkette an, so gelten die in einem früheren Aufruf gemachten Spaltenbeschränkungen nicht mehr. Gibst du nach einem NRLOCATE–Aufruf eines neues NRLOCATE mit einer (neuen) Spaltenbeschränkung ein, so gilt weiter die alte Suchkette.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal suchen. Zu diesem Zweck muss die angegebene Zeichenkette hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um rückwärts den nächsten Satz aufzusuchen, der kein Tabulatorzeichen (hexadezimal 09) enthält, gibst du den Befehl

```
nrl /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Rückwärtssuchen nach der angegebenen Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise findet

```
nrl /ab/ i
```

rückwärts die nächste Zeile, in der weder ab noch Ab noch aB noch AB vorkommen.

Wenn du den Parameter I öfters benutzen musst, kannst du auch mit dem Befehl CASE erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter CASE.

PFK [n|ALL][?|LOCK|UNLOCK|SET /kette/] PF

PFK steht für program function key, also für Tasten, die Programmfunktionen (im Gegensatz zu Betriebssystemfunktionen) aufrufen. Die betroffenen Tasten sind üblicherweise mit F1, F2, ... beschriftet, weshalb wir auch von F–Tasten sprechen. Wozu die Belegungen von F–Tasten mit Befehlen oder *exaEdit*–Funktionen dienen und wie du damit umgehen kannst, ist in Abschnitt 3.1.23, *Programmierbare Funktionstasten*, beschrieben.

PFK hat im Prinzip 2 Parameter. Der 1. gibt an, welche F–Tasten betroffen sind, der 2. gibt die Funktion an (zeigen, sperren, entsperren, belegen).

Der 1. Parameter ist entweder die Nummer der F–Taste oder ALL, wenn alle F–Tasten gemeint sind. Lässt du ihn weg, so werden bei der Funktion *zeigen* alle F–Tasten angenommen, bei den anderen Funktionen gar keine. Mit anderen Worten, der 1. Parameter ist bei den Funktionen LOCK, UNLOCK und SET erforderlich.

Statt einer einzelnen Angabe n kannst du auch einen Bereich n-m oder n:m angeben, oder eine Liste aus Zahlen und Bereichen, z.B. 3 4-7 9.

Lässt du den 2. Parameter weg, so nimmt *exaEdit* an, dass du die Funktion *zeigen* gemeint hast. Daher sind die Befehle

```
pfk all ?      pfk all      pfk ?      pfk
```

identisch, ebenso pfk n ? und pfk n, usw.

Mit dem Parameter LOCK sperrst du die angegebene(n) F–Taste(n). Dies bedeutet, dass du sie nicht mehr durch Eingabe von Zeichen und Drücken der Taste belegen kannst. F–Tasten, die mittels Parameter SET belegt wurden, sind automatisch gesperrt.

Mit dem Parameter UNLOCK hebst du die Sperre der angegebenen F–Taste(n) auf.

Mit dem Parameter SET /kette/ belegst du die angegebene(n) F–Taste(n) mit der kette. Die Begrenzungszeichen von kette sind wie üblich beliebig, den Schlussbegrenzer darfst du weglassen.

```
pfk 5 set qwer
```

belegt die F–Taste 5 mit „wer“.

Bitte beachte beim Belegen mit *exaEdit*–Funktionen, dass zu diesen der Anfangs– und Schlussapostroph dazugehört. Von den Definitionen

```
pfk 1 set /del/
pfk 2 set 'del'
pfk 3 set /'del
pfk 4 set /'del'
```

ergibt nur F4 die *exaEdit*–Funktion del, während die ersten beiden den *exaEdit*–Befehl delete ergeben und die dritte nichts Sinnvolles darstellt.

POINT num PO

Verwandte Befehle: Nummerbefehl, +, -, NEXT, BACK, UP, DOWN, TOP, BOTTOM, RETURN

Der Befehl POINT macht den Satz mit der angegebenen Nummer zum Satz der aktuellen Zeile, der gezeigte workfile–Ausschnitt wird entsprechend verschoben.

Die Satznummer kann ohne führende Nullen angegeben werden.

Als Satznummern sind auch symbolische Satznummern erlaubt (wie sie zum Beispiel bei den Befehlen COPY, DELETEL oder MOVE beschrieben sind), die aber meist mit Vorteil durch direkte Positionierungsbefehle zu ersetzen sind, also etwa TOP anstelle von POINT T.

Gibst du eine nicht vorhandene Nummer an, so erhältst du die Meldung

```
Nummer ... nicht gefunden
```

PROFILE [?|EXEC[ALL]|LIST[ALL]|LOAD[ALL]] PRO

Der Befehl PROFILE zeigt ohne Parameter oder mit „?“ die Profildateien, die *exaEdit* beim Start gesucht und verwendet hat. Einzelheiten dazu findest du in Abschnitt 3.1.26, *Die Profildateien*.

Die restlichen Parameter dienen dazu, die Befehle aus den Profildateien

- erneut auszuführen (EXEC),
- im Fenster aufzulisten (LIST),
- in den aktuellen workfile zu laden (LOAD).

Der Parameter ALL gibt jeweils an, ob alle Profilbefehle gemeint sind, oder nur diejenigen, die bei Starten eines neuen workfiles ausgeführt werden (und die deshalb mit einem Ausrufezeichen in Spalte 1 markiert sind).

`QUIT | END` `Q | E`

Der Befehl `QUIT` beendet den Editor. Vorher wird festgestellt, ob du in den vorhandenen workfiles Änderungen vorgenommen hast. Falls ja, so wirst du darauf hingewiesen und erhältst die Möglichkeit, die *exaEdit*–Sitzung weiterzuführen, beispielsweise, um die geänderten workfiles auf einen Datenträger zu schreiben.

Gibt es in der *exaEdit*–Sitzung nur den workfile `MAIN`, so erhältst du die Meldung

```
Änderungen nicht gesichert
Drücke J oder Y, um aufzuhören:
```

Befindet *exaEdit* sich im Zeilenmodus, so lautet die zweite Zeile:

```
Gib J oder Y ein, um aufzuhören:
```

Gibt es noch andere workfiles, so lautet die erste Zeile der Meldung stattdessen

```
Nicht gesicherte workfiles: ...
```

wobei anstelle von ... alle betroffenen workfiles aufgeführt sind.

Falls du dich entschließt, die Änderungen nicht zu sichern, drückst du einfach die Taste „`j`“ oder „`y`“ (klein oder groß), und der Editor beendet sofort seine Arbeit. Drückst du dagegen irgendeine andere Taste, ist die Bearbeitung des Befehls `QUIT` beendet, und du kannst ganz normal weitereditieren.

Bitte beachte, dass beim Beantworten der Frage nach dem Aufhören das bloße Drücken einer Taste genügt: das Drükken der `Enter`–Taste ist nicht erforderlich.

`REKEY [ base[incr]] | ON | OFF | ?` `REK`

Der Befehl `REKEY` stellt im Nummernfeld des workfiles wieder gleichabständige Numerierungen her.

Die Voreinstellung ist `REKEY 100 100`, also genau die Werte, die beim Laden einer Datei in den workfile verwendet werden.

Als Anfangswert `base` kannst du eine nicht negative ganze Zahl (`0`, `1`, `2`, `3`, ...) wählen, als Differenzwert `incr` eine positive Zahl (`1`, `2`, `3`, ...). Bei einem Differenzwert `0` erfolgt die Meldung

```
Anzahl 0 nicht erlaubt
```

Würden die entstehenden Zeilennummern größer als `99999999` werden, so wird der Befehl mit der Meldung

```
REKEY ergibt zu große Zahl
```

zurückgewiesen.

Gibst du den Befehl `REKEY` ohne Parameter, so werden die zuletzt benutzten Werte für Anfang und Differenz verwendet.

Die Parameter `ON` und `OFF` werden zwar geprüft und ausgewiesen, haben aber derzeit keine Bedeutung.

Mit dem Parameter `?` verlangst du die Anzeige der `REKEY`–Werte.

`REPLACE col1[col2]/kette/[n]` R

Verwandter Befehl: `CHANGE`, auch `CONCAT`

Mit dem Befehl `REPLACE` kannst du in die angegebenen Zeilen beliebige Zeichenfolgen einsetzen. Im Unterschied zu `CHANGE` ist es gleich, welchen Inhalt die zu ersetzende Zeichenkette hat.

`REPLACE` ersetzt im Satz der aktuellen Zeile oder — mit dieser beginnend — in `n` Zeilen.

Die Spalte, ab der die Zeichenkette eingesetzt werden soll, musst du immer angeben. Durch Angeben oder Weglassen einer 2. Spalte steuerst du das Verhalten von `REPLACE` wie folgt:

Gibst du keine Endspalte an, so werden, beginnend mit der Anfangsspalte, so viele Zeichen eingesetzt, wie `kette` lang ist. Zeichen, die eventuell danach noch kommen, bleiben ungeändert.

Gibst du dagegen Anfangs– und Endspalte an, so verhält sich `REPLACE` wie `CHANGE`, bei dem der zu ersetzende Inhalt beliebig wäre. Mit anderen Worten: die durch Anfangs– und Endspalte begrenzte Zeichenkette wird herausgeschnitten, `kette` wird eingesetzt, und der alte Inhalt jenseits der Endspalte wird bei Bedarf passend nach links oder rechts verschoben, so dass er unmittelbar an die eingesetzten Zeichen angrenzt.

Beispiele: Die aktuelle Zeile enthalte den Text

```
Es war gutes Wetter
```

Dann ergeben die Befehle

```
replace 4/ist/
r8 12/schlechtes
```

die Zeilen

```
Es ist gutes Wetter
Es ist schlechtes Wetter
```

`RETURN` RET

Verwandte Befehle: `POINT`, Nummerbefehl

Der Befehl `RETURN` macht denjenigen Satz zum Satz der aktuellen Zeile, der zuvor mit einem `SET`–Befehl (siehe dort) festgelegt wurde. Der gezeigte workfile–Ausschnitt wird entsprechend verschoben.

Ging dem `RETURN`–Befehl noch kein `SET`–Befehl voraus, so erfolgt die Meldung

```
SET-Speicher unbenutzt
```

und die aktuelle Zeile ändert sich nicht.

Ist der mit `SET` markierte Satz nicht mehr vorhanden, so erfolgt die Meldung

```
SET-Speicher geändert, Rückkehr zum vorhergehenden Satz
```

und der vorige Satz kommt in die aktuelle Zeile.

Ein Sprung von einem workfile in einen anderen ist mit `RETURN` nicht möglich, da jeder workfile seinen eigenen `SET`–Speicher hat.

`RLOCATE [sp1[sp2]] [/kette/[H][I]]` `RL`

Verwandte Befehle: `LOCATE`, auch `NLOCATE` und `RNLOCATE`/`NRLOCATE`

Dieser Befehl dient dem rückwärtigen Suchen von Zeichenketten. Als Parameter gibst du eine Zeichenkette an. Diese muss normalerweise von zwei Begrenzern umgeben sein. Als Begrenzer wurde oben der Schrägstrich angegeben („/“), es ist aber jedes beliebige Zeichen erlaubt. Als praktisch hat sich erwiesen, immer dasjenige Sonderzeichen zu nehmen, das rechts unten auf der Tastatur liegt und nur dann ein anderes, wenn das üblicherweise verwendete Zeichen Bestandteil der zu suchenden Zeichenkette ist. Den Begrenzer am Ende der Suchkette darfst du weglassen, wenn du den Rest der Eingabezeile frei lässt.

```
rlocate /abc/
```

sucht rückwärts nach der Zeichenkette „`abc`“,

```
rl -a/b-
```

sucht rückwärts nach der Zeichenkette „`a/b`“,

```
rl abc
```

sucht rückwärts nach der Zeichenkette „`bc`“, weil „`a`“ der Anfangsbegrenzer ist und der Endbegrenzer fehlt. `rl abca` würde ebenfalls rückwärts nach der Zeichenkette „`bc`“ suchen.

Bitte achte darauf, dass du beim Verketten mit anderen folgenden Befehlen den Endbegrenzer schreiben musst:

```
rl /abc/;-2
```

sucht rückwärts die Kette „`abc`“ und führt dann den Befehl `-2` aus, während

```
rl /abc ;-2
```

rückwärts die Kette „`abc ;-2`“ suchen würde.

Die rückwärtige Suche beginnt in dem Satz vor dem mit der aktuellen Zeile.

Wird die Zeichenkette gefunden, so erscheint der Satz, in der sie vorkommt, in der aktuellen Zeile; der vom workfile gezeigte Ausschnitt wird also entsprechend verschoben.

Wird die Zeichenkette nicht gefunden, so erscheint die Meldung

```
Zeichenkette nicht gefunden: ...
```

wobei für ... die gesuchte Kette eingetragen wird, und die aktuelle Zeile bleibt unverändert.

Ist *exaEdit* beim rückwärtigen Suchen nach der Zeichenkette auch im ersten Satz des workfiles nicht fündig geworden, so wird normalerweise die rückwärtige Suche ab dem letzten Satz wieder aufgenommen, bis entweder ohne Erfolg der Ausgangssatz erreicht oder die Zeichenkette gefunden wurde. „Normalerweise“ bedeutet, dass der mit dem Befehl `WRAP` (siehe dort) bediente Schalter auf `ON` steht.

Um anzuzeigen, dass die rückwärtige Suche ab dem Ende des workfiles weitergegangen ist, wird die Meldung

```
Suche ab Ende (wrap)
```

erzeugt. Dieser Meldung folgt dann entweder die Positionierung des gezeigten Ausschnitts beim Finden der Zeichenkette oder die oben genannte Mißerfolgsmeldung beim Nicht–Finden.

Gilt dagegen `WRAP OFF`, so endet die rückwärtige Suche spätestens im ersten workfile–Satz. War sie erfolglos, so werden die Meldungen

```
Datenanfang
Zeichenkette nicht gefunden: ...
```

erzeugt.

Oft ist es erforderlich, ein und dieselbe Zeichenkette mehrfach zu suchen. In diesem Fall genügt die Eingabe von RLOCATE ohne Parameter: Dann wird automatisch die zuletzt verwendete Suchkette genommen.

Die Befehle RLOCATE, LOCATE, NLOCATE und RNLOCATE/NRLOCATE verwenden alle dieselbe Suchkette. Sie ist auch in allen workfiles dieselbe.

Normalerweise erstreckt sich die Suche auf den gesamten Bereich eines Satzes. Durch die Angabe von zwei Spalten als erste Parameter kannst du jedoch die Suche auf den angegebenen Bereich beschränken. Gefunden wir eine Zeichenkette nur dann, wenn sie vollständig im angegebenen Bereich liegt. Gibst du nur 1 Spalte an, so erstreckt sich der Suchbereich von dieser Spalte bis zum jeweiligen Satzende. Spalteneinschränkungen kannst du auch mit dem Befehl ZONE vornehmen, verwendest du beide Einschränkungen, so geht die im RLOCATE–Befehl vor.

Hast du RLOCATE schon benutzt und rufst es dann ohne Parameter auf, so gelten auch etwaige Spalteneinschränkungen weiter. Gibst du aber beim Aufruf eine neue Suchkette an, so gelten die in einem früheren Aufruf gemachten Spaltenbeschränkungen nicht mehr. Gibst du nach einem RLOCATE–Aufruf eines neues RLOCATE mit einer (neuen) Spaltenbeschränkung ein, so gilt weiter die alte Suchkette.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal rückwärts suchen. Zu diesem Zweck muss die angegebene Zeichenkette hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um rückwärts den nächsten Satz aufzusuchen, der ein Tabulatorzeichen (hexadezimal 09) enthält, gibst du den Befehl

```
rl /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Rückwärtssuchen nach der angegebenen Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise findet

```
rl /ab/ i
```

rückwärts die nächste Zeile, in der entweder ab oder Ab oder aB oder AB vorkommt.

Wenn du den Parameter I öfters benutzen musst, kannst du auch mit dem Befehl CASE erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter CASE.

RNLOCATE | NRLOCATE [sp1 [sp2]] [/kette/ [H] [I]] RNL | RNL

Verwandte Befehle: NLOCATE, auch RLOCATE und LOCATE

RNLOCATE sucht rückwärts die jeweils nächste Zeile, die die angegebene Zeichenkette nicht enthält. Als Parameter gibst du eine Zeichenkette an. Diese muss normalerweise von zwei Begrenzern umgeben sein. Als Begrenzer wurde oben der Schrägstrich angegeben („/“), es ist aber jedes beliebige Zeichen erlaubt. Als praktisch hat sich erwiesen, immer dasjenige Sonderzeichen zu nehmen, das rechts unten auf der Tastatur liegt und nur dann ein anderes Zeichen, wenn das üblicherweise verwendete Bestandteil der angegebenen Zeichenkette ist. Den Begrenzer am Ende der Zeichenkette darfst du weglassen, wenn du den Rest der Eingabezeile frei lässt.

```
rnlocate /abc/
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „abc“ nicht enthält,

```
rnl -a/b-
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „a/b“ nicht enthält,

```
rnl abc
```

sucht rückwärts nach der jeweils nächsten Zeile, die die Zeichenkette „bc“ nicht enthält, weil „a“ der Anfangsbegrenzer ist und der Endbegrenzer fehlt. `rnl abca` würde ebenfalls rückwärts nach der jeweils nächsten Zeile suchen, die die Zeichenkette „bc“ nicht enthält.

Bitte achte darauf, dass du beim Verketten mit anderen folgenden Befehlen den Endbegrenzer schreiben musst:

```
rnl /abc/;-2
```

sucht rückwärts die jeweils nächste Zeile, die die Kette „abc“ nicht enthält und führt dann den Befehl `-2` aus, während

```
rnl /abc ;-2
```

rückwärts die jeweils nächste Zeile suchen würde, die die Kette „`abc ;-2`“ nicht enthält.

Die Suche beginnt in dem Satz vor dem mit der aktuellen Zeile.

Wird rückwärts die jeweils nächste Zeile gefunden, die die angegebene Zeichenkette nicht enthält, so erscheint diese in der aktuellen Zeile; der vom workfile gezeigte Ausschnitt wird also entsprechend verschoben.

Wird keine Zeile ohne die Zeichenkette gefunden, so erscheint die Meldung

```
Zeichenkette in allen Zeilen: ...
```

wobei für ... die angegebene Zeichenkette eingetragen wird, und die aktuelle Zeile bleibt unverändert.

Hat *exaEdit* beim Suchen nach der Zeichenkette diese auch im ersten Satz des workfiles gefunden, so wird normalerweise die Suche ab dem letzten Satz wieder aufgenommen, bis entweder ohne Erfolg der Ausgangssatz erreicht oder eine Zeile ohne die Zeichenkette gefunden wurde. „Normalerweise“ bedeutet, dass der mit dem Befehl `WRAP` (siehe dort) bediente Schalter auf `ON` steht.

Um anzuzeigen, dass die Suche ab dem Ende des workfiles weitergegangen ist, wird die Meldung

```
Suche ab Ende (wrap)
```

erzeugt. Dieser Meldung folgt dann entweder die Positionierung des gezeigten Ausschnitts beim Nicht–Finden der Zeichenkette oder die oben genannte Misserfolgsmeldung.

Gilt dagegen `WRAP OFF`, so endet die Suche spätestens im ersten workfile–Satz. War sie erfolglos, also die Zeichenkette in allen durchsuchten Zeilen vorhanden, so werden die Meldungen

```
Datenanfang
Zeichenkette in allen Zeilen: ...
```

erzeugt.

Oft ist es erforderlich, ein und dieselbe Zeichenkette mehrfach zu suchen. In diesem Fall genügt die Eingabe von `RNLOCATE` ohne Parameter: Dann wird automatisch die zuletzt verwendete Suchkette genommen.

Die Befehle `NRLOCATE`, `RNLOCATE`, `NLOCATE`, `LOCATE` und `RLOCATE` verwenden alle dieselbe Suchkette. Sie ist auch in allen workfiles dieselbe.

Normalerweise erstreckt sich die Suche auf den gesamten Bereich eines Satzes. Durch die Angabe von zwei Spalten als erste Parameter kannst du jedoch die Suche auf den angegebenen Bereich beschränken. Gefunden wir eine Zeile nur dann, wenn sich die angegebene Suchkette nicht vollständig im angegebenen Bereich befindet. Gibst du nur 1 Spalte an, so erstreckt sich der Suchbereich von dieser Spalte bis zum jeweiligen Satzende. Spalteneinschränkungen kannst du auch mit dem Befehl `ZONE` vornehmen, verwendest du beide Einschränkungen, so geht die im `RNLOCATE`–Befehl vor.

Hast du RNLOCATE schon benutzt und rufst es dann ohne Parameter auf, so gelten auch etwaige Spalteneinschränkungen weiter. Gibst du aber beim Aufruf eine neue Suchkette an, so gelten die in einem früheren Aufruf gemachten Spaltenbeschränkungen nicht mehr. Gibst du nach einem RNLOCATE–Aufruf eines neues RNLOCATE mit einer (neuen) Spaltenbeschränkung ein, so gilt weiter die alte Suchkette.

Mittels Angabe des Parameters

```
H
```

kannst du hexadezimal suchen. Zu diesem Zweck muss die angegebene Zeichenkette hexadezimal geschrieben werden. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um rückwärts den nächsten Satz aufzusuchen, der kein Tabulatorzeichen (hexadezimal 09) enthält, gibst du den Befehl

```
rnl /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Rückwärtssuchen nach der angegebenen Zeichenkette nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise findet

```
rnl /ab/ i
```

rückwärts die nächste Zeile, in der weder ab noch Ab noch aB noch AB vorkommen.

Wenn du den Parameter I öfters benutzen musst, kannst du auch mit dem Befehl CASE erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter CASE.

`SCOPE [?|ON|OFF]` SC

Mit diesem Befehl schaltest du *exaEdit* vom (normalen) Fenstermodus in den Zeilenmodus, vgl. Abschnitt 3.1.20, oder umgekehrt.

SCOPE ? zeigt die Stellung dieses Schalters.

SCOPE ohne Parameter bedeutet SCOPE ON.

`SEQUENCE [sp1[sp2]]/anf[dif]]/n[N|R][F]` SE

Dieser Befehl setzt in n aufeinanderfolgenden Zeilen in den jeweils gleichen Spalten Zahlen ein.

Voreinstellung ist, dass die Zahlenreihe mit 1 anfängt und das die Differenz zur nächsten Zahl ebenfalls 1 ist. Sollen beispielsweise 5 Zeilen mit den Zahlen 1 bis 5 gefüllt werden, so schreibst du etwa

```
sequence /1 1/ 5
```

Unter Berücksichtigung der genannten Voreinstellungen kannst du dies auch so schreiben:

```
sequence // 5
```

Die Anfangszahl und die Differenz kannst du natürlich auch anders wählen.

Die Zahlen werden rechtsbündig geschrieben. Die benötigte Spaltenbreite ergibt sich nach der größten zu erstellenden Zahl und das Feld beginnt in Spalte 1, wenn du nichts anderes angibst. Durch die Angabe des Parameters `sp1` bestimmst du, ab welcher Spalte die Zahlen einzusetzen sind. Gibst du zusätzlich `sp2` an, so legst du dadurch die Spaltenbreite fest. Durch die Angabe von `F` (fill) verlangst du, dass die eingesetzten Spalten links mit Nullen aufgefüllt werden. Beispiel:

```
se5 7/8/3f
```

ergibt in den Spalten 5 bis 7 den Inhalt

```
008
009
010
```

Die letzte zu beschreibende Voreinstellung ist `N` (new): Nach der Zeile mit dem aktuellen Satz werden `n` neue Sätze mit den gewünschten Zahlen eingesetzt. Gibst du aber den Parameter `R` (replace) an, so werden in den `n` Sätzen ab dem aktuellen die entsprechenden Spalten (in der vollen oben beschriebenen Breite) mit den Zahlen überschrieben. Gibt es am Ende des workfiles nicht genügend Sätze dafür, so gibt es zusätzlich die Meldung

```
Datenende
```

Die größte entstehende Zahl muss in die eventuell vorgegebene Feldbreite passen und darf auch nicht mehr als 8-stellig werden. Bei Verstoß erfolgt die Meldung

```
SEQUENCE übersteigt 99999999 oder Feldbreite
```

`SET [?]` SET

Gibst du `SET` ohne Parameter ein, so wird die Nummer des Satzes der aktuellen Zeile intern festgehalten.

Mit dem Befehl `RETURN` kannst du dann zu dem festgehaltenen Satz zurückspringen.

Die Angabe des Parameters „`?`“ bewirkt, dass der zuvor festgehaltene Satz in der Dialogzone gezeigt wird; ein Sprung zu diesem Satz erfolgt dabei nicht. Diesen Parameter kannst du also verwenden um festzustellen, wo ein `RETURN`–Befehl hinführen würde.

Ist der markierte Satz nicht mehr vorhanden, so erfolgt die Meldung

```
SET-Speicher geändert, Rückkehr zum vorhergehenden Satz
```

und der davor stehende Satz des workfiles wird angezeigt. Hast du mittels SET einen Satz markiert, so kannst du ihn mit der symbolischen Zeilennummer

```
s
```

in all den Befehlen ansprechen, die symbolische Zeilennummern zulassen (`COPY`, `COUNT`, `DL`, `MOVE`, `POINT`, `SORT` usw.).

`SKEY [?|n]` SK

Der Befehl „`SKEY n`“ legt fest, dass das Nummernfeld im Fenster n Ziffern breit sein soll. Die Voreinstellung ist 6, das Minimum ist 0, das Maximum ist 8.

Mit „`SKEY ?`“ erhältst du die Angabe, welches der aktuelle `SKEY`–Wert ist.

„`SKEY`“ ist gleichbedeutend mit „`SKEY 6`“.

Beachte bitte, dass beim Ändern des `SKEY`–Wertes (auch bei einer Änderung in sich selbst) der Wert von `LWWIDTH` (siehe dort) sich ebenfalls ändert.

`SORT [(line1 line2)][[A|D][I][N] where,[A|D][I][N] where,...]` SORT

Mit dem Befehl SORT kannst du Sätze sortieren. Das Befehlswort ist nicht abkürzbar, damit du den Befehl, der im allgemeinen ja nicht rückgängig gemacht werden kann, nicht so leicht aus Versehen geben kannst.

Fehlt die Angabe (line1 line2), so wird der ganze workfile sortiert. Mit (line1 line2) kannst du das Sortieren auf einen Teil der Sätze beschränken. Die Nummern können Satznummern sein, die auch ohne führende Nullen geschrieben werden können, oder sie können symbolische Nummern sein, wie sie zum Beispiel bei dem Befehl COPY beschrieben sind (siehe dort). Die erste Nummer darf nicht größer sein als die zweite.

Beispiele

`sort (500 b) ...`	sortiert die Zeilen von der Nummer 500 bis zum Ende
`sort (f n) ...`	sortiert die Zeilen von der ersten bis zur Zeile nach der aktuellen

Als nächstes kannst du Sortierrichtung, Sortierart und Sortierfelder angeben. Lässt du alle weiteren Angaben weg, so wird aufsteigend sortiert, Groß-/Kleinschreibung wird beachtet, es wird (auch bei Zahlen) nach Zeichen sortiert und die Sätze werden in ihrer ganzen Länge zum Vergleich herangezogen.

Die Sortierrichtung wird durch A (= ascending) für aufsteigend und D (= descending) für absteigend angegeben. Sie kann für jedes Sortierfeld einzeln festgelegt werden. Lässt du sie weg, so nimmt *exaEdit* A (aufsteigend) an.

Soll Groß-/Kleinschreibung nicht beachtet werden, so musst du die Angabe I (case insensitive) verwenden. Auch sie muss bei Bedarf für jedes Sortierfeld einzeln angegeben werden.

Enthalten die zu sortierenden Felder Zahlen, nach deren Wert zu sortieren ist, so musst du die Angabe N (numerisch) verwenden. Auch sie muss bei Bedarf für jedes Sortierfeld einzeln angegeben werden. Beim numerischen Sortieren gibt es noch folgenden Besonderheit: Der Satz oder das angegebene Feld wird vor dem Vergleich als Zahl interpretiert. Die Umwandlung wird mit der Funktion atof der Programmiersprache C vorgenommen. Dies bedeutet, dass die Interpretation (auch vor dem Satz- oder Feldende) vor dem ersten Zeichen endet, das nicht als zur Zahl gehörig erkannt wird. Es bedeutet außerdem den Wert 0, wenn keine Zahl vorliegt, einen unbestimmten Wert, wenn es zum Überlauf kommt (die Zahl also zu groß wäre), die Akzeptanz von führenden Zwischenraumzeichen (das sind Leerzeichen, Tabulatorzeichen, Zeilensprung, Seitenvorschub und Zeilenendekennung) und die Möglichkeit, die Zahlen mit und ohne Dezimalpunkt und mit und ohne Exponent zu schreiben.

Für die Angabe der Sortierfelder (das where in der SORT–Syntax) gibt es 2 Möglichkeiten:

```
anfangsspalte länge
```

oder

```
anfangsspalte : endspalte
```

die du beide natürlich auch gemischt verwenden kannst. Vor jeder Sortierfeldangabe können die oben beschriebenen Parameter A, D, I oder N stehen. Sollen sie für mehrere Sortierfelder gelten, so kannst du sie einmal vor die in Klammern gesetzten Sortierfelder setzen.

Schließlich kannst du noch zwischen die Sortierfeldangaben zur besseren Lesbarkeit eines der Zeichen Komma, Strichpunkt oder Schrägstrich setzen (wenn es sich nicht um den Befehlsseparator handelt).

Einige Beispiele:

```
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
```

Im letzten Beispiel sind 4 Sortierfelder angegeben, die alle 5 Zeichen lang sind. Die ersten beiden Felder werden numerisch, die letzten beiden alphanumerisch sortiert. Das dritte Feld wird absteigend sortiert, die anderen aufsteigend.

Hat *exaEdit* erfolgreich sortiert, so erfolgt die Meldung

```
Sortiert
```

Wurde dagegen ein Fehler festgestellt, so erscheint eine der im folgenden aufgeführten Meldungen:

```
Erste Nummer größer als zweite
```

Dies bedeutet, dass in der Klammer mit den Zeilennummern die erste Angabe eine numerische oder symbolische Zeilennummer ist, die größer ist als die Zeilennummer, die sich aus der zweiten Angabe ergibt.

```
Nummer ... nicht gefunden
```

Du hast eine nicht vorhandene Zeilennummer angegeben.

```
Es gibt keinen vorigen (p) Satz
```

Du hast die symbolische Zeilennummer `p` verwendet, obwohl es vor dem Satz der aktuellen Zeile keine Sätze mehr gibt.

```
Es gibt keinen folgenden (n) Satz
```

Du hast die symbolische Zeilennummer `n` verwendet, obwohl es nach dem Satz der aktuellen Zeile keine Sätze mehr gibt.

```
Die top line kannst du nicht mitsortieren
```

Die top line gehört mit zu den Sätzen, die du sortiert haben möchtest, sie kann aber nicht mitsortiert werden.

```
Anfangsspalte größer als Endspalte
```

Du hast ein Sortierfeld mit der genannten Eigenschaft angegeben.

```
Sortierfelder überlappen
```

Es gibt mindestens 1 Spalte, die in 2 Sortierfeldangaben vorkommt.

`SSPLIT [col1[col2]] [/string/[E][H][I]]` SS

Mit dem Befehl `SSPLIT` (= „string split“) kannst du den Satz der aktuellen Zeile in 2 Sätze aufteilen. Die einfachste Anwendung besteht in der Angabe einer Zeichenkette, an deren Anfang der Satz aufgetrennt wird. Hat die aktuelle Zeile beispielsweise den Inhalt

```
und sie schaute sie an
```

so macht der Befehl

```
ssplit -sie-
```

daraus die beiden Zeilen

```
und
sie schaute sie an
```

Möchtest du dagegen obige Zeile beim 2. Vorkommen des Wortes „`sie`“ trennen, so kannst du etwa

```
ssplit 10 -sie-
```

eingeben, also den Bereich einschränken, in dem die Zeichenfolge gesucht wird, an deren Beginn die Trennung erfolgen soll:

```
ssplit col1 col2 ...
```

sucht nur in den Spalten `col1` bis `col2` nach den Trennzeichen,

```
ssplit col1 ...
```

sucht in den Spalten ab `col1` nach den Trennzeichen.

Gibst du nach der Zeichenkette den Parameter `E` ein, so erfolgt die Trennung nicht am Anfang der Zeichenfolge, sondern am Ende:

```
ss-sie-e
```

ergibt daher

```
und sie
 schaute sie an
```

Eine Zeichenkette, an der getrennt werden soll, brauchst du nicht anzugeben. Es wird dann an der angegebenen Spalte getrennt:

```
ss 5
```

ergibt

```
und s
ie schaute sie an
```

Auf diese Art kannst du auch vor oder nach der aktuellen Zeile eine Leerzeile erzeugen:

`ss1` oder `ss`

ergibt eine Leerzeile vor der aktuellen Zeile,

```
ss n
```

mit einer Spalte `n`, die jenseits des letzten Nicht–Leerzeichens liegt, erzeugt eine Leerzeile nach der aktuellen Zeile, für obige Ausgangszeile also etwa

```
ss33
```

Mittels Angabe des Parameters

```
H
```

(= „hexadezimal“) kannst du die Zeichenkette auch in hexadezimaler Darstellung eingeben. Da 1 Byte durch 2 hexadezimale Zeichen dargestellt wird, verlangt *exaEdit*, dass immer eine gerade Anzahl von hexadezimalen Zeichen angegeben wird, ansonsten erfolgt die Fehlermeldung:

```
Ungerade Anzahl von Hex-Zeichen
```

Gibst du ein Zeichen ein, das nicht zur hexadezimalen Darstellung gehört, erhältst du die Meldung

```
Ungültiges Hex-Zeichen
```

Ein Beispiel: Um einen Satz an einem Tabulatorzeichen (hexadezimal `09`) aufzubrechen, das mittendrin vorkommt, schreibst du

```
ssplit /09/ h
```

Durch Angabe des Parameters

```
I
```

(= „groß/klein–**i**nsensibel“) verlangst du, dass beim Suchen nach der Zeichenkette, an der der Satz aufzubrechen ist, nicht zwischen großen und kleinen Buchstaben unterschieden wird. Beispielsweise kann

```
ssplit /ab/ i
```

bei `ab` oder `Ab` oder `aB` oder `AB` aufbrechen.

Wenn du den Parameter `I` öfters benutzen musst, kannst du auch mit dem Befehl `CASE` erreichen, dass der Parameter angenommen wird, ohne dass du ihn jedesmal angeben musst. Weitere Einzelheiten unter `CASE`.

`TEST [NO]LOG[n]|[NO]DUMP|[NO]KEEP|SHOW|EXAMINE|REPAIR|[NO]MON` `TE`

Der Befehl `TEST` dient der Gewinnung von Informationen bei der Fehlersuche im Editor. Er wird zum Editieren nie benötigt.

Du verwendest den Befehl nur, wenn du vom *exaEdit*–Autor entsprechend darum gebeten wurdest.

`TOP` `T`

Der Zeiger auf die aktuelle Zeile wird auf die top line gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach unten.

`TRANSLAT [(col1[col2])][U|L|?][n|ALL]` `TR`

Dieser Befehl (Achtung, er wird wie alle *exaEdit*–Befehle mit maximal 8 Buchstaben geschrieben, hat also kein `E` am Ende) übersetzt kleine Buchstaben in große (Parameter `U`) oder große Buchstaben in kleine (Parameter `L`). „Buchstaben“ bedeutet hier die 26 Buchstaben des gewöhnlichen Alphabets, also keine Umlaute.

Gibst du weder `U` noch `L` an, so erfolgt die Übersetzung wie im letzten Aufruf von `TRANSLAT` in der aktuellen *exaEdit*–Sitzung. Beim Start von *exaEdit* steht der Schalter auf `U`. Mit dem Parameter `?` (danach dürfen keine weiteren Parameter folgen) fragst du die Stellung des Schalters ab. Die beiden möglichen Meldungen sind

```
Übersetzung in Großbuchstaben
Übersetzung in Kleinbuchstaben
```

Gibst du keine weiteren Parameter an, so werden alle übersetzbaren Zeichen des aktuellen Satzes übersetzt.

Gibst du als letzten Parameter eine Zahl `n` an, so erstreckt sich die Übersetzung auf `n` Zeilen, beginnend mit der aktuellen. Gibt es nicht so viele Zeilen, so kommt wie üblich die Meldung

```
Datenende
```

Gibst du als letzten Parameter `ALL` an (er ist auch abkürzbar), so werden, unabhängig vom Stand der aktuellen Zeile, alle Zeilen des workfiles übersetzt. In diesem Fall bleibt auch der Zeiger auf die aktuelle Zeile konstant, während er bei einer expliziten Zahlenangabe `n` wie üblich auf die letzte bearbeitete Zeile gesetzt wird.

Ohne besondere Angabe werden alle Spalten der betroffenen Zeilen zum Übersetzen herangezogen. Du kannst aber auch die Spalten einschränken, indem du als ersten Parameter in runden Klammern erste und letzte Spalte angibst. Machst du in der Klammer nur eine Zahlenangabe, so erstreckt sich der Übersetzungsbereich von der angegebenen Spalte bis zum jeweiligen Zeilenende. Eine Einschränkung der Spalten kannst du auch durch Aufruf des Befehls `ZONE` (siehe dort) vornehmen. Machst du Spaltenangaben sowohl durch `ZONE`, als auch im Befehl `TRANSLAT`, so geht wie üblich die letztere Angabe vor.

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

Der Zeiger auf die aktuelle Zeile wird um `n` Sätze nach oben, zum Anfang des workfiles hin, gesetzt. Da die aktuelle Zeile eine feste Stelle im Fenster einnimmt, rutscht der Text entsprechend nach unten.

Gibst du `n` nicht an, so wird 1 angenommen.

Ist `n` größer als die Anzahl der Sätze, die (einschließlich der top line) vor dem Satz der aktuellen Zeile kommen, so schreibt *exaEdit*

```
Datenanfang
```

in die Dialogzone und belässt die aktuelle Zeile.

`WF [wfname|*[DELETE]]|[?[ALL]]` WF

`WF` ist eine Abkürzung des Befehls `WORKFILE`.

Mit dem Befehl `WF` kannst du workfiles erstellen, aktivieren, listen oder löschen. Ein workfile–Name besteht aus 1 bis 8 Buchstaben oder Ziffern und muss mit einem Buchstaben beginnen.

Der Befehl

```
WF wfname
```

erstellt einen workfile namens `wfname`, wenn es diesen noch nicht gibt, oder aktiviert den workfile `wfname`, wenn es diesen schon gibt. Der aktive workfile ist derjenige, der im Fenster gezeigt wird und auf den die Editierbefehle wirken.

Mit dem Befehl

```
WF wfname DELETE
```

wird der workfile `wfname` gelöscht und der workfile `MAIN` zum aktiven workfile gemacht. `DELETE` kannst du mit `D` abkürzen. Willst du den aktiven workfile löschen (das kann nicht der workfile `MAIN` sein), so kannst du auch `WF *DELETE` eingeben.

Mit dem Befehl

```
WF ?
```

wird der Name des aktiven workfiles gelistet.

Mit dem Befehl

```
WF ? ALL
```

werden die Namen aller workfiles gelistet. `ALL` kannst du mit `A` abkürzen.

`WIDTH [?|n|V]` WIDTH

Mit dem Befehl `WIDTH` kannst du angeben, wie die Daten der zu editierenden Datei in Editorzeilen aufgeteilt werden sollen. Der Befehlsname ist nicht abkürzbar.

Als Voreinstellung hat `WIDTH` den Wert `V` (= variabel) mit der Bedeutung, dass die Editorzeilen 1 : 1 den Dateisätzen entsprechen sollen. In der Datei werden Sätze durch ein bestimmtes Zeichen voneinander abgetrennt. Es handelt sich dabei um das sogenannte newline–Zeichen (`\n`, `X'0A'`). Beim Ausführen des Befehls `LOAD` werden normalerweise (das heißt, wenn `WIDTH V` gilt) die newline–Zeichen nicht mit in den workfile genommen, beim Ausführen des Befehls `FILE` werden sie wieder an jeden Satz des workfiles angehängt.

Hast du jedoch `WIDTH n` angegeben, so wird die einzulesende Datei in lauter gleich lange Stücke der Länge `n` zerhackt, von denen jedes einen Satz im workfile bildet. Die newline–Zeichen der Datei werden dabei wie normale Datenzeichen eingelesen. Beim Schreiben auf die Platte werden die Stücke aus dem workfile ohne Einfügen weiterer Zeichen wieder zusammengesetzt.

`WORKFILE [wfname|*[DELETE]] | [?[ALL]]` `WF`

Mit dem Befehl `WORKFILE` kannst du workfiles erstellen, aktivieren, listen oder löschen. Ein workfile–Name besteht aus 1 bis 8 Buchstaben oder Ziffern und muss mit einem Buchstaben beginnen.

Der Befehl

```
WORKFILE wfname
```

erstellt einen workfile namens `wfname`, wenn es diesen noch nicht gibt, oder aktiviert den workfile `wfname`, wenn es diesen schon gibt. Der aktive workfile ist derjenige, der im Fenster gezeigt wird und auf den die Editierbefehle wirken.

Mit dem Befehl

```
WORKFILE wfname DELETE
```

wird der workfile `wfname` gelöscht und der workfile `MAIN` zum aktiven workfile gemacht. `DELETE` kannst du mit `D` abkürzen. Willst du den aktiven workfile löschen (das kann nicht der workfile `MAIN` sein), so kannst du auch `WORKFILE * DELETE` eingeben.

Mit dem Befehl

```
WORKFILE ?
```

wird der Name des aktiven workfiles gelistet.

Mit dem Befehl

```
WORKFILE ? ALL
```

werden die Namen aller workfiles gelistet. `ALL` kannst du mit `A` abkürzen.

`WRAP [?|ON|OFF]` `WR`

Der Befehl `WRAP` beeinflusst das Verhalten des Befehls `LOCATE` und seiner Verwandten (`RLOCATE` usw.).

Gilt `WRAP OFF`, so endet die Suche nach Zeichenketten spätestens mit dem letzten bzw. ersten Satz des workfiles beim Vorwärts– bzw. Rückwärtssuchen.

Gilt dagegen `WRAP ON`, das ist auch die Voreinstellung, so wird nach Erreichen des letzten bzw. ersten Satzes die Suche am Anfang bzw. Ende des workfiles fortgesetzt, bis nötigenfalls alle Sätze des workfiles durchsucht worden sind.

Mit „`WRAP ?`“ kannst du den Stand des `WRAP`–Schalters abfragen.

`X [?|n|kette]` `X`

Verwandter Befehl: `Y`

Mit dem Befehl `X` kannst du eine Abkürzung für eine Folge beliebiger Befehle definieren und zur Ausführung bringen.

Mit dem Befehl

```
X kette
```

definierst du `X` als Abkürzung für „`kette`“, wobei „`kette`“ aus einer gültigen Befehlszeile (ein oder mehrere Befehle) bestehen muss.

Mit dem Befehl

```
X n
```

verlangst du, dass `X` n–mal ausgeführt wird. `X` alleine führt 1–mal aus. Willst du `X` ausführen, ohne es vorher definiert zu haben, erhältst du die Fehlermeldung

```
X ist nicht definiert
```

In „kette“ sind alle Befehle außer X erlaubt. Der Befehl Y ist nur erlaubt, wenn er nicht seinerseits wieder X aufruft. Ist dies der Fall, so wird an der entsprechenden Stelle die Ausführung mit der Meldung

```
Abbruch bei rekursivem X
```

abgebrochen. Ein Beispiel:

```
x n2;c/A/B/
```

Du möchtest in jeder 2. Zeile A durch B ersetzen. Dazu müsstest du die Befehle n2 und c/A/B/ wiederholt aufrufen. Statt dessen steckst du beide nach X und rufst dann „x 100“ auf, worauf der Inhalt von X 100–mal ausgeführt wird.

Sobald einer der Befehle, die in der Definition von X stecken, mit einer Warnung oder einem Fehler endet (Datenende, Zeichenkette nicht gefunden usw.), bricht die Abarbeitung des X–Befehls ab.

Mit dem Befehl

```
X ?
```

verlangst du, dass *exaEdit* die gültige Definition von X in das Fenster schreibt. Du kannst diesen Befehl beispielsweise verwenden, wenn du die Definition von X leicht abändern möchtest: du gehst mit dem Cursor in die ausgegebene Zeile, machst dort deine Änderungen und definierst X durch Drücken der Enter–Taste neu.

`Y [?|n|kette]` Y

Verwandter Befehl: X

Mit dem Befehl Y kannst du eine Abkürzung für eine Folge beliebiger Befehle definieren und zur Ausführung bringen.

Mit dem Befehl

```
Y kette
```

definierst du Y als Abkürzung für „kette“, wobei „kette“ aus einer gültigen Befehlszeile (ein oder mehrere Befehle) bestehen muss.

Mit dem Befehl

```
Y n
```

verlangst du, dass Y n–mal ausgeführt wird. Y alleine führt 1–mal aus. Willst du Y ausführen, ohne es vorher definiert zu haben, erhältst du die Fehlermeldung

```
Y ist nicht definiert
```

In „kette“ sind alle Befehle außer Y erlaubt. Der Befehl X ist nur erlaubt, wenn er nicht seinerseits wieder Y aufruft. Ist dies der Fall, so wird an der entsprechenden Stelle die Ausführung mit der Meldung

```
Abbruch bei rekursivem Y
```

abgebrochen. Ein Beispiel:

```
y n2;c/A/B/
```

Du möchtest in jeder 2. Zeile A durch B ersetzen. Dazu müsstest du die Befehle n2 und c/A/B/ wiederholt aufrufen. Statt dessen steckst du beide nach Y und rufst dann „y 100“ auf, worauf der Inhalt von Y 100–mal ausgeführt wird.

Sobald einer der Befehle, die in der Definition von Y stecken, mit einer Warnung oder einem Fehler endet (Datenende, Zeichenkette nicht gefunden usw.), bricht die Abarbeitung des Y–Befehls ab.

Mit dem Befehl

```
Y ?
```

verlangst du, dass *exaEdit* die gültige Definition von `Y` in das Fenster schreibt. Du kannst diesen Befehl beispielsweise verwenden, wenn du die Definition von `Y` leicht abändern möchtest: du gehst mit dem Cursor in die ausgegebene Zeile, machst dort deine Änderungen und definierst `Y` durch Drücken der `Enter`–Taste neu.

`ZONE [?|n[m]]` `Z`

Es gibt Befehle, deren Ergebnis vom Ort einer bestimmten Zeichenkette innerhalb der zu bearbeitenden Zeile abhängt. Als Beispiel betrachten wir `SSPLIT`. Der Befehl

```
ssplit /abc/
```

bedeutet bekanntlich, dass die aktuelle Zeile am Beginn der Zeichenkette '`abc`' in zwei Teile geteilt werden soll. Manchmal möchtest du vielleicht, dass dies nur geschieht, wenn '`abc`' ab Spalte 10 steht, aber nicht, wenn es vor Spalte 10 steht. Für diesen Zweck gibt es zwar die Möglichkeit, den gewünschten Bereich beim `SSPLIT`–Befehl direkt anzugeben, manchmal wäre es aber günstiger, den Befehl `SSPLIT` und ähnliche Befehle generell auf einen bestimmten Spaltenbereich zu begrenzen. Dies leistet der Befehl `ZONE`.

Durch Angabe von 1 Spalte schränkst du das Operationsfeld so ein, dass es in dieser Spalte beginnt und bis zum Ende des Satzes geht.

Durch Angabe von 2 Spalten legst du Anfang und Ende des Operationsfeldes im Satz fest.

`ZONE ?` zeigt dir die gültige Einstellung.

`ZONE` ohne Parameter führt die Voreinstellung zurück, bei der es keine Spalteneinschränkungen gibt.

Die von `ZONE` gesetzte Arbeitszone wird von den Befehlen `CHANGE`, der `LOCATE`–Familie, `SSPLIT` und `TRANSLAT` beachtet.

3.3 Die Präfixbefehle

Präfixbefehle werden nicht in der Dialogzone von *exaEdit* eingegeben, sondern sind extrem abgekürzte Befehle, die im Nummernpräfix derjenigen workfile–Zeilen eingegeben werden, auf die sie sich beziehen.

Wenn du beispielsweise mit dem Cursor in das Nummernfeld eines workfiles gehst, dort das Zeichen `"` tippst und dann die Entertaste drückst, führt die Ausführung dieses Präfixbefehls dazu, dass die markierte Zeile verdoppelt wird.

Du kannst in den verschiedenen Nummernfeldern des sichtbaren Teils des workfiles beliebig Präfixbefehle eingeben. Sie werden alle der Reihe nach (von oben nach unten) ausgeführt.

Du kannst zusätzlich (vor dem Enter) in der Dialogzone einen normalen *exaEdit*–Befehl eintippen. Auch dieser wird nach dem Enter, allerdings nach den Präfixbefehlen, ausgeführt.

Es gibt derzeit (spätere Ergänzungen sind beabsichtigt) die folgenden Präfixbefehle:

`"` Dieser Präfixbefehl verdoppelt die markierte Zeile.

`i` Dieser Präfixbefehl fügt nach der markierten Zeile eine Leerzeile ein.

`d` Dieser Präfixbefehl löscht die markierte Zeile. Um mehrere Zeilen zu löschen, kannst du entsprechend viele Zeilen im Nummernfeld mit `d` markieren und mit einem Enter löschen.

Folgen solche zu löschenden Zeilen unmittelbar aufeinander, so kannst du in der Form

```
dn
```

mit der passenden Zahl `n` die `n` Zeilen, beginnend mit der markierten, auf einmal löschen. Zwischen `d` und der Anzahl dürfen keine Leerzeichen stehen.

Da im Nummernfeld Ziffern stehen, musst du auf folgendes achten: Gibst du in einer Zeile im Nummernfeld etwa `d3` ein, ohne eine andere Spalte des Nummernfeldes zu ändern, so erkennt *exaEdit* den Befehl

`d3`, ganz gleich, ob oder welche Ziffern nach der 3 stehen. Hast du aber irgendwelche anderen Spalten geändert (ganz gleich, ob du den alten Inhalt wiederhergestellt hast oder nicht), so könnten zwischen *exaEdit* und dir Missverständnisse auftreten, wieviele Zeilen zu löschen sind. Um sicher zu gehen, muss entweder die gewünschte Anzahl am rechten Rand des Nummernfeldes stehen oder du musst nach der Anzahl ein Leerzeichen tippen.

`dd` Dieser Präfixbefehl tritt als Paar auf. Mit dem anschließenden Enter werden alle Sätze vom ersten bis zum zweiten `dd` gelöscht, die markierten Sätze eingeschlossen. Hast du nur einen Satz mit `dd` markiert, so wird diese Marke bei einem Enter nicht beachtet.

Du kannst auch mehrere Paare von `dd`–Präfixbefehlen eingeben und mit einem Enter ausführen lassen. Ist die Anzahl der `dd`–Marken ungerade, so bleibt auch hier das letzte `dd` unbearbeitet.

Mit dem Befehl `MARK` kannst du dir ein einzelnes `dd` anzeigen oder löschen lassen (nur die Marke, der Satz bleibt erhalten).

Kapitel 4

Das `exaEdit`–Lexikon

Hier sind in alphabetischer Reihenfolge Editierfunktionen aufgelistet, zu denen dann jeweils erklärt wird, mit welchen Methoden sie ausgeführt werden können.

Bitte achte im folgenden auf die verwendete Schriftart.

Befehle werden in äquidistanter Schrift mit Groß– und Kleinbuchstaben wiedergegeben. Der in großen Buchstaben geschriebene Teil des Befehlsnamens ist die Minimalabkürzung. Die Eingabe kannst du selbstverständlich beliebig in großen oder kleinen Buchstaben tippen.

Mit kursiver (ebenfalls äquidistanter) Schrift werden Teile eines Befehls (Namen, Nummern, Zeichenfolgen usw.) geschrieben, für die du eigene Zeichen einsetzen musst.

In eckige Klammern [ ] werden Angaben eingeschlossen, die du wahlweise machen oder weglassen kannst.

Ändern der Darstellung

- Befehl `HEXa` für hexadezimale aktuelle Zeile.
- Befehl `CODepage` (mit Parametern) für deutsche Sonderzeichen in Windows–Betriebssystemen.
- Befehl `LWwidth n` zur Beeinflussung des Zeilenumbruchs langer Sätze.
- Befehl `SKey n` zur Festlegung der Breite des Nummernfeldes.
- Befehl `SCope OFF|ON` zum Hin– und Herschalten zwischen Fenster– und Zeilenmodus.

Ändern der Sprache

- Befehl `LAnguage`.

Aufspalten eines Satzes

- Befehl `SSplit` (mit Parametern).

Befehlseparator ändern

- Befehl `CMDsep`.

Benutzeranleitung sehen

- Befehl `MANUAL`.

Blättern

- Um 1 Seite mit den Tasten Bild ↑ und Bild ↓ .
- Um 1 Seite mit den Tasten F7 und F8 .
- Um 1/2 Seite mit den Tasten F10 und F11 .

Einfügen einer Leerzeile

- siehe Leerzeile einfügen

Hilfetexte ansehen

- Befehl `Help` für eine Befehlsübersicht.
- Befehl `Help cmd` für eine Kurzhilfe zum Befehl `cmd`.
- Befehl `MANual` für die ganze *exaEdit*–Benutzeranleitung.

Kopieren von Sätzen

- Befehl `COpy`.
- Präfixbefehl `"` verdoppelt den markierten Satz.

Kopieren von Spalten

- Befehl `CCopy`.

Leerzeile einfügen

- Befehl `SSplit` erzeugt Leerzeile vor der aktuellen Zeile.
- Befehl `SSplit` *n* mit hinreichend großem *n* erzeugt Leerzeile nach der aktuellen Zeile.
- Befehl `Input //` erzeugt eine Leerzeile nach der aktuellen Zeile.
- In den Eingabemodus gehen und eine Zeile eingeben, in der nur ein Leerzeichen getippt wurde.
- Präfixbefehl `i` erzeugt eine Leerzeile nach der markierten.

Löschen aller Sätze eines workfiles

- Befehl `DElete All`.

Löschen eines Satzes

- Präfixbefehl `d` löscht den Satz der markierten Zeile.
- Befehl `DElete` löscht den Satz der aktuellen Zeile.
- Befehl `DL` *num1* löscht den Satz mit der Nummer *num1*.

Löschen eines workfiles

- Befehl `WF` *wfname* `Del`.

Löschen mehrerer Sätze

- Präfixbefehl `d`*n* löscht *n* Sätze ab dem Satz der aktuellen Zeile.
- Präfixbefehl `dd` löscht den Bereich damit markierter Sätze.
- Befehl `DElete` *nnn* löscht *nnn* Sätze ab dem Satz der aktuellen Zeile.
- Befehl `DL` *num1* *num2* löscht die Sätze von *num1* bis *num2*.

Löschen von Spalten

- Befehl `CDelete`.

Markieren eines Satzes

- Befehl `SET`.

Satz einfügen

- Befehl `Input /zeile/`.
- In den Eingabemodus gehen (Befehl `Input`), dann eine Zeile eingeben, dann durch nochmaliges Enter den Eingabemodus wieder verlassen.

Satz löschen

- Präfixbefehl `d` löscht den Satz der markierten Zeile.
- Befehl `DElete` löscht den Satz der aktuellen Zeile.
- Befehl `DL` *num1* löscht den Satz mit der Nummer *num1*.

Sätze löschen

- Präfixbefehl `d`*n* löscht *n* Sätze ab dem Satz der aktuellen Zeile.
- Präfixbefehl `dd` löscht den Bereich damit markierter Sätze.

- Befehl `DElete` *nnn* löscht *nnn* Sätze ab dem Satz der aktuellen Zeile.
- Befehl `DL` *num1 num2* löscht die Sätze von *num1* bis *num2*.

Sätze verketten

- Befehl `CONcat` verkettet den aktuellen Satz mit dem folgenden.

Sortieren einiger oder aller Sätze eines workfiles

- Befehl `SORT` (evtl. mit Parametern).

Spalten ersetzen

- Befehl `Change`.
- Befehl `Replace`.

Sprache wechseln

- siehe Ändern der Sprache.

Suchen von Zeichen[ketten]

- Befehl `Locate` zum Vorwärtssuchen.
- Befehl `RLocate` zum Rückwärtssuchen.
- Befehl `NLocate` zum Aufsuchen von Zeilen ohne Zeichen[kette].
- Befehl `NRLocate` oder `RNLocate` zum rückwärtigen Aufsuchen von Zeilen ohne die Zeichen[kette].

Tabulatorzeichen auflösen

- Befehl `EXpand` expandiert in den angegebenen Sätzen die Tabulatorzeichen zu Leerzeichen.

Tabulatorzeichen setzen

- Befehl `COMpress` komprimiert in den angegebenen Sätzen geeignete Folgen von Leerzeichen zu Tabulatorzeichen.

Verdoppeln eines Satzes

- Ist der Satz in der aktuellen Zeile, so genügt der Befehl `COpy`.
- Präfixbefehl `"`.

Verketten zweier Sätze

- Befehl `CON` verkettet den aktuellen Satz mit dem folgenden.

Verschieben von Sätzen

- Befehl `Move`.

Verschieben von Spalten

- Befehl `CMove`.

Vertauschen zweier Sätze

- Ist der 2. Satz in der aktuellen Zeile, so genügt der Befehl `Move P` (`P` steht für „previous“).

Zählen von Sätzen

- Befehl `COUnt num1 [num2]`.
- siehe auch Gesamtzahl in der Statuszeile.

Zeile einfügen

- siehe Satz einfügen

Zeile löschen

- siehe Satz löschen.

Zeilen löschen

- siehe Löschen aller Sätze eines workfiles.
- siehe Sätze löschen.

Kapitel 5

Die `exaEdit`–Meldungen

Hier findest du in alphabetischer Reihenfolge die meisten Meldungen, die *exaEdit* erzeugen kann, und dazu die Seite(n), auf denen sie näher beschrieben sind.

Seitenzahlen bis 32 einschließlich beziehen sich auf das Kapitel 2, *Erste Schritte*, während Seitenzahlen ab 33 zum Kapitel 3, *Der Editor und seine Befehle* gehören.

`... Dateien geladen`
40

`... Unterverzeichnisse übergangen`
40

`1 Datei geladen`
40

`1 Unterverzeichnis übergangen`
40

`Abbruch bei rekursivem X`
62, 117

`Abbruch bei rekursivem Y`
62, 117

`access errno = ...`
38, 44

`ACHTUNG: Daten nicht gesichert!`
26, 43

`Alte Datei, drücke J oder Y, um sie zu ersetzen:`
23, 42, 64

`Änderungen nicht gesichert`
26, 44, 88, 104

`Anfangsspalte größer als Endspalte`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  In einem Befehl wird ein Spaltenbereich benötigt. Hierbei muss immer erst die linke Spalte und dann die rechte Spalte angegeben werden. Hast du es umgekehrt gemacht, kommt es zur Fehlermeldung. Beachte, dass die Spaltenangaben auch Parametervariable sein können.

`Bus error`
71

`CODEPAGE ist nur für Windows-Systeme`
83

`compress #...`
83

`Curses: ..., Zeichen: ..., Escape: ..., Funktion: ...`
69

`Datei kann nur gelesen werden`
43

`Datei nicht gefunden`
37, 43

`Datei nicht geöffnet (nicht da?)`
38

`Dateisystem kann nur gelesen werden`
43

`Daten gesichert`
23, 43, 71

`Datenanfang`

Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

Ein *exaEdit*–Positionierungsbefehl, beispielsweise `UP`, verlangt die Positionierung der current line vor die erste Zeile des workfiles. Im Gegensatz zum Suchen (siehe Befehlsbeschreibung von `RLOCATE`) ist es beim Positionieren aber nicht möglich, über die erste Zeile des workfiles hinweg zur letzten Zeile des workfiles zu springen.

`Datenende`

Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

Diese Fehlermeldung kann zwei Ursachen haben:

Entweder ein *exaEdit*–Positionierungsbefehl, beispielsweise `DOWN`, verlangt die Positionierung der current line hinter die letzte Zeile des workfiles. Im Gegensatz zum Suchen (siehe Befehlsbeschreibung von `LOCATE`) ist es beim Positionieren aber nicht möglich, über die letzte Zeile des workfiles hinweg zur ersten Zeile des workfiles zu springen.

Die zweite mögliche Ursache ist, dass ein Befehl bei der Ausführung versucht, auf Daten hinter dem Ende des workfiles zuzugreifen. Dies kann z.B. bei der Verwendung des Befehls `CHANGE` zusammen mit dem Parameter `n` geschehen.

`Die Spalte 0 gibt es nicht`

Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

In manchen *exaEdit*–Befehlen wird als Parameter eine Spaltenangabe verlangt. Die Spalten werden ab 1 gezählt, eine Spalte 0 ist nicht erlaubt. Beachte, dass auch eine Parametervariable den Wert 0 haben kann und deshalb zu dieser Fehlermeldung führen kann.

`Die top line kannst du nicht ...`

Für `...` kann stehen: `ändern`, `löschen`, `mitsortieren`, `verketten`, `verschieben`.

Diese Fehlermeldungen können bei vielen *exaEdit*–Befehlen auftreten, sie werden aber nur hier beschrieben:

Du hast einen Befehl eingegeben, dessen Ausführung die top line (mit) betrifft, für diese aber nicht ausgeführt werden kann, etwa `copy t 500`. Oft besteht die Berichtigung darin, die symbolische Zeilennummer `t`, die die top line bezeichnet, durch die symbolische Zeilennummer f (= „first“) zu ersetzen, die den ersten Satz der Daten meint.

`Neu numeriert`
52

`Neue Datei, drücke J oder Y, um sie zu erstellen:`
23, 26, 42

`Nicht gesicherte workfiles: ...`
44, 88, 104

`Nummer ... nicht gefunden`
85, 97, 103, 112

`Objekt ist kein Verzeichnis`
40

`Parametervariable keine Zeichenkette`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  Für eine Zeichenkette als Parameter hast du eine Parametervariable angegeben, die zwar für Zahlen oder Zeilennummern, aber nicht für Zeichenketten definiert wurde.

`Parametervariable nicht definiert`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  Parametervariable, die nicht von vornherein Bestandteil von *exaEdit* sind, müssen vor ihrer Verwendung mit dem `&`–Befehl definiert werden.

`Parametervariable nicht numerisch`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  Für eine Zahlenangabe als Parameter (Spalte, Anzahl usw.) hast du eine Parametervariable angegeben, die zwar für Zeichenketten oder Zeilennummern, aber nicht für Zahlen definiert wurde.

`Quell- und Zielbereich überlappen`
78, 82

`REKEY ergibt zu große Zahl`
104

`Schließendes ' fehlt`
37, 43

`Segmentation fault`
71

`SEQUENCE übersteigt 99999999 oder Feldbreite`
110

`SET-Speicher geändert, Rückkehr zum vorhergehenden Satz`
54, 105, 110

`SET-Speicher unbenutzt`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  In einem Befehl, beispielsweise `COPY`, wird die symbolische Satznummer `s` verwendet, die denjenigen Satz bezeichnet, der mit dem Befehl `SET` markiert wurde. Im aktuellen workfile wurde der Befehl `SET` aber noch nicht gegeben (oder danach der gesamte workfile geleert). Dann ist die Angabe `s` unbestimmt, so dass es zur Fehlermeldung kommt. Sie kann übrigens auch von dem Befehl `RETURN` erzeugt werden.

`SET-Speicher ungültig`
: Diese Fehlermeldung kann bei vielen *exaEdit*–Befehlen auftreten, sie wird aber nur hier beschrieben:

  In einem Befehl, beispielsweise `COPY`, wird die symbolische Satznummer `s` verwendet, die denjenigen Satz bezeichnet, der mit dem Befehl `SET` markiert wurde. Im aktuellen workfile wurde der Befehl `SET` zwar

gegeben, danach aber der betreffende Satz gelöscht, so dass die symbolische Satznummer `s` nicht mehr verwendet werden kann. Beachte, dass manche Befehle in einem solchen Fall stattdessen die Meldung `SET-Speicher geändert, Rückkehr zum vorigen Satz` bringen.

`Sorry, I don't know how to deal with your '...' terminal.`
34

`Sorry, I need to know a more specific terminal type than ''.`
34

`Sortierfelder überlappen`
112

`Sortiert`
111

`stat errno = ...`
38, 44

`Suche ab Anfang (wrap)`
30, 94, 99

`Suche ab Ende (wrap)`
31, 101, 106, 108

`Teil des Namens kein Verzeichnis`
37, 43

`TERM nicht definiert`
34

`Terminaltyp ist ...`
34

`Datenanfang`
101, 106, 108

`Übersetzung in Großbuchstaben`
114

`Übersetzung in Kleinbuchstaben`
114

`Ungerade Anzahl von Hex-Zeichen`
80, 95, 100, 102, 107, 109, 113

`Ungültiges Hex-Zeichen`
81, 95, 100, 102, 107, 109, 113

`Upper: mit Übersetzung in Großbuchstaben (ohne Umlaute)`
78

`Verzeichnis kann nicht editiert werden`
37, 43

`Verzeichnis nicht gefunden`
40, 43

`Verzeichnis nicht geöffnet`
40

`WIN`
83

Stichwortverzeichnis

Seitenzahlen bis 32 einschließlich beziehen sich auf das Kapitel 2, *Erste Schritte*, während Seitenzahlen ab 33 zum Kapitel 3, *Der Editor und seine Befehle* gehören.

Stichwortverzeichnis