SIEMENS
NIXDORF

SINIX

HIT V4.1

Textverarbeitungssystem

Systemhandbuch

Sie haben

uns zu diesem Handbuch etwas mitzuteilen?
Schicken Sie uns bitte Ihre Anregungen unter
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG
AP Internationales
Dokumentationszentrum
Otto-Hahn-Ring 6
W-8000 München 83

Fax: (0 89) 6 36-8 21 87

DOPPEL

Sie haben

uns zu diesem Handbuch etwas mitzuteilen?
Schicken Sie uns bitte Ihre Anregungen unter
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG
AP Internationales
Dokumentationszentrum
Otto-Hahn-Ring 6
W-8000 München 83

Fax: (0 89) 6 36-8 21 87

HIT (SINIX)

Textverarbeitungssystem

Systemhandbuch

Ausgabestand Januar 1991 (HIT V4.1)

Wollen Sie mehr wissen ...

. . . über dieses Produkt
. . . oder ein anderes Thema der Informationstechnik?

Unsere Training Center stehen für Sie bereit.
Besuchen Sie uns in Berlin, Essen, Frankfurt/Main oder Hamburg, in Hannover, Mainz, München, Stuttgart, Wien oder Zürich.

Auskunft und Informationsmaterial erhalten Sie über:

München (0 89) 6 36-20 09
oder schreiben Sie an:

Siemens Nixdorf Training Center
Postfach 83 09 51, W-8000 München 83

Bestätigt Konformität zu den Standards des x/Open Portability Guide 3 sowie zu den internationalen Standards für UNIX-Systeme.

Vorwort

Dieses Handbuch ist das Systemhandbuch für das HIT-Textsystem. Es wendet sich an den Systembetreuer.

Das Systemhandbuch beschreibt

- die Programmierung von Textbausteinen mit CLOU (anhand von Beispielen),
- Die Funktionen für den Systemverwalter,
- Die Systemarchitektur

Der Anhang enthält eine Tabelle der Zeichen, die mit HIT darstellbar sind.

Wenn Sie mit den Einfachen Anwendungen von CLOU (Kapitel 1) arbeiten wollen, brauchen Sie gute Kenntnisse des HIT-Textsystems, vor allem der Arbeit mit Textbausteinen.
Für die Kapitel 2 bis 5 brauchen Sie Programmierkenntnisse und sie müssen mit Datenbanksystemen umgehen können.
Für die Kapitel 6 bis 11 müssen Sie die SINIX-Kommandoebene Shell gut kennen.

Bitte unterstützen Sie uns, indem Sie uns Kritik, Wünsche und Vorschläge für Verbesserungen mitteilen.

Änderungsprotokoll

Änderung des Vorgängermanuals
Stand Januar 1989 (HIT (SINIX) V4.0, Systemhandbuch)
durch den Nachtrag vom
Januar 1991 (HIT (SINIX) V4.1, Systemhandbuch)

Der Nachtrag weist gegenüber der Vorgängerversion folgende Änderungen auf:

- CLOU kennt außer den bisherigen Standard-Funktionen zusätzliche Variablen-Funktionen. Neu hinzugekommen sind die Signalbehandlungs- und Sonstige Funktionen.
- Die FETCH-Anweisung bei Datenbank-Zugriffen von CLOU aus wurde erweitert.
- Durch den Anschluß an INFORMIX V4.0 gibt es zusätzliche Feldtypen, die in CLOU verarbeitet werden können.
- CLOU kennt zusätzliche Optionen für automatisches Runden bei Rechenvariablen und für die 8-Bit-Konvertierung beim Datenaustausch mit INFORMIX.
- Das Menü für den Systemverwalter erhält man nicht mehr über das Standard-Menüsystem, sondern über die Bedienoberfläche COLLAGE.
- Es gibt die Möglichkeit, interaktive Druckerskripten zu erstellen.
- Das Druckprogramm 'filter' und der Proportionalschrift-Formatierer 'proform' kennen neue Optionen.
- Bei den Druckertabellen gibt es eine Erweiterung bei dem Schlüsselwort ZEILABST.
- Bei den Dicktentabellen gibt es Besonderheiten bei den Schlüsselworten MITTL_BREITE und ATTRIBUTFONTS.
- Bei einer Parallelinstallation von HIT V4.1 und HIT-COL V4.1 sind getrennte Zentrale Archive möglich.

Änderungsprotokoll

- Die Funktionsauswahl im HIT-MENÜ kann nun um insgesamt 32 Anwendungen erweitert werden.
- Die Syntax, mit der die Funktionsauswahl bei 'Markieren + Bearbeiten, ext-bearbeiten' vereinbart wird, wurde erweitert.
- Es können zusätzliche Funktionen für 'Bereich einfügen, importieren' vereinbart werden.
- Das Drucken von Zeichnungen, die im COLLAGE-Metafile-Format abgelegt sind, erfolgt nun mit dem Programm 'hmetapr'.
- Im gesamten HIT-Textsystem können nun Dateinamen, die Zeichen aus dem 8-Bit-ASCII-Zeichensatz enthalten, verarbeitet werden. Zu diesem Zweck stehen zwei Shell-Variablen zur Verfügung.
- Das Konvertierungsprogramm PROCON kennt zusätzliche Aufruf-Optionen.
- Bei der Konvertierung in TELEX-Format werden jetzt auch deutsche Umlaute und einige Sonderzeichen berücksichtigt.
- Das Modul 'hgrep' kennt eine neue Option, mit der nach Steuerzeilen im HIT-Dokument gesucht werden kann.
- Die Struktur der Programme unter /usr/bin/HIT wurde erweitert.
- Die Vorgänge beim Aufruf des Textsystems HIT haben sich geändert.
- Über eine Shell-Variable kann festgelegt werden, in welchen Dateiverzeichnissen die temporären Dateien abgelegt werden sollen.
- Die Installation des HIT-Systems hat sich geändert.

Die Syntaxübersicht von CLOU für das HIT-Systemhandbuch V4.1 liegt aus produktionstechnischen Gründen diesem Nachtrag nicht bei.
Bitte bestellen Sie sie separat.

Inhalt

1 Einfache Anwendungen

1.1 Textbausteine, HIT und CLOU

Das Baustein-Verarbeitungsprogramm CLOU ermöglicht es Ihnen, in Textbausteine neben einfachem Einfügetext auch Anweisungen aufzunehmen, die später - während des Einfügens in ein Dokument - von HIT bzw. CLOU befolgt werden. CLOU wird damit für Sie zu einer Art Hilfskraft, die Ihnen viele der täglichen Routinearbeiten abnehmen kann.

Zum Beispiel kann CLOU für Sie:

- alle HIT-Funktionen auslösen, wenn Sie ihm die dazu notwendigen Tastenfolgen sagen;
- Tabellen erstellen;
- das aktuelle Tagesdatum in beliebigem Format in den Text einfügen;
- rechnen, z.B. Summen bilden und die Mehrwertsteuer hinzuaddieren;
- auf INFORMIX-Datenbanken zugreifen;
- SINIX-Kommandos aufrufen;
- SINIX-ASCII-Dateien lesen und schreiben

und vieles mehr.

Es gibt einfache Anweisungen, die leicht zu erlernen sind und mit denen Sie schon viel erreichen können. Auf der anderen Seite gibt es aber auch kompliziertere Anweisungen, die z.B. Kenntnisse über Datenbanken voraussetzen.

- Kapitel 1 erklärt Wirkungsweise und Verwendung von CLOU und führt darüber hinaus bereits eine hinreichend große Zahl einfacher CLOU-Anweisungen ein, mit denen Sie sinnvolle CLOU-Anwendungen realisieren können.

- Wenn Sie die Erleichterungen, die das Arbeiten mit den einfachen CLOU-Anweisungen mit sich bringt, zu schätzen gelernt haben, dann wollen Sie sich vielleicht auch mit komplexeren Verwendungen von CLOU beschäftigen. In diesem Fall finden Sie die benötigten Informationen in Kapitel 2 und 3.

- Das Kapitel 4 schließlich ist für jene Anwender gedacht, die bereits Erfahrung mit SINIX bzw. INFORMIX mitbringen und deren Möglichkeiten auch der Textverarbeitung zugänglich machen wollen.

Wie Sie diese Beschreibung verwenden können

Die vorliegende CLOU-Beschreibung soll Ihnen in zweifacher Hinsicht nützlich sein:

- Als **Einführung** sollte man sie der Reihe nach, Kapitel für Kapitel lesen. Die Reihenfolge ist so gewählt, daß eines auf dem anderen aufbaut. Die zahlreichen in den Text eingestreuten Beispiele können Sie gleich während der Lektüre am Bildschirm nachvollziehen: gerade wenn etwas unklar geblieben ist, kann nichts diese direkte Anschauung ersetzen.
- Als **Nachschlagewerk** ist die Beschreibung übersichtlich gegliedert, so daß Sie über das Inhaltsverzeichnis schnell die gesuchte Stelle auffinden. Wenn Sie bereits mehr Übung haben, dann wird Ihnen die tabellarische "HIT Syntaxübersicht (CLOU)" nützlich sein. Sie benötigen das Handbuch dann nur noch in Zweifelsfällen.

Dieses einführende Kapitel erläutert die Arbeitsweise von CLOU und sein Zusammenspiel mit HIT. Außerdem erfahren Sie hier, wie Sie CLOU-Textbausteine erstellen und was diese Ihnen alles an Arbeit abnehmen können. Dieses Kapitel setzt voraus, daß Sie einfache formatfreie und formatierte Textbausteine erstellen und in Dokumente einfügen können. Im Zweifelsfall sollten Sie sich damit anhand des HIT-Benutzerhandbuchs vertraut machen, bevor Sie an die Lektüre dieses Kapitels gehen.

Um die Wirkungsweise von CLOU genau verstehen zu können, sollte man sich zunächst die normale Situation beim Arbeiten mit HIT vor Augen halten. (Bild 1-1) Sie geben Ihren Text sowie verschiedene Anweisungen an HIT (z.B. Bewegen der Schreibmarke, Absatz formatieren usw.) über die Tastatur ein. HIT fügt den Text dort in das Dokument ein, wo die Schreibmarke gerade steht und befolgt Ihre Anweisungen. Das Ergebnis sehen Sie unmittelbar auf dem Bildschirm.

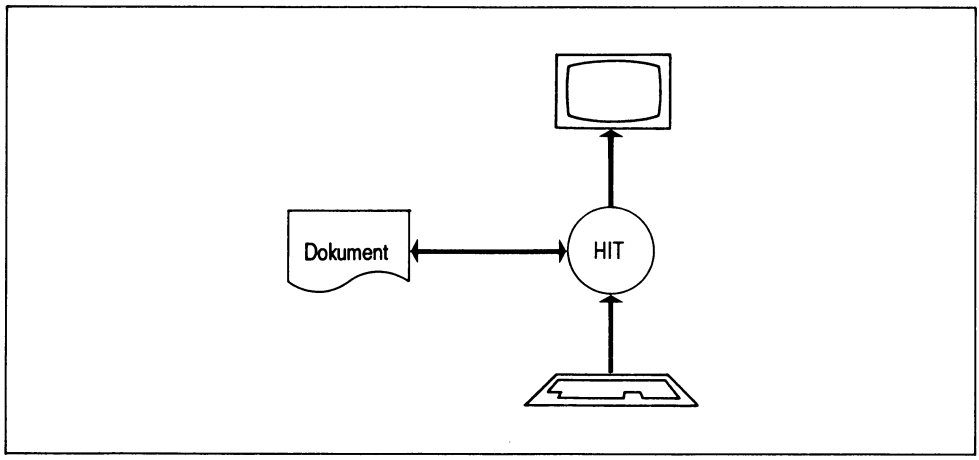


Bild 1-1: Das Arbeiten mit HIT

Sie machen also zwei verschiedene Sorten von Eingaben:

- **Texteingaben**. Der neue Text paßt sich automatisch dem aktuellen Zeilenlineal an, indem die Schreibmarke an den Anfang einer neuen Zeile springt, wenn der rechte Rand des Zeilenlineals erreicht ist.
- **Anweisungen durch Tastendruck**. Wenn Sie eine Pfeil- oder Funktionstaste drücken, weisen Sie HIT damit an, eine bestimmte Aktion durchzuführen: z.B. die Schreibmarke eine Zeile nach oben bewegen.

In den folgenden Kapiteln wird nun untersucht, wie sich die Gegebenheiten ändern, wenn einfache bzw. CLOU-Textbausteine ins Spiel kommen.

1.1.1 Einfache Textbausteine

Einfache Textbausteine sind Dokumente, die - einmal erstellt - beliebig oft in andere Dokumente eingefügt werden können. Wenn Sie die Taste F12 **Baustein einfügen** gedrückt, einen Bausteinnamen und ↵ eingegeben haben, dann ändert sich die Situation aus Bild 1-1 für die Dauer des Einfügevorgangs, wie in Bild 1-2 gezeigt:

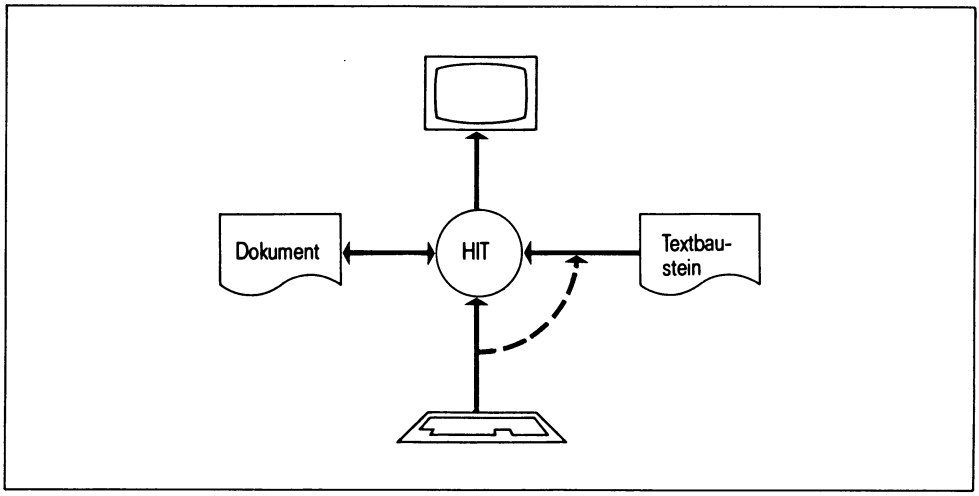


Bild 1-2: Das Einfügen eines einfachen Textbausteins

Der Text kommt also für eine Weile nicht mehr von der Tastatur, sondern aus dem Textbaustein. In der Wirkung kommt das einem unsichtbaren "Phantomschreiber" gleich, der nun an Ihrer Stelle die Tastatur bedient und damit den Text erstellt. Der "Phantomschreiber" kann aber eben nur Text eingeben, Funktions- und Pfeiltasten kann er für Sie nicht drücken.

Stellen Sie sich nun folgenden Fall vor: Sie haben einige Dokumente, die jeweils lediglich noch um die Überschrift RECHNUNG zu ergänzen sind. Dabei soll die Überschrift jedesmal zentriert, also in der Mitte der Zeile ausgegeben werden. Diese Ergänzung können Sie vornehmen, indem Sie einen Textbaustein erstellen, der nur das Wort "RECHNUNG" enthält. Wie wollen Sie aber erreichen, daß die Überschrift tatsächlich jedesmal zentriert erscheint? Jedes Dokument hat ja sein eigenes Format - bestimmt durch das Zeilenlineal.

Es wird Ihnen nichts anderes übrig bleiben, als jedesmal nach dem Baustein-Einfügen selbst die Tasten SHIFT + F10 **Absatzattribute** und F10 **Absatz formatieren** zu drücken. In dieser Situation wäre es angenehm, die Anweisung "zentriere die Zeile" gleich mit in den Baustein hinein schreiben zu können. Genau das ist mit CLOU-Textbausteinen möglich.

1.1.2 CLOU-Textbausteine

CLOU-Bausteine sind zunächst einmal ganz normale Textbausteine, d.h. Sie können sie mit HIT wie gewohnt als formatfrei oder formatgebunden erstellen und in einem Bausteinordner Ihrer Wahl ablegen. Formatgebundene CLOU-Bausteine - also solche, die Zeilenlineale enthalten - bringen diese Zeilenlineale beim Einfügen ins Zieldokument wie gewohnt mit. Formatfreie Bausteine passen sich dem aktuellen Zeilenlineal des Doku-ments an, in das sie eingefügt werden.

Darüber hinaus haben CLOU-Bausteine die folgenden Merkmale:

- Das **erste Zeichen in der ersten Zeile** muß das sog. Nummernzeichen # sein. Daran erkennt HIT später beim Einfügen, daß es sich um einen CLOU-Baustein handelt.
- Wie normale Textbausteine können auch CLOU-Bausteine einfachen **Text** enthalten, der beim Einfügen ins Zieldokument übernommen wird.
- Neben solchem Text können CLOU-Bausteine auch **Anweisungen** enthalten, die ebenfalls durch ein einleitendes Nummernzeichen gekennzeichnet werden. Diese Anweisungen werden nicht in das Zieldokument eingefügt, sondern von CLOU interpretiert und ausgeführt.

Ein CLOU-Baustein für das Einfügen einer zentrierten Überschrift könnte z.B. so aussehen:

Beispiel:

```
#
#^ "ZNT"

RECHNUNG●

#^ "LBD"
```

Wenn Sie diesen Beispieltext als formatfreien Baustein erstellen und dann in ein beliebiges Dokument einfügen, so erscheint das Wort RECHNUNG zentriert. Der nächste Abschnitt beschreibt, was während des Einfügevorgangs im einzelnen geschieht.

1.1.3 Wie CLOU und HIT zusammenarbeiten

Die Situation während des Einfügens von CLOU-Bausteinen zeigt Bild 1-3:

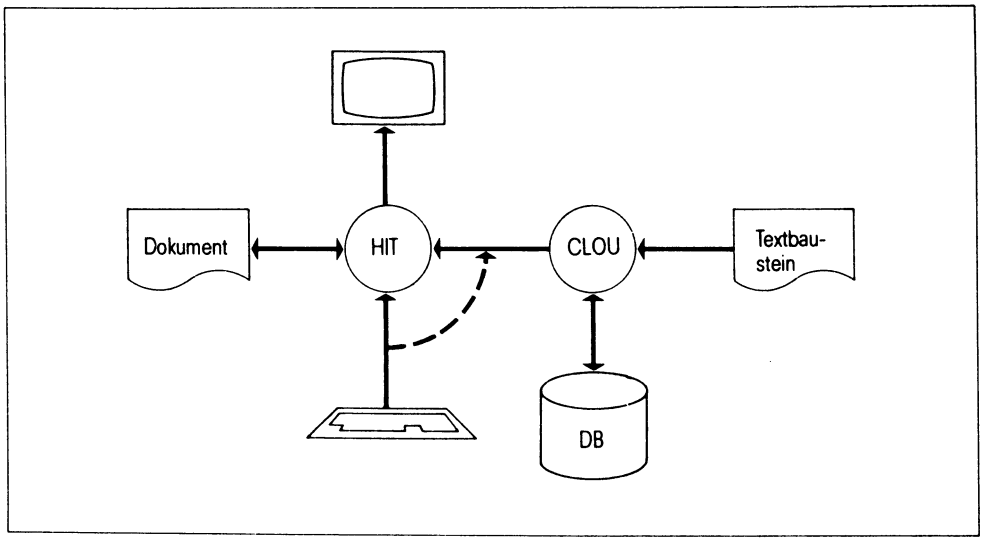


Bild 1-3: Das Einfügen eines CLOU-Bausteins

Wenn Sie HIT anweisen, einen CLOU-Baustein einzufügen, so wird das Baustein-Verarbeitungsprogramm CLOU aktiviert. Dieses übernimmt nun die Regie für die Dauer des Einfügevorgangs. Das "Drehbuch" stammt aber von Ihnen: beim Erstellen eines CLOU-Bausteins schreiben Sie - neben gewöhnlichem Einfügetext - auch Anweisungen an CLOU, z.B. daß eine bestimmte Zeile zentriert werden soll.

Jedesmal, wenn Sie danach einen solchen Baustein in ein Dokument einfügen, liest CLOU Ihren Text und gibt ihn an HIT weiter. Findet er nun eine Anweisung (erkenntlich an dem Zeichen #), so führt er diese aus.

Hinweis:

Bevor CLOU mit dem Einfüge-Vorgang beginnt, verändern sich automatisch einige der Einstellungen, die Sie normalerweise über Funktionstasten, Modus-Menü oder die Umgebungstabelle beeinflussen können. Dies ist notwendig, damit ein einmal erstellter CLOU-Baustein immer in der gleichen Weise abläuft - unabhängig von den Einstellungen, die Sie vorher in HIT gewählt haben.

Eingeschaltet werden:

- der Einfügemodus und
- automatische Absatzformatierung.

Ausgeschaltet werden:

- Automatisches Trennen,
- Absatzmarke mit [↵] sowie
- sämtliche Text-Attribute.

Die von Ihnen gewählten Einstellungen werden nach dem Baustein-Einfügen wiederhergestellt.

1.1.4 Wie ein CLOU-Baustein aussieht

Zur Unterscheidung von normalen Textbausteinen müssen CLOU-Bausteine in der ersten Zeile allein ein #-Zeichen enthalten. Dieses erscheint später nicht im Zieldokument. Eine oder mehrere Zeilen darunter beginnt der Einfügetext, in den beliebig CLOU-Anweisungen eingestreut werden können. In der Verarbeitung dieses Textes gibt es nun einen weiteren wichtigen Unterschied zu normalen Textbausteinen.

Beispiel 1:

```
#

Wie Sie an dem Nummernzeichen in der ersten Zeile
erkennen,

bin
ich
ein
CLOU-Baustein.
```

Wenn Sie diesen Text als formatfreien Textbaustein ablegen und dann von HIT aus in ein Dokument einfügen, so erhalten Sie folgendes Ergebnis:

```
>-------------------------------------------------<
 Wie Sie an dem Nummernzeichen in der ersten Zeile er-
 kennen, bin ich ein CLOU-Baustein.
```

CLOU hat also alle Zeilenvorschübe ignoriert bzw. durch Leerzeichen ersetzt! CLOU verhält sich so, als würden Sie ihm den Bausteintext diktieren: er fügt Wort für Wort in Ihr Dokument ein. Wenn Sie an irgendeiner Stelle eine neue Zeile haben wollen, so müssen Sie das explizit sagen. Das kann z.B. mit der Absatzmarke geschehen:

Beispiel 2:

```
#
●
Wie Sie an dem Nummernzeichen in der ersten Zeile
erkennen,●
●
bin●
ich●
ein●
CLOU-Baustein.●
```

Wenn Sie diesen Baustein einfügen, dann bleiben die Zeilenvorschübe erhalten, und der Einfügetext sieht so aus wie der Baustein selbst - bis auf das # am Anfang.

Die Eigenschaft von CLOU, den Bausteintext als **Fließtext** aufzufassen (siehe Beispiel 1), erweist sich als sehr nützlich. Dadurch ist es möglich, komplexe Bausteine mit vielen Anweisungen übersichtlich zu gestalten: Sie können überall, wo eine solche Auflockerung die Lesbarkeit erhöht, beliebig viele Leerzeilen einstreuen, ohne daß dies im Zieldokument sichtbar wird. Die Beispiel-Bausteine in den nächsten Kapiteln machen davon intensiven Gebrauch.

1.2 Einfache Anweisungen (Teil I)

CLOU-Bausteine können bestehen aus

- Text, der in das Zieldokument übernommen wird, und
- Anweisungen, die von CLOU ausgeführt werden, wann immer er während eines Einfügevorgangs auf sie stößt.

Dieses Kapitel macht Sie mit einigen einfachen CLOU-Anweisungen bekannt. Wie alle weiteren Kapitel ist auch dieses in mehrere Abschnitte unterteilt; im allgemeinen werden Sie in jedem Abschnitt eine neue CLOU-Anweisung kennenlernen.

> *Hinweis:*
>
> Den nächsten Abschnitt, "Einfügen eines Textbausteins", sollten Sie unbedingt als ersten lesen, da an diesem Beispiel auch wichtige allgemeine Begriffe erläutert werden!

1.2.1 Einfügen eines Textbausteins

Der einfachste Grund für die Verwendung von Textbausteinen ist die Arbeitsersparnis: häufig vorkommende Textbestandteile können einmal in Bausteinen abgelegt und bei Bedarf an beliebiger Stelle in Ihr Dokument eingefügt werden. Selbst das Einfügen von Textbausteinen kann aber zu einer lästigen Arbeit werden, wenn z.B. drei verschiedene Textbausteine häufig in der gleichen Kombination vorkommen.

Ein einfacher Textbaustein mit dem Namen `briefkopf` könnte z.B. so aussehen:

```
Fa. Maier GmbH●
Rübezahlweg 33●
8888 Hintertupfelfing●
●
Tel.: (08888) 1234●
```

Zusätzlich könnten Sie einen Baustein `mahnung` geschrieben haben, der etwa so aussieht:

```
Zahlungserinnerung●
●
Unsere  Rechnung vom ..1987 ist Ihrer Aufmerksamkeit
bisher entgangen.  Wir  erlauben  uns  hiermit  höf-
lichst, Sie an unsere  Lieferung von ... Rasenmähern
zu erinnern.●
```

Schließlich haben Sie noch einen Baustein `briefende`:

```
Hintertupfelfing, den ..1987,●
●
mit freundlichen Grüßen●
●
●
.................●
   (A. Müller)●
```

Jedesmal, wenn Sie nun eine Mahnung schreiben wollen, müssen Sie alle drei Bausteine der Reihe nach in Ihr Dokument einfügen. Diese Arbeit kann CLOU Ihnen abnehmen, wenn Sie einen CLOU-Baustein z.B. mit dem Namen `mahnbrief` anlegen, der dann so aussieht:

Beispiel:

```
#

#B "briefkopf"
#B "mahnung"
#B "briefende"
```

Das #-Zeichen in der ersten Zeile kennzeichnet diesen Textbaustein als CLOU-Baustein. Die mit `#B` eingeleiteten Zeilen weisen CLOU an, den jeweils genannten Baustein in Ihr Zieldokument einzufügen. Anstatt also alle drei Bausteine einzeln aufzurufen, brauchen Sie nur noch den CLOU-Baustein `mahnbrief` in Ihr Dokument einzufügen: Alles weitere erledigt CLOU für Sie.

Wie dieses Beispiel zeigt, bestehen CLOU-Anweisungen im allgemeinen aus folgenden Komponenten:

- Das **Einleitungszeichen**. Am Anfang jeder Anweisung muß ein #-Zeichen stehen, damit CLOU den nachfolgenden Text als Anweisung (und nicht als normalen Einfügetext) interpretiert.
- Der **Kennbuchstabe**. Unmittelbar hinter dem Einleitungszeichen (d.h. ohne Leerzeichen) muß ein Kennbuchstabe stehen. An ihm erkennt CLOU, *was* er tun soll. Im obigen Beispiel ist dies das 'B': CLOU erkennt daran, daß er einen Baustein einfügen soll. Der Kennbuchstabe muß großgeschrieben werden. Manche Anweisungen benutzen anstelle eines Buchstaben auch andere Zeichen, wie z.B. *, + usw.
- **Parameter**. Viele Anweisungen benötigen darüber hinaus einen oder mehrere Parameter. Das sind Angaben, denen CLOU z.B. entnimmt, womit er eine bestimmte Aktion durchführen soll: im Beispiel-Baustein steht nach dem ersten #B der Bausteinname in Anführungszeichen.

  Die Anführungszeichen (") dürfen nicht fehlen. Sie gehören streng genommen nicht zum Parameter selbst, sondern zu der CLOU-Anwei-sung: an Ihnen erkennt CLOU, wo der Parameter anfängt und wo er aufhört. Soll der Parameter selbst Anführungszeichen enthalten, so müssen diese besonders gekennzeichnet werden; darauf wird noch einmal ausführlich in Kapitel 1.2.3 eingegangen.

Bausteine, die Sie mit der #B-Anweisung einlesen lassen, können selbst auch wieder CLOU-Bausteine sein. Als solche können sie wieder Anweisungen enthalten - auch weitere #B-Anweisungen. Man spricht dann von verschachtelten Baustein-Aufrufen. Damit ist es möglich, ganze Baustein-Hierarchien aufzubauen.

Hinweis:

Ruft ein Baustein sich selbst auf, enthält er also eine #B-Anweisung mit seinem eigenen Namen als Parameter, so würde dies im Prinzip eine unendliche Kette von Einfüge-Vorgängen auslösen. Aus technischen Gründen ist die Verschachtelungstiefe für Baustein-Aufrufe aber begrenzt. Die Grenze ist systemabhängig und liegt bei ca. 50. Wird dieser Wert überschritten, so bricht CLOU mit einer entsprechenden Fehlermeldung den Einfüge-Vorgang ab.

Anweisungs-Syntax

In den folgenden Kapiteln finden Sie zu jeder CLOU-Anweisung ihre sog. "Syntax":

Syntax

#B "*Bausteinname*"

Die Syntax beschreibt in sehr kurzer Form genau, aus welchen Elementen eine bestimmte Anweisung aufgebaut ist. Buchstaben und Zeichen, die in der Syntax in normalem Schrifttyp erscheinen, müssen genau so in die Anweisung geschrieben werden (im Beispiel oben sind dies das B sowie die Anführungszeichen). Worte in Kursivschrift hingegen müssen Sie durch eigene Angaben ersetzen: Sie schreiben nicht *Bausteinname* sondern zum Beispiel `briefkopf`.

Es gibt noch zwei Varianten der `#B`-Anweisung. Die erste ermöglicht das Einfügen von Textbausteinen aus anderen Ordnern als aus dem in HIT-MENÜ eingestellten:

Syntax

#B "*Ordnername*/*Bausteinname*"

Für *Ordnername* kann der Name eines beliebigen Ordners im eingestellten Bausteinarchiv angegeben werden. Zwischen *Ordnername* und *Bausteinname* muß der Schrägstrich / stehen. Wenn Sie die Angabe von *Ordnername* fortlassen, so benutzt CLOU den jeweils zuletzt verwendeten (bzw. den voreingestellten) Ordnernamen.

Wenn ein Teil der Anweisung wie in diesem Fall optional ist, d.h. daß Sie ihn angeben können aber nicht *müssen*, wird er künftig in eckige Klammern gesetzt. In dieser Form sieht die Syntax für die `#B`-Anweisung so aus:

Syntax

#B "[*Ordnername*/]*Bausteinname*"

Die eckigen Klammern dürfen Sie nicht in die Anweisung schreiben! Sie erkennen daran lediglich: Sie können einen Ordnernamen angeben, müssen aber nicht.

Ein Doppelpunkt vor dem Ordner- bzw. Bausteinnamen bewirkt, daß der entsprechende Baustein im "Rohformat" eingefügt wird. Das heißt, daß die in ihm enthaltenen Anweisungen nicht befolgt werden, sondern der ganze Bausteintext unverändert ins Zieldokument übernommen wird. Die Syntax für die #B-Anweisung lautet also in vollständiger Form:

Syntax

#B "[:][*Ordnername*/]*Bausteinname*"

Diese Syntax ist so zu lesen:

- Die Anweisung muß mit #B anfangen.
- Darauf muß das Anführungszeichen folgen.
- Dann kann ein Doppelpunkt gesetzt werden, muß aber nicht.
- Ebenso kann ein beliebiger Ordnername folgen. Schreibt man den Ordnernamen in die Anweisung, so muß der Schrägstrich gesetzt werden: die Angaben innerhalb eines Klammerpaares müssen entweder alle fortgelassen oder alle eingesetzt werden.
- Schließlich muß ein Bausteinname, gefolgt von abschließenden Anführungszeichen eingesetzt werden.

Zusammenfassend kann über die Anweisungs-Syntax gesagt werden:

- Alle Angaben, die nicht kursiv gesetzt sind, müssen genau so in den Anweisungstext übernommen werden. Wo ein Leerzeichen steht, können aber beliebig viele Leerzeichen und sogar Zeilenvorschübe eingesetzt werden.
- Kursiv gesetzte Anweisungsteile müssen durch Ihre eigenen Angaben ersetzt werden.
- Anweisungsteile, die in eckigen Klammern stehen, können auch fortgelassen werden. Die eckigen Klammern selbst gehören nicht zur Anweisung.

1.2.2 Kommentare

Damit Ihre CLOU-Bausteine gut lesbar und übersichtlich sind, sollten Sie diese ausführlich kommentieren. **Kommentare** sind Erklärungen, die an beliebigen Stellen in einen Baustein eingestreut werden können. Komplexere CLOU-Bausteine werden durch sie leichter verständlich, so daß Sie sich zu einem späteren Zeitpunkt, wenn Sie z.B. eine Änderung an einem Baustein vornehmen wollen, leichter darin zurechtfinden. Damit sie von CLOU nicht als Einfügetext interpretiert werden, müssen Kommentare als solche gekennzeichnet werden:

Syntax

Kommentar #*

Für *Kommentar* können Sie jeden beliebigen Text einsetzen, der sich auch über mehrere Zeilen erstrecken darf. Das abschließende #-Zeichen dürfen Sie auf keinen Fall vergessen: CLOU würde sonst den gesamten Text Ihres Bausteins bis zum nächsten # überlesen.

Mit Hilfe von Kommentaren können Sie z.B. jeden CLOU-Baustein mit einem Titel versehen, aus dem sein Verwendungszweck hervorgeht. Das Beispiel aus dem vorigen Abschnitt könnte dann so aussehen:

Beispiel:

```
#

#* Mahnschreiben aus drei Textbausteinen
#

#B "briefkopf"
#B "mahnung"
#B "mfg"
```

Wenn Sie diesen Textbaustein in Ihr Dokument einfügen, geschieht nichts anderes als in dem Beispiel aus dem vorigen Abschnitt: CLOU ignoriert alles, was zwischen #* und dem nächsten # steht. In den Beispiel-Bausteinen in diesem Handbuch werden Sie zahlreiche Kommentare finden, die Ihnen die Orientierung erleichtern.

1.2.3 Manuelle Texteinfügung

Häufig werden Sie nach dem Aufruf eines Bausteins am soeben eingefügten Text noch Änderungen vornehmen wollen; z.B. wenn der Baustein noch Leerstellen für den Namen eines Kunden, das Datum oder ähnliches enthält. Damit Sie keine dieser Stellen vergessen, können Sie CLOU anweisen, Sie daran zu erinnern. CLOU unterbricht dann an der richtigen Stelle den Einfügevorgang und sagt Ihnen, welcher Text dort vorgesehen ist. Daraufhin können Sie z.B. einen Namen eingeben, ohne selbst die Schreibmarke an die gewünschte Position bewegen zu müssen.

Es gibt zwei Arten dieser manuellen Texteinfügung:

- die **Freie Texteinfügung**

  Sie ist für kurze Eingaben, z.B. Namen, Adressen etc. gedacht und kann neben Text auch CLOU-Anweisungen enthalten.

- die **Variable Texteinfügung**

  Sie ermöglicht dem Benutzer längere Eingaben, z.B. ganze Briefteile. Hier ist im Gegensatz zur **Freien Texteinfügung** auch die Verwendung von Pfeil- und Funktionstasten möglich.

Freie Texteinfügung

Eine große Anzahl von Geschäftsbriefen folgt einem starren Aufbau. Manchmal müssen nur Anrede und Name des Empfängers eingesetzt werden, um aus einem "Standardtext" einen "persönlichen Text" zu machen. In solchen Fällen bietet es sich an, den Brief als Textbaustein abzulegen. An den Stellen, die Sie variabel halten wollen, setzen Sie dann die Einfüge-Anweisung ein:

Syntax

#F ["*Meldetext*"]

Für *Meldetext* können Sie beliebigen Text einsetzen, wobei Sie die Anführungszeichen nicht vergessen dürfen. Die eckigen Klammern zeigen Ihnen, daß Sie den *Meldetext* auch fortlassen können; CLOU verwendet dann eine eigene Standardmeldung.

Wenn CLOU beim Einfügen eines Bausteins auf diese Anweisung trifft, geschieht folgendes: CLOU zeigt Ihnen den *Meldetext* (bzw. die Standardmeldung) in der Kommandozeile und fordert Sie damit zur Eingabe auf. Sie können nun beliebigen Text eingeben - einschließlich weiterer CLOU-Anweisungen (siehe unten!). Mit ↵ schließen Sie Ihre Eingabe ab. CLOU fügt den Text dann an der aktuellen Schreibmarkenposition in Ihr Dokument ein.

Beispiel:

```
#

#* Beantwortung einer Produktanfrage
#

#F "Bitte Anrede eingeben:"
●
●
Vielen Dank für Ihr Interesse an unserem Produkt. In
der Anlage zu diesem Schreiben finden Sie unseren
Prospekt sowie die Preisliste 4/87.
```

Wenn Sie diesen Baustein von HIT aus aufrufen, dann erscheint zunächst die Meldung "Bitte Anrede eingeben:" in der Kommandozeile. Sie können dann z.B. schreiben: "Sehr geehrte Frau Müller!". CLOU fügt dann diese Anrede, gefolgt von dem restlichen Bausteintext, in Ihr Zieldokument ein. Die beiden Absatzmarken unter der `#F`-Anweisung sind nötig, weil CLOU einfache Leerzeilen überliest.

Sie können in der Kommandozeile neben Text aber auch weitere **Anweisungen** eingeben. CLOU befolgt diese, nachdem Sie Ihre Eingabe mit ↵ abgeschlossen haben. Sie können diese Eigenschaft der **Freien Texteinfügung** z.B. benutzen, um die Wirkung einfacher CLOU-Anweisungen auszuprobieren. Statt für jeden Versuch einen eigenen CLOU-Baustein zu erstellen, legen Sie einmal einen Baustein an, der nur eine `#F`-Anweisung enthält. Wenn Sie diesen Baustein nun von HIT aus aufrufen, dann können Sie in der Kommandozeile eine beliebige CLOU-Anweisung eingeben und ihre Wirkung sofort am Bildschirm sehen.

Ein weiterer Anwendungsfall wäre, daß Ihnen beim Einfügen des obigen Beispiel-Bausteins der Platz in der Kommandozeile für die gewünschte Anrede nicht aus reicht. Dann geben Sie einfach eine weitere `#F`-Anweisung mit ein.

Beispiel:

```
Sehr geehrter Herr Professor #F "weiter:"
```

CLOU reagiert darauf, indem er zunächst den Text `Sehr geehrter Herr Professor` in das Zieldokument einfügt. Dann befolgt er die `#F`-Anweisung: er fordert Sie also mit der Meldung `weiter:` erneut zu einer Eingabe auf. Jetzt können Sie z.B. "Maier-Schwarzenfeld" eingeben, und CLOU fügt diesen Namen unmittelbar hinter dem Wort `Professor` in das Dokument ein.

Variable Texteinfügung

Sollen längere Textpassagen von Hand eingefügt werden, so würde die **Freie Texteinfügung** zu umständlich. In solchen Fällen sollten Sie die **Variable Texteinfügung** verwenden:

Syntax

#V ["*Meldetext*"]

Wieder fordert CLOU Sie mit dem *Meldetext* bzw. einer Standardmeldung in der Kommandozeile zur Eingabe auf. Diese Eingabe erfolgt jetzt aber nicht in der Kommandozeile, sondern direkt im Zieldokument, d.h. CLOU übergibt die Kontrolle über HIT für die Dauer der **Variablen Texteinfügung** an Sie.

Anders als bei der **Freien Texteinfügung** können Sie also z.B. die Schreibmarke auch an andere Stellen bewegen oder Funktionen wie **Absatz formatieren**, **Attribut setzen** usw. ausführen. Ihr Einfügetext wird nicht mehr von CLOU interpretiert; eventuelle CLOU-Anweisungen also nicht ausgeführt. Mit anderen Worten: während der **Variablen Texteinfügung** arbeiten Sie mit HIT wie gewohnt, als wäre CLOU gar nicht vorhanden.

Mit (ENDE) geben Sie die Kontrolle an CLOU zurück; dieser fährt dann mit dem Baustein-Einfügen fort.

Parameter und der Trenn-Kommentar

Wie schon erwähnt, benötigen viele CLOU-Anweisungen **Parameter**. Damit CLOU erkennen kann, wo ein Parameter anfängt und wo er wieder aufhört, müssen Parameter in Anführungszeichen gesetzt werden. Diese Anführungszeichen sind aber nicht Bestandteil des Parameters selbst: die Anweisung

```
#F "Name:"
```

bewirkt, daß in der Kommandozeile die Aufforderung

```
Name:
```

ohne die Anführungszeichen erscheint. Wenn Sie im Meldungstext selbst Anführungszeichen stehen haben wollen, so müssen Sie diese in der Anweisung durch das Zeichen '\' (das "Fluchtsymbol") kenntlich machen. Alternativ dazu können Sie, wie von früheren CLOU-Versionen her gewohnt, die Anführungszeichen auch doppelt setzen:

Beispiel:

Anweisung	Meldetext
`#F "Name:"`	`Name:`
`#F "Der ""Name"":"`	`Der "Name":`
`#F "Der \"Name\":"`	`Der "Name":`
`#F """Name:\""`	`"Name:"`
`#F """Name:"""`	`"Name:"`

Außer den hier besprochenen Text-Parametern können auch Zahlen und sog. Variablennamen als Parameter auftreten. Auf diese wird jeweils an entsprechender Stelle eingegangen.

Häufig sind Parameter optional, d.h. sie können auch weggelassen werden. Wenn beim Fortlassen eines Parameters der nachfolgende Text zufällig mit einem Anführungszeichen beginnt, faßt CLOU ihn irrtümlich als Parameter auf. Das geschieht selbst dann, wenn Sie zwischen der Anweisung und dem Einfügetext Leerzeilen stehen haben: CLOU überliest sie bekanntermaßen.

Um CLOU in einer solchen Situation mitzuteilen, wo die Anweisung zu Ende ist, setzen Sie einen **Trenn-Kommentar**. Das ist ein "leerer" Kommentar, der nicht der Kommentierung dient, sondern nur als Trennmarke dient:

Beispiel:

```
#
#F #*#
"Dies hier ist normaler Einfügetext."

#* Der Trenn-Kommentar verhindert, daß obige Zeile als
   Parameter aufgefaßt und als Meldetext angezeigt wird.
#
```

1.2.4 Automatische Absatzformatierung

Fließtext, den CLOU aus einem Baustein in Ihr Dokument einfügt, verhält sich genauso, als würden Sie ihn von der Tastatur aus eingeben: - er erscheint in Ihrem Zieldokument zunächst im Flattersatz.

Wenn Sie aber einen Absatz in einem CLOU-Baustein mit einer **Absatzmarke** versehen, so löst CLOU unmittelbar nach dem Einfügen dieses Absatzes ein Nachformatieren aus. Der Effekt ist derselbe, als würden Sie die Taste F10 **Absatz formatieren** von Hand drücken.

Der Absatz wird dabei entsprechend dem gültigen Absatzattribut formatiert Dies geschieht nicht, wenn Sie die Absatzmarke fortlassen und den Absatz lediglich mit einer Leerzeile abschließen.

Diese automatische Absatzformatierung ist aber nicht immer erwünscht - der Kapitel 1.2.6 bringt hierfür ein Gegenbeispiel. Für solche Fälle können Sie die **automatische Absatzformatierung** nach Belieben ein- und ausschalten:

Syntax

#K ["*Modus*"]

Für *Modus* können Sie `ein` zum Einschalten bzw. `aus` zum Ausschalten der Automatik setzen. Zu Beginn des Baustein-Einfügens ist die Automatik eingeschaltet. Fehlt der Modus-Parameter, so wird in den jeweils anderen Modus umgeschaltet:

Beispiel:

```
#

#* Beginn des Baustein-Einfügens: Automatik ist
   eingeschaltet.

#

#K "aus"
#* Automatik ist ausgeschaltet
#

#K
#* Automatik ist eingeschaltet
#

#K
#* Automatik ist ausgeschaltet
#
```

Hinweis:

- `#K "aus"` muß immer gesetzt werden, wenn Format-Anweisungen verwendet werden (z.B. + beim Datum).
- Wenn Sie nicht explizit ein Absatzattribut (über #^) gesetzt haben, gilt das dem letzten CLOU-Zeilenlineal zugeordnete Absatzattribut.

1.2.5 Block einfügen

Die im letzten Abschnitt beschriebene Funktion `#K "aus"` sorgt dafür, daß Texte mit Formatanweisungen nicht nachträglich durch Absatzformatierung in Ihrem Format verändert werden.

Was Sie mit `#K` nicht ausschalten können, ist die CLOU-spezifische Art, Texte einzulesen. CLOU überliest generell Leerzeichen und -zeilen, damit Sie einen CLOU-Baustein übersichtlich schreiben können, die Struktur des Bausteins hinterher aber nicht im eingefügten Text erscheint. Wollen Sie ausnahmsweise doch einmal einen Teil des Textes genauso übernehmen, wie er im CLOU-Baustein steht, so müssen Sie dies CLOU über die Funktion **Block einfügen** mitteilen.

Der als Block gekennzeichnete Textteil wird dann zeilenweise unverändert in den Einfügetext übernommen. Das bedeutet im einzelnen:

- Jedes Zeichen erscheint genau an derselben Position im Einfügetext wie im CLOU-Baustein.
- Leerzeichen und Leerzeilen werden grundsätzlich mit in den Text übernommen.
- Zeilenlineale und Steuerzeilen werden unverändert übernommen.
- CLOU-Anweisungen werden nicht interpretiert, sondern als normaler Einfügetext übernommen.

Hinweis:

Text sollte auch dann als Block eingefügt werden,

- wenn im Randbereich Einfügetext steht, da dieser sonst nicht übernommen wird.
- wenn der Einfügetext keine Anweisungen enthält und bereits im richtigen Format vorliegt, da das Einlesen als Block weniger Zeit in Anspruch nimmt (CLOU muß nicht nach Anweisungen suchen).

CLOU übernimmt einen Text formatgetreu, wenn Sie diesen wie folgt klammern:

Syntax

```
#[
Text
#
```

Für *Text* können Sie beliebig viele Zeilen Text einsetzen, die auch Zeilenlineale und Steuerzeilen enthalten dürfen. Das abschließende # muß alleine in einer Zeile stehen, sonst wird es als normaler Einfügetext interpretiert. Fehlt es, wird bis zum Ende des aktuellen Bausteins gelesen.

Beispiel:

```
#

#* Die nachfolgende Aufzählung soll formatgetreu
  übernommen werden.
#

#[
    ▸ >---------------------------------------<
    1. Leerzeilen bleiben erhalten
    2. Leerzeichen bleiben erhalten
    3. Anweisungen werden als Text eingefügt
>-------------------------------------------<
#
```

Hinweis:

Zeilenlineale, die in einem als Block gekennzeichneten Textteil enthalten sind, werden nur dann unverändert übernommen, wenn es sich um einen formatierten Baustein handelt. In formatfreien Bausteinen werden Zeilenlineale nicht mitgesichert.

1.2.6 Ausgabe von Datum und Uhrzeit

Wenn Sie das aktuelle Datum und/oder die Uhrzeit in Ihr Dokument einfügen wollen, müssen Sie dazu nicht auf die Uhr sehen - CLOU erledigt das für Sie, wenn Sie ihm die Anweisung dazu erteilen:

Syntax

#T ["*Format*"]

Wenn CLOU diese Anweisung in einem Baustein findet, so fügt er an der aktuellen Schreibmarken-Position Datum und evtl. die Uhrzeit ein. In welcher Form er das tut, können Sie mit dem *Format* bestimmen. Dieses besteht aus einer oder mehreren Format-Angaben, das sind Buchstaben und Buchstaben-Kombinationen, die von CLOU durch das Tagesdatum und die aktuelle Uhrzeit zur Zeit des Einfügens ersetzt werden. So ersetzt er z.B. den Buchstaben `M` (für Monat) durch eine der Zahlen `1` bis `12`.

Folgende Formatangaben sind möglich:

T	Kalendertag (1, 2, ..., 31)
0T	Kalendertag zweistellig mit führender Null (01, 02, ..., 31)
TT	abgekürzter Name des Wochentages (Mo, Di, ..., So)
TTT	Name des Wochentages (Montag, Dienstag, ..., Sonntag)
M	Kalendermonat (1, 2, ..., 12)
0M	Kalendermonat zweistellig mit führender Null (01, 02, ..., 12)
MM	abgekürzter Name des Kalendermonats (Jan, Feb, ..., Dez)
MMM	Name des Kalendermonats (Januar, Februar, ..., Dezember)
J	Jahr zweistellig (87)
JJ	Jahr vierstellig (1987)
h	Stunde (0, 1, ..., 23)
0h	Stunde zweistellig mit führender Null (00, 01, ..., 23)
m	Minute (0, 1, ..., 59)
0m	Minute zweistellig mit führender Null (00, 01, ..., 59)
s	Sekunde (0, 1, ..., 59)
0s	Sekunde zweistellig mit führender Null (00, 01, ..., 59)
W	Tag im Jahr (1, 2, ..., 366).
0W	Tag im Jahr dreistellig mit führenden Nullen (001, 002, ..., 366)

Außer den eigentlichen Formatangaben darf das *Format* auch beliebigen Text enthalten. Sollen in diesem Text auch Buchstaben mit Sonderbedeutung enthalten sein, so müssen Sie sie mit zwei vorangestellten \\-Zeichen kenntlich machen (Fluchtsymbol), da CLOU sie sonst als Formatangaben interpretiert. Wie in früheren Versionen können Sie als Fluchtsymbol auch noch das !-Zeichen verwenden:

Beispiel:

```
#
#* Der in den Kommentaren angegebene Einfügetext
   bezieht sich auf den 1. April 1987, 13 Uhr 43.
   Wenn Sie die Anweisungen ausprobieren, erhalten
   Sie natürlich das aktuelle Tagesdatum und die
   genaue Uhrzeit.

#T "T. M. 'J"
#* Einfügetext: 1. 4. '87
#

#T "0T.0M.JJ"
#* Einfügetext: 01.04.1987
#

#T "0h:0m:0s"
#* Einfügetext: 13:43:07

#T "München, den T. MMM JJ"
#* Falsch! Diese Anweisung würde folgenden Einfügetext
   ergeben: 4ünc13en, den 1. April 1987
#

#T "\\Münc\\hen, den T. MMM JJ"
#* Einfügetext: München, den 1. April 1987
#

#T
#* Einfügetext: 1. April 1987
#
```

Wie das letzte Beispiel zeigt, ist der Standardwert für Format `"T. MMM JJ"`.

Wenn Sie eine Formatangabe attributieren, z.B. unterstreichen, so fügt CLOU den dazugehörigen Wert mit diesem Attribut in Ihr Dokument ein. Besteht eine Formatangabe aus mehreren Zeichen, so gibt das erste Zeichen den Ausschlag:

```
#T "\\Münc\\hen, den T. MMM JJ"
#* Einfügetext: München, den 1. April 1987
#
```

Ist das erste Zeichen im *Format* ein +, so fügt CLOU das Datum rechtsbündig am Ende der aktuellen Zeile ein. Dazu muß die automatische Absatzformatierung ausgeschaltet sein, da sonst die nachfolgende Formatierung die vor dem Datum eingefügten Leerzeichen wieder entfernen würde:

```
#

#K "aus"
#T "+T.M.JJ"
#* Einfügetext:                                    1.4.1987
#
```

1.3 Einfache Anweisungen (Teil II)

1.3.1 Aktivieren einer HIT-Funktion

Neben den Anweisungen, die CLOU für Sie ausführen kann, gibt es auch eine Reihe von Kommandos, die HIT selbst versteht: diese lösen Sie normalerweise aus, indem Sie Pfeil- oder Funktionstasten drücken. Wenn CLOU das für Sie tun soll, so sagen Sie ihm das mit der Funktionstasten-Anweisung:

Syntax

#^ [Anzahl] "Tastenname"

Für *Tastenname* können Sie einen der im Kapitel 5.2 vollständig aufgelisteten Namen einsetzen. Zu jeder Funktions- oder Pfeiltaste, die Sie von der Tastatur her kennen, finden Sie dort eine Bezeichnung. So steht dort z.B. für das Bewegen der Schreibmarke nach oben der Tastenname `C_OBEN`.

Die Anweisung wirkt genauso, als würden Sie selbst die entsprechende Taste drücken. Zur Abkürzung können Sie vor dem Tastennamen noch angeben, wie oft die Funktion ausgelöst werden soll: statt elfmal hintereinander die Anweisung

```
#^ "C_OBEN"
```

in Ihren Baustein zu setzen, können Sie einmal schreiben:

```
#^ 11"C_OBEN"
```

Sie können für *Anzahl* außer einer Zahl auch den Namen einer Rechenvariablen angeben (siehe Kapitel 2.1).

> *Hinweis:*
>
> Zahlen-Parameter wie *Anzahl* werden im Gegensatz zu Text-Parametern nicht in Anführungszeichen gesetzt!

Beispiel:

```
#
#^ "ZNT"
Rechnung
#^ "LBD"
#^ 3 "RETURN"

Sehr geehrte Damen und Herren,

   ...
```

Wenn Sie obigen Baustein einfügen, erhalten Sie folgendes Bild:

```
      >-------------------------------------------------<
                            Rechnung


      Sehr geehrte Damen und Herren,

      ...
```

Für die häufig benötigte Verstärkertaste gibt es eine Kurzform: Sie können den Namen VERSTÄRKER einfach fortlassen.

Beispiel:

```
#

  #^#^"A_OBEN"

  #* Die Schreibmarke steht jetzt am Anfang des Doku-
     ments. Die gleiche Wirkung hätten die Anweisungen:

  #

  #^ "VERSTÄRKER" #^ "A_OBEN"
```

Funktionstasten mit Parametern

Bei insgesamt fünf Funktionstasten ist zusätzlich zum Tastennamen die Angabe eines weiteren Parameters erforderlich. Dabei ist darauf zu achten, daß zwischen den beiden Parametern mindestens ein Leerzeichen steht! Zahlenangaben sind bei diesen Funktionstasten sinnlos.

Syntax

#^"SUCHEN" "*Suchwort*"

Die Schreibmarke wird auf das nächste Auftreten von *Suchwort* positioniert. Mit #^"WDH_SUCHEN" kann weitergesucht werden.

Syntax

#^"DOK_FORMATIEREN" "*Formatiertabelle*"

Das aktuelle Dokument wird formatiert. Für *Formatiertabelle* muß der Name einer Formatiertabelle stehen. Um anschließend in das unformatierte Do-kument zurückzukommen, können Sie die Anweisung:

#^"DOK_FORMATIEREN" ""

verwenden.

Syntax

#^"DOK_WECHSELN" "*Dokumentname*"

Auf diese Anweisung hin wechselt CLOU in das angegebene Dokument *Dokumentname* über. Dabei ist Vorsicht geboten: das aktuelle Dokument wird weder gesichert noch erfolgt irgendeine Warnung oder Rückfrage. Deshalb sollte unmittelbar vor dieser Anweisung explizit gesichert werden:

Syntax

#^"SICHERN" "*Dokumentname*"

Das aktuelle Dokument wird unter dem Namen *Dokumentname* gesichert. Auch hier ist zu beachten: existiert bereits ein anderes Dokument gleichen Namens, so wird dieses ohne Rückfrage überschrieben.

Syntax

#^"DRUCKEN" "*Druckername*"

Das aktuelle Dokument wird auf dem Drucker mit der Bezeichnung *Druckername* ausgedruckt. Die Namen der auf Ihrem System verfügbaren Drucker bekommen Sie z.B. angezeigt, wenn Sie in HIT-MENÜ die Taste [DRUCKEN] betätigen.

Hinweis:

- Für die Funktionstaste [F8] **Suchen und Ersetzen** ist nur die Teilfunktion **Suchen** als Tastenname realisiert. **Suchen und Ersetzen** muß über die CLOU-Standard-Funktion `suchen( )` angestoßen werden.
- Bei den Tasten, die HIT-Funktionsmenüs aufrufen, beachten Sie bitte: Die Tastenfolge, die benötigt wird, um diese Funktionen zu bedienen, muß exakt im Cloubaustein programmiert werden.
- Die Bearbeitung von HIT-Funktionsmenüs läßt sich komfortabler über CLOU-Funktionen oder -Makros (s. Kap. 3.2) realisieren.
- Ein Tabulator-Sprung [→|] muß im CLOU-Baustein nicht als Tastenname, sondern als normales Textzeichen eingegeben werden.

1.3.2 Ausgabe von Meldungen

In Ihre CLOU-Bausteine können Sie auch Meldungen einbauen, die während des Einfügens auf dem Bildschirm erscheinen und dem Benutzer wichtige Hinweise geben:

Syntax

#M "*Meldetext*"

Für *Meldetext* können Sie einen beliebigen, maximal 80 Zeichen langen Text angeben. Die Anweisung bewirkt, daß dieser *Meldetext* für ca. drei Sekunden in der Kommandozeile erscheint; der Benutzer wird durch einen Piepston darauf aufmerksam gemacht. Danach verschwindet die Meldung wieder. Auf das Zieldokument hat diese Anweisung keinen Einfluß.

Das Ende eines CLOU-Bausteins könnte z.B. so aussehen:

Beispiel:

```
...
#M "Bitte das fertige Dokument in Ablage 3b!"
```

1.3.3 Auswahl von Alternativen

Häufig soll an einer bestimmten Stelle im Text eine von mehreren möglichen Formulierungen stehen. Wenn Sie z.B. einen CLOU-Baustein `Rechnung` erstellen wollen, so stellt sich das Problem der Anredeform: einmal ist es eine `Sehr geehrte Dame`, die die Rechnung erhalten soll, ein anderes Mal ist es ein `Sehr geehrter` oder auch `Lieber Herr`. Damit Sie in solchen Situationen die Anrede nicht jedesmal "von Hand" - z.B. durch Freie Texteinfügung - eingeben müssen, können Sie die verschiedenen Anreden in einer **Alternativen**-Anweisung im Baustein verankern:

Syntax

```
#A
        Alternative1
/
        Alternative2
        ...
/
        Alternativen
#
```

Für *Alternative1* bis *AlternativeN* kann beliebiger Text stehen, der sich auch über mehrere Zeilen erstrecken darf. Dieser Text darf beliebig viele CLOU-Anweisungen enthalten - einschließlich weiterer `#A`-Anweisungen. Es dürfen maximal 20 Alternativen angegeben werden.

Die Wirkung ist folgende: CLOU zeigt die Texte, die jeweils zwischen den /-Zeichen stehen, in Form eines Menüs an und gibt die Meldung "`Alternative auswählen:`" aus. Sie können nun mit den Pfeiltasten eine der Alternativen ansteuern und mit [↵] auswählen. Der ausgewählte Text wird dann in das Zieldokument übernommen; die anderen ignoriert.

Beispiel:

```
#

#* Rechnung mit alternativen Anredeformen.
#
#^ "ZNT"
RECHNUNG
#^ "LBD"
#^3 "RETURN"

#A
    Sehr geehrte Dame!
/
    Sehr geehrter Herr!
/
    Sehr geehrte Damen und Herren!
#

    ...
```

Wenn CLOU beim Einfügen des Bausteins die #A-Anweisung erreicht, zeigt er Ihnen die drei möglichen Anredeformen als Menü auf dem Bildschirm an. Wenn Sie nun z.B. den Text `Sehr geehrter Herr!` ansteuern und mit ↵ auswählen, so fügt CLOU eben diesen Text in das Zieldokument ein.

Die einzelnen Alternativen können neben reinem Einfügetext aber auch CLOU-Anweisungen enthalten. Das folgende Beispiel zeigt, wie nützlich das ist:

Beispiel:

```
#

#* Baustein für die Erstellung von Geschäftsbriefen
#

#B "briefkopf"
#A
    #B "angebot"
/
    #B "rechnung"
/
    #B "mahnung"
/
    #V "Bitte Brieftext eingeben:"
#

#B "briefende"
```

Wenn Sie diesen Baustein von HIT aus aufrufen, so geschieht der Reihe nach folgendes:

- CLOU befolgt die erste Anweisung, indem er den Textbaustein `briefkopf` in das Zieldokument einfügt. Dieser kann z.B. die Adresse Ihrer Firma, die CLOU-Datumsanweisung usw. enthalten.
- Dann zeigt CLOU Ihnen ein Menü, in dem Sie die drei `#B`- sowie die `#V`-Anweisung sehen. Der weitere Verlauf hängt von Ihrer Entscheidung ab:
  - Wenn Sie eine der `#B`-Anweisungen auswählen, so befolgt CLOU diese, indem er den entsprechenden Baustein einfügt.
  - Wählen Sie hingegen die `#V`-Anweisung, so gibt CLOU die Meldung `Bitte Brieftext eingeben:` in der Kommandozeile aus und Sie können beliebigen Text direkt in das Dokument einfügen.
- Schließlich fügt CLOU den Baustein `briefende` ein, der z.B. den Text `mit freundlichen Grüßen` sowie den Namen des Unterzeichnenden enthält.

CLOU befolgt also nur die Anweisungen, die Sie aus dem Menü ausgewählt haben.

1.3.4 Positionieren der Schreibmarke

Mit Hilfe der folgenden Anweisungen können Sie die Schreibmarke in eine beliebige Spalte innerhalb der aktuellen Zeile oder sogar an eine beliebige Stelle innerhalb Ihres Dokuments bewegen:

Syntax

#G Spalte
#J Spalte Zeile

Für *Spalte* bzw. *Zeile* müssen Zahlen stehen. (Es können stattdessen auch die Namen von Rechenvariablen eingesetzt werden, diese sind Gegenstand des Kapitels 2.1). Die Anweisung bewirkt, daß CLOU die Schreibmarke in die angegebene Spalte und Zeile Ihres Dokuments bewegt.

Die Spalten werden vom linken Rand des Zeilenlineals aus gerechnet, Zeilen von der ersten Zeile des Dokuments an. Negative Werte für *Spalte* zählen vom rechten Rand des Zeilenlineals rückwärts, -1 steht also für den rechten Rand des Textbereichs selbst. Entsprechend bedeutet -1 als Angabe für *Zeile* die letzte Zeile des aktuellen Dokuments, -2 die vorletzte usf. Die Zahl 0 bezieht sich auf die aktuelle Spalte bzw. Zeile.

Bitte beachten Sie:

- Mit `#G -1` positionieren Sie unmittelbar unter die rechte Randspalte. Eingaben dort werden sofort in die nächste Zeile übernommen. Um also am rechten Rand ein einzelnes Zeichen zu schreiben, müssen Sie auf die Spalte davor (`#G -2`) positionieren.
- Das Dokument muß groß genug sein, um die angegebene Position zu enthalten. Ein Dokument mit nur drei Zeilen Länge wird nicht automatisch verlängert, wenn Sie versuchen, mit der `#J`-Anweisung in Zeile 9 zu positionieren! Die Anweisung wird in einem solchem Fall ignoriert.

Beispiel:

```
#

#* Verlängern des Dokuments auf 10 Zeilen Länge:
#
#^10 "RETURN"

#* Nun werden an verschiedenen Stellen im Dokument
   einzelne Buchstaben eingefügt:
#

#G 8       A      #* Spalte 8 #
#G 12      B      #* Spalte 12, gleiche Zeile #
#J -2 -4   C      #* letzte Zeile, rechter Rand #
#J 0 -2    D      #* gleiche Spalte, vorletzte Zeile #
```

Sie können die #G-Anweisung z.B. zum Erstellen einer Tabelle verwenden. Dazu bewegen Sie die Schreibmarke nacheinander auf die gewünschten Zeilen- und Spaltenpositionen und fügen dort den gewünschten Text ein. Dies ist insbesondere im Zusammenhang mit "Schleifen-Anweisungen" sinnvoll, die in Kapitel 3 (Komplexe Anweisungen) beschrieben sind.

Hinweis:

Beim Erstellen von Tabellen muß unbedingt die automatische Absatzformatierung abgeschaltet werden (#K-Anweisung), da sonst die nachfolgende Formatierung das Tabellenformat wieder zerstören würde.

Bearbeiten des Zeilenlineals

Sie können die #G-Anweisung auch zum Positionieren im Zeilenlineal verwenden. Die Spaltenangabe erfolgt absolut, d.h. mit dem Wert 1 für *Spalte* ist der linke Bildschirmrand gemeint; der max. zulässige Wert ist 250. Negative Angaben zählen von rechts: -1 entspricht 250, -2 entspricht 249 usw.

Die verschiedenen Marken für linken und rechten Rand, Tabulatoren etc. setzen Sie über Tastennamen (s. Übersicht Kap. 5.3).

Beispiel:

```
#

#* Dieser Baustein verändert das aktuelle Zeilenlineal
   in Ihrem Dokument. Der linke Rand wird auf Spalte
   10, der rechte auf Spalte 70 gesetzt.
#

#^"ZL_BEARBEITEN""
#G 10 #^ "L_RAND"
#G 70 #^ "R_RAND"
#^"RETURN"
```

Hinweis für HIT3-Benutzer:

CLOU-Bausteine der HIT-Versionen 3.0 oder 3.1, in denen Marken und Tabulatoren über Ziffern gesetzt wurden, sind weiterhin ablauffähig.

1.3.5 Position der Schreibmarke abfragen

Dieser Abschnitt setzt Kenntnisse über Rechenvariablen (Kapitel 2.1) voraus. Sie können ihn bei der ersten Lektüre übergehen.

Diese Anweisung dient dazu, die aktuelle Position der Schreibmarke in Variablen zu speichern. Zu einem späteren Zeitpunkt kann dann z.B. mit der #J-Anweisung an die gleiche Stelle zurückgekehrt werden.

Syntax

#Z *Spalte Zeile*

Für *Spalte* bzw. *Zeile* muß jeweils der Name einer Rechenvariablen eingesetzt werden. Die Anweisung bewirkt, daß der Variablen *Spalte* der aktuelle Wert des Spaltenzählers - vom linken Rand des Zeilenlineals aus gerechnet - zugewiesen wird. Ebenso bekommt die Variable *Zeile* den aktuellen Wert des Zeilenzählers.

Beispiel:

```
#

#D x 0    #* Variablen-Definition für Spaltenposition #
#D y 0    #* Variablen-Definition für Zeilenposition #

#* Merken der aktuellen Position:
#
#Z x y

#* Anfügen eines Bausteins ans Ende des Dokuments:
#
#^#^"A_UNTEN"
#^"RETURN"
#B "briefende"

#* Zurück zur gemerkten Position:
#
#J x y
```

1.4 Anweisungen für Test- und Demonstrationszwecke

Dieses Kapitel beschreibt einige Anweisungen, die auf das Aussehen des Zieldokuments selbst keinen Einfluß haben. Sie beeinflussen aber den Einfügevorgang. So können Sie z.B. CLOU "bei der Arbeit zuschauen".

Die hier zusammengefaßten Anweisungen sind daher besonders dann nützlich, wenn Sie einen neu erstellten Baustein testen oder bereits festgestellte Fehler in ihm lokalisieren wollen. Es kann aber durchaus auch bei einwandfrei arbeitenden Bausteinen interessant sein, den Einfügevorgang einmal genauer unter die Lupe zu nehmen: viele Eigenschaften von CLOU werden dadurch verständlicher.

1.4.1 Bildschirmausgabe veranlassen

Normalerweise verläuft das Einfügen eines Textbausteins "unsichtbar", d.h. auf dem Bildschirm ist solange nichts davon zu sehen, bis der ganze Baustein eingefügt ist. Würde stattdessen jedes Zeichen, das CLOU in Ihr Dokument einfügt, auch sofort am Bildschirm erscheinen, so würde das Einfügen wesentlich länger dauern.

In manchen Situationen kann es aber durchaus nützlich sein, CLOU bei der Arbeit "über die Schulter zu gucken". Das ist z.B. der Fall, wenn der Baustein einen Fehler enthält und man genau sehen will, wann dieser sich bemerkbar macht: es ist dann oft leichter, die fehlerhafte Stelle ausfindig zu machen. Ganz allgemein wird die Arbeitsweise von CLOU bei komplexeren Bausteinen leichter verständlich, wenn man den Einfügevorgang direkt am Bildschirm mitverfolgen kann.

Dafür gibt es die folgende Anweisung:

Syntax

#+

Wenn CLOU auf diese Anweisung stößt, so wartet er mit der weiteren Bausteinbearbeitung, bis HIT alle bisherigen Einfügungen "verdaut" und das Resultat am Bildschirm angezeigt hat. Wenn Sie diese Anweisung an verschiedenen Stellen in einen Baustein einstreuen, so können Sie später den Einfügevorgang am Bildschirm mitverfolgen. Eine Variante dieser Anweisung löst ebenfalls eine Bildschirmausgabe aus, ohne daß CLOU aber auf deren Abschluß wartet. Das beschleunigt den Einfügevorgang, was für ein optisches Mitverfolgen nicht immer günstig sein muß. Sie können im Einzelfall beide Anweisungen ausprobieren:

Syntax

```
#-
```

Hinweis:

Sollte Ihnen das Einfügen auch mit der #+-Anweisung noch zu schnell gehen, so können Sie die im nächsten Abschnitt beschriebene Halt-Anweisung verwenden.

Beispiel:

```
#^ "SICHERN" "hier"
#+
#! "mv hier /usr/gast/dort"
#* Der Sicherungsvorgang muß abgeschlossen sein,
   bevor das Dokument übertragen werden kann.
#
```

1.4.2 Einfügevorgang anhalten

Zu Test- und Demonstrationszwecken kann es mitunter günstig sein, wenn man verschiedene Stadien des Einfügevorgangs am Bildschirm mitverfolgen kann. Damit dabei genug Zeit bleibt, die Resultate zu überprüfen, können Sie die **Halt-Anweisung** verwenden:

Syntax

#% Anzahl

Diese Anweisung bewirkt, daß der aktuelle Einfügetext am Bildschirm angezeigt wird. Danach wartet CLOU *Anzahl* Sekunden, bevor er mit dem Einfügen fortfährt. Für *Anzahl* kann außer einer Zahl auch eine Rechenvariable eingesetzt werden (siehe Kapitel 2.1).

Achtung:

Die #%-Anweisung darf auf keinen Fall innerhalb eines abgeschlossenen Vorgangs (z.B. **Markieren und Bearbeiten**) verwendet werden, da sie in einem solchen Fall von CLOU bzw. HIT nicht verarbeitet werden kann!

1.4.3 Bearbeitung abbrechen

Wenn Sie größere Bausteine erstellen, dann können sich manchmal Fehler einschleichen, die nicht mehr leicht zu finden sind: Das Dokument sieht nach dem Einfügevorgang ganz anders aus, als Sie es sich vorgestellt haben. Wo liegt aber der Fehler in Ihrem Baustein? Um die Sucharbeit zu erleichtern, können Sie z.B. vor eine Anweisung, die Ihnen zweifelhaft erscheint, die **Abbruch-Anweisung** setzen:

Syntax

```
#;
```

CLOU bricht an dieser Stelle den Einfügevorgang ab. An dem eingefügten Text können Sie dann erkennen, ob der Baustein bis dahin in Ordnung war.

Hinweis:

Außer bei der Fehlersuche findet die **Abbruch-Anweisung** vor allem im Zusammenhang mit den in Kapitel 3 behandelten komplexen Anweisungen Verwendung. Dort finden Sie auch Beispiele.

2 Variablen

2.1 Rechenvariablen

Das Bausteinverarbeitungsprogramm CLOU kann für Sie auch zählen, rechnen und überhaupt gut mit Zahlen umgehen. Sie brauchen also in den meisten Fällen keinen Taschen- oder Tischrechner mehr auf Ihrem Schreibtisch, wenn Sie z.B. Rechnungen oder einen Lohnsteuer-Jahresausgleich erstellen.

Für den Umgang mit Zahlen benutzt CLOU **Rechenvariablen**, im folgenden kurz "Variablen" genannt (im Kapitel 2.2 wird eine zweite Variablen-Art, die "Stringvariable" eingeführt). Eine Variable ist für CLOU eine Art Notizzettel, auf dem er sich eine Zahl - den **Wert** der Variablen - sowie ein Wort - den **Namen** der Variablen - notieren kann. Für Sie heißt das, daß Sie Ihrer Hilfskraft CLOU sagen können: "Merke dir unter dem Namen `Stückpreis` die Zahl `2.99`!". Dafür gibt es eine einfache CLOU-Anweisung, die Variablen-Definition; diese wird im nächsten Abschnitt ausführlich beschrieben.

Von dem Moment an, zu dem CLOU diese Anweisung in einem Baustein gelesen hat, kann er jederzeit für den Namen `Stückpreis` die Zahl `2.99` einsetzen - und damit rechnen.

Der Name einer Variablen liegt - wenn Sie ihn einmal definiert haben - bis zum Ende des Einfügevorgangs fest. Anders ist das mit dem Wert der Variablen: Sie können CLOU an jeder beliebigen Stelle anweisen, einer bestimmten Variablen einen neuen Wert zu geben - also, bildlich gesprochen, den alten Wert auf dem entsprechenden Notizzettel auszuradieren und einen neuen einzutragen (das ist übrigens auch der Grund für die Bezeichnung "Variable"). Dieser neue Wert gilt dann solange - und kann solange immer wieder für irgendwelche Berechnungen herangezogen werden - bis er erneut von einer anderen Zahl "überschrieben" wird.

Schließlich können Sie CLOU mit einer Ausgabe-Anweisung auffordern, den Wert einer Variablen in Ihr Dokument einzufügen. Da es viele Möglichkeiten gibt, eine Zahl zu schreiben - mit oder ohne Nachkommastellen, mit führenden Nullen, unterstrichen oder fett etc. - hat jede CLOU-Variable auch ein **Ausgabeformat**. Dieses können Sie nach Belieben festlegen.

Insgesamt gibt es vier verschiedene Anweisungen, die Sie CLOU im Zusammenhang mit Variablen erteilen können:

- Die Variablen-Definition

  Bevor Sie eine Variable verwenden können, müssen Sie CLOU mit deren Namen bekannt machen. Dies geschieht in einer speziellen Definitions-Anweisung. Erst nach dieser akzeptiert CLOU weitere Anweisungen, die sich auf die Variable beziehen. Mit der Definition legen Sie außerdem einen Anfangswert sowie das Ausgabeformat für jede Variable fest.

- Die Eingabe-Anweisung

  Mit ihr ist es möglich, den Wert einer Variablen während des Einfügevorgangs vom Benutzer eingeben zu lassen. Die eingegebenen Werte können dann von CLOU weiterverarbeitet werden.

- Die Wertzuweisung

  Das Ergebnis einer Berechnung, die CLOU für Sie durchführt, können Sie mit dieser Anweisung einer Variablen als Wert zuweisen. Dieser Wert kann in einer nachfolgenden Ausgabe-Anweisung in Ihr Dokument eingefügt oder in anderen Berechnungen weiterverarbeitet werden.

- Die Ausgabe-Anweisung

  Damit veranlassen Sie CLOU dazu, den aktuellen Wert einer Variablen in Ihr Dokument einzufügen.

2.1.1 Definition von Variablen

Zu jeder Variablen gehören drei Angaben:

- ein Variablenname, mit dem man die Variable ansprechen kann;
- ein Wert, d.h. die Zahl, für die der Variablenname steht;
- und ein Ausgabeformat, das z.B. angibt, mit wie vielen Nachkommastellen die Variable in Ihr HIT-Dokument geschrieben werden soll.

Bevor Sie eine Variable verwenden können, müssen Sie diese Angaben machen. Dazu dient die Definitionsanweisung:

Syntax

#D VarName Wert ["*Format*"]

Ein *VarName* besteht aus bis zu 30 Buchstaben. Damit Ihre CLOU-Bausteine übersichtlich und leicht zu lesen sind, sollte er einen leicht erkennbaren Zusammenhang mit dem Verwendungszweck der Variablen aufweisen. Außer Buchstaben dürfen Namen auch die Ziffern '0' bis '9', den Unterstrich '_' sowie das geschützte Leerzeichen '␣' enthalten. Das erste Zeichen muß jedoch ein Buchstabe sein. Andere Zeichen wie Komma, Bindestrich etc. sind nicht zulässig.

Erlaubt sind also z.B. folgende Variablennamen:

```
Summe
MWSt
DM_pro_Liter
i
x1
```

Unzulässig sind hingegen:

```
Preis_in_$          ("$" ist kein Buchstabe und keine Zahl)
3fach               (Zahl als erstes Zeichen)
Donaudampfschiffahrtsgesellschaftskapitän
                    (länger als 30 Buchstaben)
```

Der *Wert* kann im einfachsten Fall eine Zahl, darf aber auch ein beliebig komplizierter Rechenausdruck sein, wie er im nächsten Kapitel beschrieben ist.

Zahlen können mit oder ohne Nachkommastellen geschrieben werden; zur Trennung der Vor- von den Nachkommastellen muß der Dezimalpunkt verwendet werden. Eine Null vor dem Dezimalpunkt kann weggelassen werden. Negative Zahlen werden durch ein vorangestelltes "-" dargestellt; zwischen dem Minuszeichen und der Zahl selbst darf kein Zwischenraum stehen.

Erlaubte Zahlen sind also:

```
0
1985
-333
3.14159
2.00
.5
-0.33
```

Nicht erlaubt sind:

`10e3`	(enthält einen Buchstaben)
`- 333`	(Leerzeichen zwischen "-" und "333")
`2,5`	(Komma statt Punkt)
`+3`	(nur "-" als Vorzeichen erlaubt)

Auf das *Format* wird im Abschnitt "Ausgabe von Variablenwerten" eingegangen. Nachdem die Formatangabe optional ist, also auch weg-gelassen werden darf, wird sie in den Beispielen dieses Kapitels noch nicht verwendet.

In den meisten Fällen wird der Wert einer Variablen erst nach deren Definition festgelegt - etwa weil er vom Benutzer eingegeben oder aus anderen Werten berechnet werden soll. Trotzdem muß in der Definition ein Wert angegeben sein. In solchen Fällen ist es üblich, die Zahl 0 als Wert einzusetzen.

Beispiel:

```
#D Gesamtpreis 0
#D MWSt 14
#D DM_pro_Liter 1.29
```

Mit diesen Definitionen kennt CLOU drei Variablen: eine mit dem Namen `Gesamtpreis` und dem Wert `0`, eine namens `MWSt` mit dem Wert `14` und die dritte mit dem Namen `DM_pro_Liter` und dem Wert `1.29`.

Gültigkeitsbereich von Variablennamen

Prinzipiell muß jede Variable definiert werden, bevor sie das erste Mal verwendet wird. Von dem Zeitpunkt an, zu dem CLOU die Definitions-Anweisung gelesen hat, "kennt" er den Variablennamen *bis zum Ende des* Baustein-Einfügens, d.h. auch dann noch, wenn mit einer `#B`-Anweisung ein weiterer Textbaustein eingelesen wird.

Dadurch ist es möglich, beim verschachtelten Einfügen von Bausteinen eine Variable in einem Baustein zu definieren und - jeweils mit ihrem Wert - in einem anderen zu verwenden. Achten Sie aber bei Verschachtelungen bitte darauf, nicht in zwei verschiedenen Bausteinen den gleichen Namen zu benutzen: die jeweils letzte Definition "überschreibt" alle vorangehenden mit demselben Namen, und der alte Wert der Variablen geht verloren.

Neben den von Ihnen definierten Variablen gibt es auch einige, die CLOU von sich aus kennt. Diese spielen im Zusammenhang mit bestimmten Anweisungen eine besondere Rolle und werden in dieser Beschreibung an entsprechender Stelle einzeln beschrieben.

2.1.2 Rechenausdrücke

Wie schon oben erwähnt, kann als Wert einer Variablen auch ein **Rechenausdruck** eingesetzt werden. Ein solcher Ausdruck kann bestehen aus:

- Zahlen,
- Variablennamen,
- Rechenzeichen,
- Klammern.

CLOU kennt fünf Rechenzeichen:

\+ für die Addition,
\- für die Subtraktion,
* für die Multiplikation,
/ für die Division.
% für die Restberechnung (Modulo-Operator)

Mit Hilfe des Modulo-Operators '%' kann der Rest einer Division von zwei ganzzahligen Werten berechnet werden. Für die Verwendung gelten dieselben Regeln wie für das Divisions-Zeichen '/'.

Zwischen den einzelnen Zahlen, Rechenzeichen, Namen und Klammern darf beliebig viel Abstand gelassen werden, um die Ausdrücke übersichtlich zu gliedern; ein Ausdruck darf sogar über mehrere Zeilen gehen.

CLOU berechnet einen solchen Ausdruck von links nach rechts unter Berücksichtigung der Regel "Punkt-vor-Strich". So haben folgende Ausdrücke die jeweils angegebenen Werte:

Ausdruck	Wert
`3 + 4`	`7`
`11 - 5 - 4`	`2 (nämlich 6-4)`
`2*2 + 3*3`	`13 (nämlich 4+9)`
`60/10/2`	`3 (nämlich 6/2)`
`10%7 + 1`	`4 (nämlich 3+1)`

Wenn Sie wollen, daß CLOU einen Ausdruck in anderer Reihenfolge berechnet, dann müssen Sie Klammern setzen. Alles, was innerhalb eines Klammerpaares steht, wird zuerst berechnet; anschließend werden die Teilergebnisse weiterverarbeitet:

Ausdruck	Wert
`11 - (5 - 4)`	`10 (nämlich 11-1)`
`2 * (2 + 3) * 3`	`30 (nämlich 2*5*3)`
`60/(10/2)`	`12 (nämlich 60/5)`
`10%(7 + 1)`	`2 (nämlich 10%8)`

Was können Sie nun mit einem solchen Rechenausdruck anfangen? Sie können ihn als Wert in eine Variablen-Definition einsetzen, und CLOU rechnet für Sie. Nehmen Sie z.B. an, Sie wollten eine Rechnung erstellen. Ein Ausschnitt aus dem entsprechenden CLOU-Baustein könnte dann so aussehen:

Beispiel:

```
#

#D Stückpreis 2.10
#D Stückzahl 20
#D Nettopreis Stückpreis * Stückzahl

#* CLOU multipliziert jetzt 2.10 mit 20 und notiert
   das Resultat (42.00) als Wert der Variablen mit
   dem Namen "Nettopreis".
#

#D MWSt 14
#D Endpreis Nettopreis + (Nettopreis * (MWSt/100))

#* Nun hat "Endpreis" den Wert 47.88
#
```

2.1.3 Ausgabe von Variablenwerten

Damit CLOU den Wert einer Variablen in Ihr Dokument einfügt, müssen Sie ihm eine **Ausgabe-Anweisung** erteilen:

Syntax

#> ["*Format*"] *VarName*

Mit dieser Anweisung sagen Sie: "Schreibe den aktuellen Wert der Variablen *VarName* an der Stelle, an der sich gerade die Schreibmarke befindet, in mein HIT-Dokument". Je nach dem angegebenen Format verwendet CLOU dabei führende Nullen, schreibt rechts- oder linksbündig usw. (siehe unten!). Fehlt die Formatangabe, so wird das Format aus der Variablen-Definition verwendet. Fehlt auch dieses, so wird das unten angegebene Standardformat verwendet.

Das *Format* hat folgenden Aufbau:

Syntax

[–][*Feldbreite*][.*Nachkommastellen*]

Für *Feldbreite* und *Nachkommastellen* müssen jeweils ganze Zahlen eingesetzt werden. Die Feldbreite gibt an, wieviele Zeichen inclusive Dezimalpunkt mindestens ausgegeben werden sollen. Falls die Zahl kürzer als die Feldbreite ist, werden entsprechend viele Leerzeichen vor der Zahl selbst eingefügt: dadurch erreicht man in Tabellen die gewohnte rechtsbündige Darstellung. Ist die Zahl länger als die Feldbreite, so wird sie trotzdem in voller Länge ausgegeben: Die Feldbreite gibt die Mindestzahl der auszugebenden Zeichen an, führt aber nie zum "Abschneiden" der Zahl.

Die Zahl der auszugebenden *Nachkommastellen* ist ebenfalls frei wählbar. Ist sie Null, so wird die Zahl - gegebenenfalls gerundet - ohne Nachkommastellen ausgegeben.

Ein '-' am Anfang des Formats bewirkt, daß die Zahl linksbündig ausgegeben wird, evtl. nötige Leerzeichen also hinter der Zahl eingefügt werden.

Wie beim Datumsformat kann durch Attributierung des ersten Zeichens der Formatangabe eine Attributierung des ausgegebenen Wertes erzielt werden.

Beispiele:

Der Wert der auszugebenden Variablen sei 12.05.

Format	Ausgabe
"8.2"	12.05
"8.2"	12.05
"-8.2"	12.05
".4"	12.0500
"2.2"	12.05
"6.0"	12
"6.1"	12.1
".0"	12
"6"	12.050000

Wird weder bei der Variablen-Definition noch in der Ausgabe-Anweisung ein Format angegeben, so wird das Standardformat ".0" verwendet, das nur ganze Zahlen liefert (siehe letztes Beispiel oben!)

Erweitertes Ausgabeformat

Für die Ausgabe von Variablen kann auch ein erweitertes Ausgabeformat verwendet werden. Ein solches kann - ähnlich wie das Datumsformat - aus beliebigem Text bestehen, wobei folgende **Formatzeichen** eine Sonderbedeutung haben:

0	(Null) Vorkommastelle. Stehen mehr Nullen im Formatstring, als die Zahl Vorkommastellen hat, so wird mit führenden Nullen aufgefüllt.
V	Vorkommastelle. Gegebenenfalls wird mit führenden Leerzeichen aufgefüllt.
X	Vorkommastelle. Alle Zeichen links von dem ersten 'X' im Formatstring, für das eine Vorkommastelle existiert, werden ignoriert.
N	Nachkommastelle.

Alle übrigen Zeichen werden unverändert ins Zieldokument übernommen. Soll eines der Formatzeichen selbst im Text erscheinen, so muß ihm seine Sonderbedeutung durch ein vorangestelltes \- oder !-Zeichen (Fluchtsymbol) genommen werden. Die verschiedenen Formatzeichen können auch vermischt verwendet werden. Dabei ist folgende Reihenfolge einzuhalten: `X` vor `V` vor `0` vor `N`.

Beispiele:

Der Wert der auszugebenden Variablen sei 1234.5678.

Format	Ausgabe
`"VVVVVV"`	`1234`
`"X.XXX.XXX,NN"`	`1.234,57`
`"00000,NN DM"`	`01234,57 DM`
`"VV"`	`35`

2.1.4 Manuelle Eingaben von Variablenwerten

Häufig soll der Wert einer Variablen nicht im CLOU-Baustein festgelegt, sondern erst beim Einfügen des Bausteins über die Tastatur eingegeben werden. Das geschieht so:

Syntax

#< VarName ["Meldetext"]

Bei dieser Anweisung hält CLOU mit der Bausteinverarbeitung an und fordert den Benutzer in der HIT-Kommandozeile dazu auf, eine Zahl einzugeben. Die Variable *VarName* erhält dann diese Zahl als Wert. Für *Meldetext* kann ein beliebiger Text angegeben werden; dieser wird dann als Aufforderung in der Kommandozeile angezeigt. Fehlt der *Meldetext*, so wird stattdessen eine vordefinierte Standardmeldung ausgegeben. Macht der Benutzer bei der Eingabe Fehler, indem er z.B. Buchstaben statt Zahlen eintippt, so wird seine Eingabe zurückgewiesen und die Aufforderung solange wiederholt, bis er eine erlaubte Zahl eingegeben hat.

Beispiel:

```
#D Preis 0
#< Preis "Bitte Stückpreis eingeben:"

#* die vorige Anweisung bewirkt, daß der Text
   "Bitte Stückpreis eingeben:"
   in der Kommandozeile erscheint. Der Benutzer kann
   jetzt z.B. "3.99" eingeben: dann bekommt die
   Variable "Preis" den Wert 3.99
#
```

Um die Eingabe unsinniger Werte durch den Benutzer zu verhindern, können Sie den erlaubten Zahlenbereich auch einschränken:

Syntax

#< "Bedingung" VarName ["Meldetext"]

In dieser Form bewirkt die Anweisung, daß der Benutzer solange zur Eingabe aufgefordert wird, bis die angegebene *Bedingung* erfüllt ist. Wie eine *Bedingung* aussieht, ist in Kapitel 3.1.2 beschrieben. Dort finden Sie auch ein Beispiel für die bedingte Eingabe-Anweisung. Bei der ersten Lektüre können Sie diese Form einfach übergehen.

Wurde der Variablen bereits ein Wert zugeordnet (bei der Definition oder einer früheren Eingabe), so kann dieser als Voreinstellung für die Eingabe herangezogen werden. Sie müssen dazu die Eingabe-Anweisung in folgender Form verwenden:

Syntax

#« VarName ["Meldetext"]

2.1.5 Wertzuweisung

Nachdem Sie eine Variable einmal definiert haben, können Sie ihren Wert jederzeit ändern:

Syntax

 # = VarName Ausdruck

Für *VarName* muß der Name einer vorher definierten Variablen stehen. *Ausdruck* kann ein beliebiger Rechenausdruck sein, im einfachsten Fall auch eine Zahl. Die Anweisung bewirkt, daß die Variable *VarName* den Wert von *Ausdruck* erhält.

Beispiel:

```
#D Netto 0
#D Brutto 0

#< Netto "Bitte Nettopreis eingeben:"
#= Brutto Netto*1.14

#* Angenommen, der Benutzer habe "10" eingegeben.
   Dann hat die Variable "Brutto" jetzt den Wert 11.40
#
```

Ausdruck darf auch den Namen der Variablen selbst enthalten. Damit kann man obiges Beispiel auch kürzer formulieren:

Beispiel:

```
#D Preis 0

#< Preis "Bitte Nettopreis eingeben:"
#= Preis Preis*1.14

#* Angenommen, der Benutzer habe "10" eingegeben.
   Dann hat die Variable "Preis" jetzt den Wert 11.40
#
```

2.1.6 Automatisches Runden in CLOU 4.1

Beim Rechnen mit CLOU-Variablen können derzeit Rundungsfehler auftreten. Die Ursache hierfür ist, daß CLOU bei Rechenvariablen intern immer mit der vollen Genauigkeit (entsprechend der Maschinengenauigkeit bei Gleitpunktzahlen) arbeitet. Nur bei der Ausgabe eines Variablenwertes wird der Wert entsprechend der Formatangabe gerundet, intern wird jedoch stets mit dem ungerundeten Wert weitergearbeitet.

Das Problem tritt i.d.R. relativ selten auf, nämlich nur dann, wenn mit sehr vielen Nachkommastellen gearbeitet wird (etwa beim Dividieren). Ein typischer Folgefehler davon ist, daß bei einer längeren Aufsummierung die Summe der gerundeten Posten ungleich der gerundeten Summe der Posten ist. Beispiel:

	genauer Wert	gerundeter Wert (2 Stellen)
	1.475	1.48
	1.475	1.48
Summe:	2.95	2.95 !

Dieser Fehler kann ab CLOU-Version 4.1 durch Setzen der Option **-j** behoben werden. Diese bewirkt, daß bei jeder Zuweisung eines Wertes an eine Rechenvariable dieser Wert entsprechend der Anzahl Nachkommastellen, die im Format der Variablen angegeben sind, gerundet wird. Damit stimmt der genaue Wert stets mit dem gerundeten überein und das obige Problem kann nicht mehr auftreten.

Das Setzen dieser Option kann erfolgen entweder aus der SHELL:

```
CLOUOPT=-j
export CLOUOPT
```

oder über die Builtin-Funktion **setopt()** im CLOU-Baustein:

```
...
#$ setopt("j", "ein")
#* automatisches Runden eingeschaltet #
...
#$ setopt("j","aus")
#* automatisches Runden wieder ausgeschaltet #
```

Bitte informieren Sie sich in Kapitel 3.2.3 und Kapitel 5.6 über Builtin-Funktionen und Aufruf-Optionen.

2.2 Stringvariablen

Stringvariablen haben vieles gemeinsam mit Rechenvariablen, so daß an vielen Stellen in diesem Kapitel auf das vorige verwiesen wird. Sie sollten sich daher mit den Rechenvariablen vertraut machen, bevor Sie hier weiterlesen.

Stringvariablen unterscheiden sich von Rechenvariablen vor allem darin, daß ihre Werte nicht Zahlen, sondern sogenannte "Strings" sind. Ein String ist eine Aneinanderreihung beliebiger Zeichen - Buchstaben, Ziffern und Sonderzeichen - zu einer Zeichenkette. Genau so, wie CLOU sich unter dem Namen einer Rechenvariablen eine Zahl merken kann, kann er sich unter dem Namen einer Stringvariablen eine solche Zeichenkette merken und sie an beliebiger Stelle in Ihr Dokument einfügen.

2.2.1 Definition von Stringvariablen

Auch Stringvariablen müssen definiert werden, bevor sie verwendet werden können. Die Definitionsanweisung für Stringvariablen hat denselben Aufbau wie die für Rechenvariablen:

Syntax

#D *VarName Wert* [*"Format"*]

Für *VarName* muß wieder ein Wort mit max. 30 Zeichen stehen; es gelten dieselben Regeln wie bei den Namen von Rechenvariablen.

Der *Wert* kann im einfachsten Fall eine String, darf aber auch ein beliebig komplizierter **Stringausdruck** sein, wie er im Kapitel 2.2.2 beschrieben ist. Ein String ist eine Aneinanderreihung beliebiger Zeichen, die zwischen doppelten Anführungszeichen (") stehen. Wichtig ist dabei, daß die Anführungszeichen nicht zum Wert der Variablen gehören: CLOU benötigt sie nur, damit er erkennen kann, wo der String anfängt und wo er aufhört. Wird der Wert der Variablen mit einer Ausgabeanweisung in Ihr Dokument eingefügt, so erscheinen im Text keine Anführungsstriche. Die folgenden Beispiele verdeutlichen dies:

Beispiel:

```
#D s1 "Ich bin ein String." * Wert:Ich bin ein String.#
#D s2 "Maier & Co"          * Wert:Maier & Co#
#D s3 "   "                 * Wert:   #
```

Der Wert der Variablen `s3` besteht z.B. nur aus drei Leerzeichen. Würde man bei ihrer Definition die Anführungszeichen weglassen, so könnte CLOU nicht wissen, wieviele Leerzeichen gemeint sind.

Häufig werden Variablen zu einem Zeitpunkt definiert, zu dem ihr späterer Wert noch unbekannt ist: er soll z.B. erst durch eine Benutzer-Eingabe festgelegt werden. In solchen Fällen setzen Sie bei Rechenvariablen den Wert 0 ein. Stringvariablen bekommen zur Unterscheidung den "leeren" String zugewiesen:

```
#D Name ""
#D Adresse ""
```

An den Leerstrings erkennt CLOU, daß es sich bei `Name` und `Adresse` nicht um Rechen- sondern um Stringvariablen handelt. Sollen Anführungszeichen zum Wert der Variablen selbst gehören, so müssen sie durch ein vorangestelltes \ -Zeichen, das Fluchtsymbol, kenntlich gemacht werden. Daneben ist es wie in früheren CLOU-Versionen auch erlaubt, die Anführungszeichen doppelt zu setzen:

```
#D s4 "Mein Name ist \"CLOU\"."
#D s5 "Mein Name ist ""CLOU""."
#* Wert in beiden Fällen: Mein Name ist "CLOU".#
```

Fehlerhaft sind die folgenden Anweisungen:

```
#D s5 blabla
#* die Anführungsstriche fehlen!
#

#D s6 "Mein Name ist "CLOU""
#* die Anführungsstriche vor und nach dem Wort CLOU
   müssen doppelt oder ein "\" davor stehen!
#
```

Auf das *Format* wird im Kapitel 2.2.3 "Ausgabe von Variablenwerten" eingegangen. Wie bei Rechenvariablen spielt es erst beim Einfügen des Variablenwertes in das HIT-Dokument eine Rolle und kann hier zunächst übergangen werden.

2.2.2 Stringausdrücke

Ebenso, wie Sie Zahlen und Rechenvariablen zu Rechenausdrücken verknüpfen können, deren Werte letztlich wieder Zahlen sind, können Sie auch Strings und Stringvariablen zu **Stringausdrücken** verknüpfen, deren Werte Strings sind. Stringausdrücke können bestehen aus:

- Strings,
- Stringvariablen und dem
- Zeichen &.

Zwischen diesen einzelnen Komponenten darf - wie bei Rechenausdrücken - beliebig viel Abstand gelassen werden, um die Ausdrücke übersichtlich zu gliedern; ein Ausdruck darf auch in diesem Fall über mehrere Zeilen gehen. Das &-Zeichen muß jeweils zwischen zwei Strings bzw. Stringvariablen stehen.

Mit dem &-Zeichen werden zwei Strings zu einem verbunden; man nennt dies auch "Verkettung" oder "Konkatenation". Ein Beispiel verdeutlicht das:

Beispiel:

```
#

#D Vorname  "Hans"
#D Nachname  "Maier"
#D Name  Vorname & Nachname

#* Der Wert der Variablen "Name" ist jetzt "HansMaier".
   Richtig ist also die folgende Anweisung:
#

#D Name Vorname & " " & Nachname

#* Jetzt hat die Variable "Name" den Wert "Hans Maier".
#

#D Anrede  "Sehr geehrter Herr " & Nachname

#* Die Variable "Anrede" hat jetzt den Wert:
   "Sehr geehrter Herr Maier"
#
```

Teilstrings

In Stringausdrücken können anstelle von Strings oder Variablen auch Teilstrings stehen. Während man mit dem Zeichen & einzelne Strings zu einem längeren verketten kann, bieten Teilstrings umgekehrt die Möglichkeit, aus dem Wert einer Stringvariablen Teilstücke oder einzelne Buchstaben "herauszubrechen".

Ein Teilstring hat den Aufbau:

VarName[*Index1*, *Index2*]

oder

VarName[*Index*]

Hier bedeuten die eckigen Klammern ausnahmsweise nicht, daß die entsprechenden Angaben fortgelassen werden können. Vielmehr müssen sie genau so im Bausteintext erscheinen, wie die Beispiele unten das zeigen. Für *Index1*, *Index2* oder *Index* können beliebige Rechenausdrücke stehen. *VarName* muß der Name einer Stringvariablen sein. Zwischen *VarName* und der ersten eckigen Klammer darf kein Leerzeichen stehen, ansonsten können beliebig viele Leerzeichen und Zeilenvorschübe eingestreut werden.

Die Zeichen, aus denen der Wert der Variablen *VarName* besteht, kann man sich von 1 an durchnumeriert denken. Aus dem Wert der Variablen *VarName* werden nun die Zeichen mit den Nummern *Index1* bis einschließlich *Index2* als Wert des Teilstrings verwendet. Steht nur eine Nummer *Index* in den Klammern hinter *VarName*, so besteht der Teilstring nur aus dem *Index*-ten Zeichen des Variablenwerts:

Beispiel:

```
#

#D s "MALEREI"

#D a s[1, 3]                    #* Wert: "MAL" #
#D b "NUDE" & s[3]              #* Wert: "NUDEL" #
#D c s[6, 7] & s[4, 5]          #* Wert: "EIER" #
```

Bereiche, die über den Wert der Variablen hinausgehen, werden durch Leerstrings ersetzt:

Beispiel:

```
#

#D s "MALEREI"

#D e s[-3, 2]                    #* Wert: "MA" #
#D f s[6, 1000]                  #* Wert: "EI" #
#D g s[10, 20]                   #* Wert: "" #
```

Hinweis:

Teilstrings spielen insbesondere bei der Bearbeitung bereits fertiger HIT-Dateien eine Rolle, wie sie in Kapitel 4.1 beschrieben ist. Dort finden Sie auch praktische Beispiele für ihre Anwendung.

2.2.3 Ausgabe von Variablenwerten

Stringvariablen werden mit der gleichen Anweisung in das Zieldokument ausgegeben wie Rechenvariablen:

Syntax

#> [*"Format"*] *VarName*

Mit dieser Anweisung sagen Sie: "Schreibe den aktuellen Wert der Variablen *VarName* an der Stelle, an der sich die Schreibmarke gerade befindet, in mein HIT-Dokument." Je nach dem angegebenen *Format* verwendet CLOU dabei führende Leerstellen, schreibt rechts- oder linksbündig usw., wie unten näher erläutert. Fehlt die Formatangabe, so wird das Format aus der Variablendefinition verwendet. Fehlt auch dieses, so wird der String unverändert (aber ohne begrenzende Anführungszeichen!) in das Zieldokument übernommen.

Das *Format* hat den Aufbau:

Syntax

[–][*Feldbreite*][*.MaxLänge*]

Für *Feldbreite* bzw. *MaxLänge* müssen jeweils ganze Zahlen eingesetzt werden. Die *Feldbreite* gibt an, wieviele Zeichen mindestens ausgegeben werden sollen. Wie bei Rechenvariablen wird ein String, der länger ist als *Feldbreite*, nicht abgeschnitten.

Hingegen gibt *MaxLänge* die maximale Anzahl von Zeichen an, die ausgegeben werden. Darüber hinausgehende Zeichen werden abgeschnitten.

Ist das erste Zeichen einer Formatangabe ein '-', so wird der String innerhalb des durch *Feldbreite* festgelegten Ausgabefeldes linksbündig ausgegeben.

Wie beim Datumsformat kann durch Attributierung des ersten Zeichens der Formatangabe eine Attributierung des ausgegebenen Wertes erzielt werden.

Beispiele:

Der Wert der auszugebenden Variablen sei "hallo".

Format	Ausgabe
`"8"`	`"   hallo"`
`"8.2"`	`"      ha"`
`"-8"`	`"hallo   "`
`"0"`	`"hallo"`

2.2.4 Manuelle Eingabe von Variablenwerten

Auch in Stringvariablen können Werte direkt mit der Tastatur eingegeben werden. Die Anweisung dafür ist die gleiche wie bei Rechenvariablen:

Syntax

#< VarName ["*Meldetext*"]

Bei dieser Anweisung hält CLOU mit der Bausteinverarbeitung an und fordert den Benutzer in der HIT-Kommandozeile dazu auf, einen beliebigen Text einzugeben. Die Variable *VarName* erhält dann diesen Text als Wert. Der *Meldetext* wird als Aufforderung in der Kommandozeile angezeigt. Fehlt der *Meldetext*, so wird stattdessen eine vordefinierte Standardmeldung ausgegeben. Anders als bei der Eingabe von Werten für Rechenvariablen können hier keine Benutzerfehler auftreten: während Buchstaben und Sonderzeichen in Zahlen nicht erlaubt sind, dürfen Strings jedes beliebige Zeichen enthalten.

> *Hinweis:*
>
> Auch hier sind die bedingte Form der Eingabe-Anweisung und die Eingabe-Anweisung mit Voreinstellung möglich; siehe hierzu Kapitel 2.1.4.

2.2.5 Wertzuweisung

Nachdem Sie eine Variable einmal definiert haben, können Sie ihren Wert jederzeit ändern:

Syntax

 # = VarName Ausdruck

Für *VarName* muß der Name einer vorher definierten Variablen stehen. *Ausdruck* kann ein beliebiger Stringausdruck sein, im einfachsten Fall auch nur ein String. Die Anweisung bewirkt, daß die Variable *VarName* den Wert von *Ausdruck* erhält.

Beispiel:

```
#
#D Name ""
#D Anrede ""

#< Name "Wie lautet der Name der Dame?"
#= Anrede "Sehr geehrte Frau " & Name & "!"

#* Angenommen, der Benutzer hat "Maier" eingegeben.
   Dann hat die Variable "Anrede" jetzt den Wert:
   Sehr geehrte Frau Maier!
#
```

Der *Ausdruck* darf auch den Namen der Variablen selbst enthalten. Damit kann man obiges Beispiel auch kürzer formulieren:

Beispiel:

```
#
#D Anrede ""

#< Anrede "Wie lautet der Name der Dame?"
#= Anrede "Sehr geehrte Frau " & Anrede & "!"

#* Das Resultat ist das gleiche wie oben.
#
```

2.2.6 Automatische Typkonvertierung

CLOU kennt zwei Typen von Variablen: Rechen- und Stringvariablen. Die Werte der einen sind Zahlen, die der anderen Strings (Zeichenketten). Diese dürfen nicht verwechselt werden: ein String, selbst wenn er nur aus Ziffern besteht, ist keine Zahl und kann deshalb nicht für Rechenoperationen wie z.B. die Addition verwendet werden.

Trotzdem dürfen in Rechenausdrücken Stringvariablen auftreten und umgekehrt und in Wertzuweisungen Werte des anderen Typs verwendet werden. In einem solchen Fall erkennt CLOU, daß der Wert der Stringvariablen in eine Zahl verwandelt (konvertiert) - werden muß bzw. der Wert der Rechenvariablen in einen String. Letzteres ist immer möglich, schließlich verwandelt CLOU eine Zahl ja auch bei der Ausgabe in das Zieldokument in eine Zeichenreihe, wobei er ihr Format berücksichtigt. Genauso geht er vor, wenn Sie einer Stringvariablen als Wert eine Rechenvariable zuweisen:

Beispiel:

```
#
#D Preis 3 "0.2"
#D s "Der Stückpreis beträgt "

#= s  s & Preis & " DM"
#> s

#* In Ihrem Dokument steht jetzt:
   Der Stückpreis beträgt 3.00 DM
#
```

CLOU hat also erkannt, daß die Variable `Preis` als Rechenvariable in einem Stringausdruck steht. Folglich hat er ihren Wert - gemäß ihrem Ausgabeformat mit zwei Nachkommastellen - in einen String verwandelt. Der umgekehrte Vorgang, die Verwandlung eines Strings in eine Zahl, ist hingegen nicht uneingeschränkt möglich. Voraussetzung hierfür ist, daß der String zumindest mit einer Zahl (oder einem Minuszeichen) beginnt. Ist dies nicht der Fall, so bricht CLOU den Einfügevorgang mit einer entsprechenden Fehlermeldung ab. Eventuell der Zahl nachfolgende Zeichen werden ignoriert:

Beispiel:

```
#

#D s "5 DM"
#D z 0

#= z 10 * s
#> z

#* CLOU fügt "50" in das Zieldokument ein.
#
```

2.2.7 Stringvariablen als Parameter

Viele CLOU-Anweisungen benötigen zusätzliche Angaben (Parameter) in Form von Strings, z.B. Meldungstexte, Bausteinnamen, Ausgabeformate usw., die - in Anführungszeichen eingeschlossen - Bestandteil der Anweisungs-Syntax sind. All diese Parameter können nun auch durch Stringvariablen ersetzt werden, wobei dem Variablennamen ein $-Zeichen vorangestellt werden muß. Dadurch ist es z.B. möglich, die Parameter-Werte während des Baustein-Einfügens vom Benutzer eingeben zu lassen:

Beispiel:

```
#
#* Das aktuelle Datum soll in das Dokument eingefügt
   werden. Der Benutzer soll das Datums-Format
   selbst wählen können:
#

#D format ""

#< format "Format für Datum:"
#T $format
```

Beim Einfügen dieses Bausteins erscheint in der Kommandozeile die Aufforderung `Format für Datum:`. Der Benutzer kann dann z.B. eingeben:

```
0T.0M.JJ
```

CLOU fügt das aktuelle Datum dann in diesem Format ein (in diesem Fall z.B.: `01.04.87`; zu den Datums-Formaten siehe Kapitel 1.2.6).

2.3 Variablentyp Liste

In einer Listen-Variable können Sie mehrere Stringausdrücke zusammenfassen. Dabei kommt es auf die Reihenfolge an, in der Sie die Stringausdrücke in die Liste schreiben, da die einzelnen Elemente der Listen-Variable später über einen Index (vgl. Teilstrings) angesprochen werden.

Listen-Variable können Sie verwenden,

- um Alternativen auszuwählen, die im CLOU-Baustein weiterverarbeitet werden sollen (im Gegensatz zur sofortigen Übernahme ins Dokument durch die `#A`-Anweisung).
- um die Auswahl komplexer Zusammenhänge auf die Auswahl von Schlagwörtern zu reduzieren (Zuordnung über Indices).
- wenn Sie externe Dateien oder die Ausgabe von shell-Kommandos verarbeiten wollen.

2.3.1 Definition von Listen-Variablen

Listen müssen - wie alle anderen Variablen - definiert werden, bevor Sie sie verwenden können. Dabei können Sie eine Listen-Variable aus einer Folge von Stringausdrücken aufbauen und/oder auf eine bestehende Liste zurückgreifen:

Syntax

#L *ListName* D {*String1*, *String2*, ... }
oder
#L *ListName* D *Liste*

Für *ListName* muß ein Wort mit max. 30 Zeichen stehen; es gelten dieselben Regeln wie bei den Namen von Rechenvariablen.

Für *String1* usw. setzen Sie einen beliebigen Stringausdruck (z.B. "hallo hier bin ich!" oder eine Stringvariable) ein; für *Liste* den Namen einer bereits definierten Listen-Variablen.

Sie können auch mehrere Stringfolgen und mehrere Listen zu einer neuen Liste zusammenfügen. Zwischen den Teilen, aus denen die neue Listen-Variable bestehen soll, muß dann jeweils das Zeichen '&' stehen. Ähnlich wie bei der Bildung von Stringausdrücken durch Verkettung, können Sie hier Stringfolgen und Listen zu einer umfassenden Liste "verketten".

Syntax

#L *ListName* D {*String1*, *String2*, ... } & *Liste* & ...

Beispiele:

```
#L Liste1 D {"1. Element", "2. Element"}
#L Liste2 D Liste1 & {"3. Element", "4. Element"}
```

Liste2 könnten Sie auch so definieren:

```
#L Liste2 D {"1. Element", "2. Element", "3. Element",
             "4. Element"}
```

Wenn Sie eine Listen-Variable definieren wollen, deren späterer Wert noch nicht bekannt ist - weil er erst durch den Benutzer angegeben wird, tragen Sie bei der Definition eine leere Stringfolge ein:

#L *ListName* D {}

Externe Datei / shell-Kommando als Liste

Alternativ zur Definitionsanweisung können auch HIT- und SINIX-Dateien sowie die Standardausgabe eines shell-Kommandos als Liste interpretiert werden. Dabei stellt jede Zeile jeweils ein Listen-Element dar.

Syntax

#L *ListName* H "*Dokumentname*"
#L *ListName* X "*Dateiname*"
#L *ListName* ! "*shell-Kommando*"

Für *Dokumentname* kann der Name eines beliebigen HIT-Dokuments eingesetzt werden. *Dateiname* steht für den Namen einer SINIX-Datei. An Stelle von *shell-Kommando* geben Sie ein Kommando des Betriebssystems an.

Bei der dritten Anweisung wird die globale CLOU-Variable RC wie bei der Anweisung #! mit dem Return-Code des letzten ausgeführten Shell-Kommandos besetzt.

Beispiel:

```
#L Inhalt ! "ls"

#* Listen-Elemente sind die Namen der im aktuellen
   Ordner befindlichen Dokumente
#
```

Datenbankanfrage als Liste

Ebenfalls alternativ zur Definitionsanweisung kann die folgende Anweisung stehen, mit der einer Liste das Ergebnis einer Datenbankanfrage zugewiesen wird.

Syntax

#L *ListName* Q "*Anweisung*"

Für *Anweisung* setzen Sie eine SELECT-Anweisung ein
(s. auch Kap. 4.3).

Jedes selektierte Feld bildet nach der Zuweisung ein Listen-Element. Werden mehrere Datensätze gefunden, so werden diese nacheinander in die Listen-Variable geschrieben, d.h wenn Ihre SELECT-Anweisung z.B. 5 Felder aus der Datenbank selektiert, beginnt mit den Listen-Elementen 6, 11, 16 usw. ein neuer Datensatz.

2.3.2 Liste ändern

Wollen Sie von einer bestehenden Liste nicht den Inhalt sondern die Länge oder die Reihenfolge der Elemente ändern, so müssen Sie die Liste neu definieren. Anweisungen existieren nur für die Spezialfälle:

- Erweitern einer Liste am Ende
- Sortieren

Liste erweitern

Bei der Erweiterung können neue Elemente an die bestehenden Elemente der Liste angehängt werden. Dabei sind alle von der Definitons-Anweisung bekannten Kombinationen möglich.

Syntax

#L *ListName* & {*String1*, *String2*, ... }
oder
#L *ListName* & *Liste*
oder
#L *ListName* & {*String1*, *String2*, ... } & *Liste* & ...

Beispiel:

```
#L Liste2 & {"5. Element"}
```

Liste sortieren

Mit der Funktion **sortieren** kann die Reihenfolge der Listen-Element geändert werden, ohne die Liste neu zu definieren. **Sortieren** ist z.B. dann sinnvoll, wenn Sie die Elemente einer Liste zur Auswahl anbieten wollen, die Sie aus anderen Listen zusammengestellt haben.

Syntax

#L *ListName* S

Hinweis:

Beim Sortiervorgang werden Großbuchstaben, Kleinbuchstaben und Umlaute gleich behandelt (z.B. alle o, O, ö, Ö, ô usw. vor p); 'ß' wird als 'ss' interpretiert. Vor den Buchstaben kommen die Ziffern und davor wiederum alle Sonderzeichen.

2.3.3 Bearbeitung einzelner Elemente einer Liste

2.3.3.1 Zugriff auf Listen-Elemente

Der Zugriff auf die einzelnen Listen-Elemente erfolgt - wie der Zugriff auf Teilstrings bei Stringvariablen - über Indices.

Syntax

ListName[*Index*]

Die eckigen Klammern gehören zur Syntax und dürfen nicht weggelassen werden. Für *Index* kann ein beliebiger Rechenausdruck stehen, jedoch darf *Index* nicht größer als die Listenlänge oder kleiner oder gleich 0 sein, sonst wird mit einer Fehlermeldung abgebrochen. Zwischen *ListName* und der ersten eckigen Klammer darf kein Leerzeichen stehen.

Die einzelnen Elemente der Liste kann man sich von 1 an durchnumeriert denken. Aus der Liste wird damit der an *Index*-ter Stelle stehende Stringausdruck ausgewählt.

Da Listen-Elemente Stringausdrücke sind, können alle String-Operationen - wie Verkettung, Teilstring-Bildung, usw. - auf indizierte Listen-Elemente angewandt werden.

Beispiele:

```
#D a14 Liste2[1] & " " & Liste2[4]
#* Wert: "1. Element 4. Element" #

#D b "6" & Liste2[5] [2, 7 ]
#* Wert: "6. Elem" #

#L Liste1 & { b }
#* Wert: "1. Element", "2. Element", "6. Elem" #
```

Länge einer Liste

Wenn Sie unabhängig von Benutzereingaben alle Elemente einer Liste in Ihrem CLOU-Baustein bearbeiten wollen (z.B. in einer gezählten Wiederholung), ist es wichtig zu wissen, wieviele Element die Liste enthält.

Syntax

#L *ListName* L *i*

Die Länge der Liste *ListName* wird in der Rechenvariablen *i* hinterlegt. Die Rechenvariable *i* kann als Abbruchkriterium für die gezählte Wiederholung (s. Kap. 3.1.4) dienen.

2.3.3.2 Auswahl einzelner Elemente über ein Element-Menü

Während der Bausteinverarbeitung können einzelne Elemente einer Liste über ein Element-Menü am Bildschirm ausgewählt werden. Je nachdem, welche der folgenden Anweisungen Sie in Ihrem CLOU-Baustein verwenden, können mehrere Elemente ausgewählt und die Indices in einer Index-Liste vereinigt werden, oder es kann nur genau ein Element ausgewählt werden, dessen Index dann in einer Rechenvariablen hinterlegt wird.

Syntax

#L *ListName* A *Indexliste*
oder
#L *ListName* A *i*

Für *Indexliste* setzen Sie den Namen einer bereits definierten Listen-Variable. Nach der Auswahl enthält *Indexliste* die Indices aller ausgewählten Listen-Elemente. *i* muß als Rechenvariable definiert sein und bekommt den Index des ausgewählten Listen-Elements zugewiesen.

Mit dieser Anweisung werden die Elemente der Liste *ListName* in Form eines HIT-Funktionsmenüs am Bildschirm zur Auswahl angeboten. Eine Auswahl treffen Sie wie in einem Objektauswahl-Menü, indem Sie mit der Taste [MENU] oder [F19] die Unterstreichung unter die ausgewählten Listen-Elemente setzen. Wird kein Element ausgewählt oder ist die Liste *ListName* leer, wird *Indexliste* mit der leeren Liste bzw. *i* mit 0 belegt.

2.3.3.3 Wertzuweisung, Ausgabe und Eingabe

Für die Wertzuweisung, die Ausgabe und die manuelle Eingabe von Werten existieren für Listen-Elemente keine eigenen Anweisungen. Da jedoch jedes Listen-Element gleichzeitig eine Stringvariable darstellt, können die bei Stringvariablen (Kap. 2.2) beschriebenen Anweisungen verwendet werden.

= *ListName*[*i*] *Ausdruck*
> ["*Format*"] *ListName*[*i*]
< *ListName*[*i*] ["*Meldung*"]

Ebenso wie bei Stringvariablen können indizierte Listen-Elemente auch in den Anweisungen `#X`, `#H`, `#Q` verwendet werden (s. Kap. 4).

2.3.4 Beispiel - Einfügen mehrerer Bausteine

```
#
#* Bausteine aus dem aktuellen Ordner einlesen #

#D i 0
#L bausteine ! "ls *"
#L einlesen D {}
#L bausteine A einlesen
#L einlesen L i

#S i > 0:
      #B $bausteine[einlesen[i]]
      #= i i-1
#
```

3 Komplexe Anweisungen

Bei aller Flexibilität, die Ihnen die Bausteinverarbeitung mit CLOU bietet, läuft das Einfügen eines einmal erstellten CLOU-Bausteins nach dem bisher Gesagten starr ab: CLOU befolgt Ihre Anweisungen in genau der Reihenfolge, in der sie im Baustein stehen. Gäbe es nicht die Eingabe-Anweisung, die eine Beeinflussung des Einfügetextes noch während des Einfügens ermöglicht, würde ein bestimmter CLOU-Baustein immer das gleiche Resultat erbringen.

Dies gilt aber nur für die in den vorangegangenen Kapiteln beschriebenen sog. **einfachen Anweisungen**. Demgegenüber bieten die **komplexen Anweisungen** Ihnen die Möglichkeit, den Ablauf des Baustein-Einfügens flexibel zu steuern. So können Sie CLOU z.B. anweisen, bestimmte Teile des Bausteins unter gewissen Bedingungen einfach zu übergehen oder auch mehrmals auszuführen. Wie Ihnen die Beispiele in diesem Kapitel zeigen werden, kann das eine enorme Aufwandsersparnis bedeuten.

Darüber hinaus bilden die in Kap. 3.1 beschriebenen Anweisungen die Grundlage für das Arbeiten mit HIT- und SINIX-Dateien sowie die Kommunikation mit Datenbanken, wovon in späteren Kapiteln die Rede sein wird.

3.1 Anweisungen mit Bedingungen

3.1.1 Die Fallunterscheidung

Mit den früher besprochenen Mitteln können Sie CLOU z.B. anweisen, von einem irgendwie berechneten Gesamtpreis 10% Rabatt abzuziehen. Nun wird der Rabatt aber üblicherweise erst ab einer gewissen Menge, z.B. ab 100 Stück gewährt. Damit Sie nun nicht zwei verschiedene CLOU-Bausteine schreiben müssen, wollen Sie CLOU sagen: "Wenn die Stückzahl größer als 99 ist, dann ziehe den Rabatt ab; andernfalls berechne den vollen Preis!". Dies ist mit der Fallunterscheidung möglich:

Syntax

```
#? Bedingung:
/J
    Ja-Zweig
/N
    Nein-Zweig
#
```

Für *Ja-Zweig* bzw. *Nein-Zweig* können Sie beliebigen Einfügetext sowie CLOU-Anweisungen einsetzen - einschließlich weiterer Fallunterscheidungen. Wenn CLOU auf eine solche Anweisung stößt, verhält er sich so: Zunächst prüft er, ob die angegebene *Bedingung* erfüllt ist. Ist dies der Fall, so führt er die unter `/J` stehenden Anweisungen aus bzw. fügt den dort stehenden Text in das Zieldokument ein, übergeht aber Anweisungen und Text im *Nein-Zweig*, d.h. er ignoriert alles, was zwischen `/N` und `#` steht. Ist die Bedingung hingegen nicht erfüllt, so übergeht CLOU den *Ja-Zweig* und behandelt stattdessen den *Nein-Zweig*. Sie können einen der beiden Zweige auch fortlassen oder ihre Reihenfolge vertauschen.

Das eingangs genannte Problem kann also so gelöst werden:

Beispiel:

```
#
#D Stückpreis 0
#D Stückzahl 0
#D Rabatt 10
#D Summe 0 "0.2"

#< Stückpreis "Stückpreis:"
#< Stückzahl "Stückzahl:"
#= Summe  Stückpreis * Stückzahl

#? Stückzahl > 99 :
/J
   #= Summe Summe * ((100-Rabatt)/100)
   Gesamtpreis abzüglich 10% Rabatt: #> Summe DM
/N
   Gesamtpreis: #> Summe
#
```

Als *Bedingung* steht in diesem Beispiel `"Stückzahl > 99"`. Im folgenden Kapitel erfahren Sie mehr darüber, wie Bedingungen allgemein aussehen. In diesem Fall heißt die Bedingung einfach: "*Wenn* der Wert der Variablen `Stückzahl` größer als 99 ist, dann führe den *Ja-Zweig* aus; sonst den *Nein-Zweig*. Der Rabatt wird also nur dann vom Gesamtpreis abgezogen, wenn die Stückzahl größer als 99 ist.

Bitte beachten Sie:

Falls das letzte Zeichen des ersten Zweiges eine Ziffer ist, muß ein Trenn-Kommentar folgen, damit das folgende /-Zeichen nicht als Divisionszeichen interpretiert wird.

3.1.2 Bedingungen

Eine Bedingung besteht aus zwei Rechen- oder zwei Stringausdrücken, zwischen denen ein Vergleichszeichen steht. So bedeutet z.B. `Stückzahl > 99`: "Der Wert der Variablen `Stückzahl` ist größer als 99". Diese Bedingung kann nun zutreffen oder nicht, je nachdem was der Wert der Variablen tatsächlich ist.

Es stehen folgende Vergleichszeichen zur Verfügung:

<	kleiner,
< =	kleiner oder gleich,
=	gleich,
> =	größer oder gleich,
>	größer,
< >	ungleich.

Die folgenden Beispiele demonstrieren die Bedeutungen der Vergleichszeichen im Einzelnen (die Variable `Preis` habe dabei den Wert 3):

Beispiele:

`Preis < 4`	trifft zu, denn 3 ist kleiner als 4.
`Preis < 2+1`	trifft nicht zu, denn 3 ist nicht kleiner als 1 + 2 (sondern gleich).
`3 <= Preis`	trifft zu, denn der linke Ausdruck muß nur kleiner oder gleich dem rechten Ausdruck sein.
`4+5 <> Preis`	trifft zu, denn 4 + 5 ist ungleich 3.

Anstelle von Rechenausdrücken können Sie auch Strings oder Stringausdrücke vergleichen. Solange es dabei nur um Gleichheit oder Ungleichheit geht, ist die Sache einfach:

Beispiele:

`"Henne" <> "Hahn"`	trifft zu.
`"Hahn" = "Gockel"`	trifft nicht zu.
`"ha" & "ha" = "haha"`	trifft zu.

Eventuell gesetzte Attribute werden dabei ignoriert; so gelten z.B. die Strings *Hahn* und Hahn als gleich. Wollen Sie die strenge Gleichheit zweier Strings prüfen, so setzen Sie das doppelte Gleichheitszeichen:

`"Henne" = "Henne"`	trifft zu.
`"Henne" == "Henne"`	trifft nicht zu.
`"Henne" == "Henne"`	trifft nicht zu.

Etwas komplizierter ist der Vergleich zweier Strings auf kleiner bzw. größer. Prinzipiell gilt der String als kleiner, der in einem Lexikon vor dem anderen stehen würde:

Beispiele:

`"Elefant" < "Maus"`	trifft zu ("Elefant" steht vor "Maus").
`"Herrgott" < "Herr"`	trifft nicht zu ("Herrgott" steht nach "Herr").
`"Wiege" >= "Liege"`	trifft zu.

Probleme treten jedoch bei der Groß-/Kleinschreibung sowie bei der Verwendung von Umlauten auf. Die den Stringvergleichen zugrunde liegende Reihenfolge richtet sich nach dem T61-Alphabet. Der Tabelle im Anhang können Sie entnehmen, daß sämtliche Großbuchstaben vor den Kleinbuchstaben eingeordnet sind; die Umlaute folgen nach allen anderen Buchstaben. Wie nachfolgende Beispiele zeigen, kann das zu verwirrenden Ergebnissen führen. In Zweifelsfällen ziehen Sie daher bitte immer die Tabelle zu Rate:

`"Zorro" < "aachen"`	trifft zu!
`"Ähre" < "Ehre"`	trifft nicht zu!

Verknüpfung von Bedingungen

Mehrere Bedingungen können zu komplexen Bedingungen verknüpft werden, indem man sie mit den Worten **und** bzw. **oder** verbindet:

`z > 0 und z < 3`	Diese Bedingung trifft nur zu, wenn der Wert der Variablen z größer als 0 und kleiner als 3 ist, d.h. beide Teilbedingungen müssen erfüllt sein.
`z = 1 oder z = 2 oder z = 3`	z muß einen der Werte 1, 2 oder 3 haben, damit die Bedingung zutrifft. Eine der Teilbedingungen muß erfüllt sein.

Komplexe Bedingungen können Sie überall einsetzen, wo auch einfache erlaubt sind. Klammerung von komplexen Bedingungen ist nicht möglich; stattdessen kann man z.B. verschachtelte Fallunterscheidungen verwenden. Werden mehrere Bedingungen miteinander verknüpft, so werden sie von links nach rechts abgearbeitet.

Bedingte Eingabe von Variablenwerten

Mit Hilfe von Bedingungen können Sie eine erweiterte Form der schon früher (Kapitel 2.1.4 bzw. 2.2.4) eingeführten Eingabeanweisung benutzen:

Syntax

#< "Bedingung:" VarName ["Meldetext"]

In dieser Form bewirkt die Anweisung, daß der Benutzer solange zur Eingabe aufgefordert wird, bis die angegebene *Bedingung* erfüllt ist. Dadurch können Sie verhindern, daß der Benutzer versehentlich unsinnige Werte eingibt.

Beispiel:

```
#

#D Preis 0
#< "Preis > 0 :"  Preis  "Bitte Preis eingeben"
#* Negative Zahlen werden abgewiesen.
#

#D Antwort ""
#< "Antwort = \"j\" oder Antwort = \"n\" :" Antwort
   "Bearbeitung abbrechen (j/n):"
#* Nur die Eingaben "j" oder "n" werden zugelassen.
#
```

Ja/Nein-Abfragen

Das letzte Beispiel kann mit Hilfe der Spezialbedingung "jn" auch kürzer geschrieben werden:

Beispiel:

```
#D Antwort ""
#< "jn" Antwort "Bearbeitung abbrechen:"
```

Die Bedingung `"jn"` erlaubt nur die Eingabe von `j` oder `n` bzw. `J` oder `N`; alle anderen Eingabe werden abgewiesen. Bei Rechenvariablen wird die Bedingungen `"jn"` ignoriert.

3.1.3 Die Bedingte Wiederholung

Mit der **Bedingten Wiederholung** können Sie eine oder mehrere CLOU-Anweisungen solange wiederholen bzw. einen Text so oft einfügen lassen, wie eine bestimmte Bedingung zutrifft.

Syntax

```
#S Bedingung:
    Schleifenkörper
#
```

Der Kennbuchstabe "S" steht für "Schleife", wie die Bedingte Wiederholung auch kurz genannt wird. Wie bei der Fallunterscheidung darf auch hier der Doppelpunkt hinter der *Bedingung* nicht fehlen. Der Schleifenkörper kann aus beliebig vielen CLOU-Anweisungen - auch weiteren Wiederholungs-Anweisungen - sowie beliebigem Einfügetext bestehen. Dieser Schleifenkörper wird nun solange immer wieder von CLOU interpretiert, bis die angegebene *Bedingung* nicht mehr erfüllt ist. Trifft die *Bedingung* von Anfang an nicht zu, so wird der Schleifenkörper von CLOU übergangen, also auch nichts in Ihr Dokument eingefügt.

Beispiel:

```
#

#* Berechnung der Summe von beliebig vielen Einzel-
   preisen. Der Benutzer wird solange immer wieder
   zur Eingabe aufgefordert, bis er mit 0 (Null)
   antwortet. Die Einzelpreise und ihre Summe wer-
   den untereinander ins Zieldokument geschrieben.
#

#D preis 0 "8.2"
#D summe 0 "8.2"
#D meldung "Bitte Preis eingeben (0 für Ende):"

#< preis $meldung

#S preis <> 0 :
   #> preis
   #^"RETURN"
   #= summe  summe + preis
   #< preis $meldung
#
========
#^ "RETURN"
#> summe
```

3.1.4 Die Gezählte Wiederholung

Wenn es darum geht, einen Teil eines Bausteins z.B. genau zehnmal von CLOU abarbeiten zu lassen, dann ist die **Gezählte Wiederholung** die einfachste Lösung:

Syntax

```
#W Anzahl
   Schleifenkörper
#
```

Der Kennbuchstabe "W" steht für "Wiederholung". Für Anzahl darf eine Zahl oder der Name einer Rechenvariablen, jedoch kein Rechenausdruck stehen. Für den *Schleifenkörper* gilt das im vorigen Kapitel Gesagte. Er wird sooft durchlaufen, wie es die Zahl bzw. der Wert der Variablen vorschreibt. Ist die *Anzahl* gleich Null oder negativ, so wird der Schleifenkörper übergangen.

Beispiel:

```
#
#W 10
    X
#

#* Einfügetext:
X X X X X X X X X X
#
```

Den gleichen Effekt könnten Sie auch mit Hilfe der **Bedingten Wiederholung** erzielen, was aber mehr Aufwand erfordert:

```
#D Zähler 0
#S Zähler < 10:
    X
    #= Zähler  Zähler+1
#
```

3.1.5 Die Mehrfachauswahl

Diese Anweisung dient dazu, in Abhängigkeit von einem Variablenwert eine von mehreren Alternativen auszuwählen.

Syntax

```
#C VarName
/ Wert1:
    Zweig1
/ Wert2:
    Zweig2
...
/ WertN:
    ZweigN
[ /:
    Sonst-Zweig]
#
```

VarName steht für eine beliebige Rechen- oder Stringvariable. Für *Wert1* bis *WertN* können Zahlen, Strings oder Variablen eingesetzt werden. CLOU befolgt diese Anweisung so:
Der Wert der Variablen *VarName* wird - falls es sich um eine Rechenvariable handelt - zunächst auf die nächstkleinere ganze Zahl abgerundet. Dann wird er der Reihe nach mit den Werten *Wert1*, *Wert2* usw. bis *WertN* verglichen. Stimmt ein Wert mit dem von *VarName* überein, so wird der darunter stehende *Zweig* ausgeführt, der wie üblich aus Einfügetext und CLOU-Anweisungen bestehen kann. Stimmt *VarName* mit mehreren Werten überein, so wird nur der erste "passende" Zweig ausgeführt.

Stimmt *VarName* mit keinem der Werte überein, so wird - falls vorhanden - der *Sonst-Zweig* ausgeführt. Fehlt in einem solchen Fall der *Sonst-Zweig*, so wird die gesamte Anweisung übersprungen.

Beispiel:

```
#D i 0
#< i "bitte wählen: 1) männl. / 2) weibl. / 3) Kind"
#C i
/1:
    Sehr geehrter Herr,
/2:
    Sehr geehrte Dame,
/3:
    Liebes Kind,
/:
    Sehr geehrte Damen und Herren,
#
```

Bitte beachten Sie:

Falls das letzte Zeichen eines Zweiges eine Ziffer ist, muß ein Trenn-Kommentar folgen, damit das folgende /-Zeichen nicht als Divisionszeichen interpretiert wird.

3.2 Variable Anweisungen

3.2.1 Makros

Stringvariablen können nicht nur die Stelle von Parametern innerhalb von CLOU-Anweisungen übernehmen, sondern sogar eine oder mehrere ganze Anweisungen ersetzen. Dazu definieren Sie eine Stringvariable mit einem Wert, der seinerseits CLOU-Anweisungen enthält. CLOU arbeitet diese noch nicht ab, da er zwischen Anführungszeichen stehende Textpassagen nicht interpretiert. Um die in der Stringvariablen enthaltenen Anweisungen ausführen zu lassen, setzen Sie die Makro-Anweisung:

Syntax

#$ VarNarne

VarName ist der Name einer Stringvariablen, deren Wert aus einer oder mehreren CLOU-Anweisungen besteht.

Makros sind vor allem dann sinnvoll, wenn Sie längere Anweisungen oder Anweisungsfolgen mehrmals innerhalb eines Bausteins benötigen.
Dazu ein Beispiel:
Der Benutzer soll an verschiedenen Stellen einen Zwischentitel eingeben können. Dieser soll jedesmal zentriert und unterstrichen in das Zieldokument eingefügt werden.

Beispiel:

```
#
#D TITEL "#^ \"UNTERSTREICHEN\"
          #^ \"ZNT\"
          #F \"Bitte Titel eingeben:\"●
          #^ \"UNTERSTREICHEN\"
          #^ \"LBD\"
          #^ \"RETURN\""
```

Statt die ganze Anweisungsfolge nun an mehreren Stellen im CLOU-Baustein zu wiederholen, können Sie einfach jedesmal die Makro-Anweisung

```
#$ TITEL
```

einsetzen - CLOU führt an jeder dieser Stellen alle fünf Anweisungen durch, aus denen die Makro-Variable `TITEL` besteht.

Variable Makros

Sie können Makros auch variabel gestalten, indem Sie bei der Definition eines Makros bestimmte Ausdrücke durch Platzhalter ersetzen. Welche Werte anstelle der Platzhalter eingesetzt werden sollen, müssen Sie CLOU beim Aufruf des Makros mitteilen:

Syntax

#$ VarName(Wert1, Wert2, ...)

Zwischen VarName und der ersten runden Klammer darf kein Leerzeichen stehen. Für *Wert1* usw. setzen Sie beliebige String- oder Rechenausdrücke ein, die anstelle der Platzhalter im Makro abgearbeitet werden sollen.

Die Platzhalter im Makro müssen dazu von 1 bis 9 durchnummeriert sein. Daß es sich um einen Platzhalter handelt, teilen Sie CLOU mit, indem Sie der Ziffer das $-Zeichen voranstellen, d.h. Sie tragen im Makro der Reihe nach `$1`, `$2` usw. ein. Vergessen Sie dabei nicht, Platzhalter für Stringausdrücke in Anführungszeichen zu setzen. Die Zuordnung der Werte zu den Platzhaltern erfolgt in der Reihenfolge, in der Sie die Werte in den Makro-Aufruf schreiben.

Beispiel:

```
#D TITEL        "#^ \"$1\"
                 #^ \"ZNT\"
                 #F \"Bitte Titel eingeben:\"●
                 #^ \"$1\"
                 #^ \"LBD\"
                 #^ \"RETURN\""
```

Beim Aufruf geben Sie dann das gewünschte Attribut an.

```
#$ TITEL("FETT")
```

Bitte beachten Sie:

Wenn Sie beim Aufruf weniger Werte angeben, als Platzhalter vorhanden sind, wird mit Fehlermeldung abgebrochen; bei zu vielen Werten werden die überschüssigen ignoriert.

> *Hinweis:*
>
> Wenn Sie in CLOU-Bausteinen HIT-Funktions-Menüs bearbeiten wollen, dann sollten Sie dies nach Möglichkeit in einem zentralen Baustein abwickeln, der bei einer Änderung des Funktions-Menüs leicht angepaßt werden kann. Variable Makros (oder Funktionen) ermöglichen Ihnen, die Programmierung flexibel zu gestalten.

3.2.2 Funktionen

CLOU-Funktionen sind ähnlich aufgebaut wie Makros - sie sind Stringvariable, deren Wert seinerseits CLOU-Anweisungen enthält. Anders als Makros, die nach der Definition einfach über die **Makro-Anweisung** aufgerufen und abgearbeitet werden, liefern Funktionen einen Wert zurück ("Return-Wert").

Das Zurückliefern erfolgt über die **Rückkehr-Anweisung**.

Syntax:

#R *Wert*

Für *Wert* können Sie einen beliebigen Rechen- oder Stringausdruck einsetzen. Dabei bestimmt der Typ des Ausdrucks *Wert* den Typ der ganzen Funktion.

Zur Unterscheidung von Makros muß bei der Definition einer Funktion unmittelbar nach dem Funktions-Namen das Klammernpaar () folgen.

Beispiel:

```
#D i 0
#D lfdnr() "#= i i+1
           #R i"
```

Die Funktion `lfdnr()` liefert der Reihe nach die Zahlen 1, 2, 3 usw.

Eine Funktionen kann - je nach Typ von *Wert* -

- eine Rechenvariable oder
- eine Stringvariable ersetzen.

An allen Stellen, an denen einer Variablen ein Wert zugewiesen bzw. auf den Wert einer Variablen zugegriffen wird, kann daher auch eine Funktion stehen. Bei Funktionen, die einen String zurückliefern, kann zusätzlich die Teilstring-Operation angewandt werden.

Beispiel:

```
#> lfdnr()
#= j 7 * lfdnr()
```

Beim Aufruf - nicht bei der Definition - von argumentlosen Funktionen kann die Angabe des Klammernpaares entfallen.

Beispiel:

```
#> lfdnr
```

Variable Funktionen

Wie Makros können auch Funktionen variabel gestaltet werden. Für Platzhalter und Werte gelten dieselben Regeln (s. Kap. 3.3.1). Die Werte müssen bei jedem Funktions-Aufruf angegeben werden.

Beispiel:

```
#* Berechnung der Fakultät: #

#D fac() "#? $1 = 0:
          /J
                #R 1 #*#
          /N
                #R $1 * fac($1-1)
          #"
```

Der Aufruf könnte dann lauten:

```
Fakultät von 4: #> fac(4)
```

Wie das Beispiel zeigt, können Funktionen rekursiv aufgerufen werden. Sie sollten allerdings berücksichtigen, daß dadurch ein mehrfaches an Speicherplatz und an Rechenzeit verbraucht wird.

Hinweis:

Wird die #R-Anweisung außerhalb einer Funktion verwendet, so bewirkt sie den Abbruch des betreffenden Makros oder Bausteins.

3.2.3 Standard-Funktionen

Zusätzlich zu den Funktionen, die Sie selbst definiert haben, stellt Ihnen CLOU eine Reihe von Standard-Funktionen zur Verfügung. Die Namen der Standard-Funktionen sind fest vorgegeben und dürfen nicht verändert werden. Sie dürfen auch nicht für selbst definierte Variable vergeben werden!

Abgesehen von dem fest definierten Namen und der genau vorgegebenen Wirkungsweise unterscheiden sich Standard-Funktionen nicht von normalen Funktionen:

- Sie werden genauso aufgerufen.
- Sie erwarten eine bestimmte Anzahl von Argumenten.
- Sie liefern entweder eine Zahl oder einen String zurück.

CLOU kennt folgende Standard-Funktionen:

Dokument- und Bausteinnamen

`DOKUMENT`	Name des aktuellen Dokuments
`BAUSTEIN`	Name des aktuellen Textbausteins
`BAUSTEINPFAD`	Pfadname des aktuellen Textbausteins

Die Funktionen werden ohne Argument aufgerufen und liefern die Namen als Strings zurück.

Löschpuffer

`C_LÖPU`	zuletzt gelöschtes Zeichen
`TW_LÖPU`	zuletzt gelöschtes Teilwort
`W_LÖPU`	zuletzt gelöschtes Wort
`TZ_LÖPU`	zuletzt gelöschte Teilzeile
`Z_LÖPU`	zuletzt gelöschte Zeile

Die Funktionen werden ohne Argument aufgerufen und liefern die Namen als Strings zurück. Ist einer der Löschpuffer nicht besetzt, wird ein Leerstring zurückgeliefert.

HIT-Funktionsmenü 'Suchen und Ersetzen'

Wie im Kapitel 3.2.1 erwähnt, lassen sich HIT-Funktions-Menüs komfortabel über CLOU-Funktionen oder -Makros bedienen. Standardmäßig stellt Ihnen CLOU die Funktion **suchen** zur Bedienung des Funktions-Menüs **Suchen und Ersetzen** zur Verfügung.

`suchen` (*suchtext*, *ersatztext*, *ersetzungen*,*großschreibung*, *suchattribute*, *nachformatieren*,*suchrichtung*, *textattribute*, *bestätigen*)

Als Argumente können sie folgende String- bzw. Rechenausdrücke einsetzen:

suchtext, *ersatztext*	Suchmuster und Ersatztext als Strings.
ersetzungen	gewünschte Anzahl; für beliebig viele Ersetzungen kann "*" oder irgendeine negative Zahl stehen.
großschreibung *such-, textattribute*	`"gültig"` oder `"ungültig"`; bei Angabe des Leerstrings "" wird der bestehende Eintrag nicht verändert.
nachformatieren *bestätigen*	`"ja"` oder `"nein"`; "" wie oben.
suchrichtung	`"vorwärts"` oder `"rückwärts"` "" wie oben.

Als Return-Wert wird die Anzahl der ersetzten Wörter zurückgeliefert, falls *ersetzungen* nicht '0' gesetzt war. Andernfalls wird 1 zurückgeliefert, falls das Suchwort gefunden wurde, 0, falls es nicht existiert.

Alle Argumente außer *suchtext* sind optional und werden bei Bedarf durch den jeweils letzten Wert ersetzt. Beachten Sie jedoch, daß die Argumente genau in der oben definierten Reihenfolge angegeben werden müssen. Wenn Sie z.B. das 2. und das 3. Argument nicht benötigen, jedoch das 4. Argument *großschreibung*, so setzen Sie für das 2. und 3. Argument einfach den Leerstring "" ein. Fehlt die Angabe von *ersetzungen* wird stattdessen 0 eingesetzt.

Beispiel:

```
#? suchen("Hit", "HIT", -1, "ungültig", "ungültig",
          "nein", "", "ungültig", "ja") = 0:
/J
        #M "Suchwort kommt nicht vor!"
/N
        #> suchen("HIT", "Textsystem HIT", -1)
#
```

Wenn der Return-Wert nicht benötigt wird, kann anstelle von suchen ("*suchtext*") auch *#^* "SUCHEN" "*suchtext*" stehen.

Variablen-Funktionen

`strlen(`*string*`)`

liefert als Return-Wert die Länge des Stringausdrucks *string*.
`strlen("hallo")` ergibt z.B. `5`.

`isdef("`*VarName*`")`

prüft, ob die Variable *VarName* bereits definiert wurde. Folgende Return-Werte sind möglich:

0 nicht definiert
1 Rechenvariable
2 Stringvariable
3 Listen-Variable
4 Funktion mit Return-Wert *zahl*
5 Funktion mit Return-Wert *string*
6 Standard-Funktion mit Return-Wert *zahl*
7 Standard-Funktion mit Return-Wert *string*

`#> isdef("isdef")` liefert z.B. `6`

`SHELLVARIABLE("`*Name*`")`

liefert den Inhalt der angegebenen Shell-Variablen *Name* zurück; falls sie nicht gesetzt ist, wird der Leerstring "" zurückgeliefert.

`extern("`*Shell-Kommando*`")` ruft das angegebene Shell-Kommando auf und liefert als Return-Wert den Exit-Status des zuletzt ausgeführten Shell-Kommandos zurück.
Die Standard-Ein-/Ausgabe ist dabei mit der Datei verbunden, die auch HIT verwendet (also i.d.R. `"/dev/tty"`).
Diese Standard-Funktion wird für den Aufruf interaktiver Shell-Kommandos verwendet.

`exit("`*Meldetext*`")` beendet CLOU und gibt den Meldetext aus.

`listlen("`*LVarname*`")` liefert als Return-Wert die Länge der angegebenen Listenvariablen *LVarName*.

```
#L Liste1 D {"1.Element","2.Element"}
#> listlen("Liste1")
```

liefert z.B. 2.

`setatt("`*String*`", "`*Attribut*`" [,`*l*`[,`*r*`]])`
setzt das *Attribut* im *String*.

`delatt("`*String*`", "`*Attribut*`" [,`*l*`[,`*r*`]])`
löscht das *Attribut* im *String*.

Das Attribut muß mit seinen Tastennamen analog zur `#^`-Anweisung angegeben werden. *l* bezeichnet den Index des ersten Zeichens, dessen Attribut verändert werden soll, *r* den des letzten. Fehlen beide Angaben, wird der ganze *String* attributiert. Fehlt nur *r*, wird nur das *l*.te Zeichen attributiert. Als Ergebnis wird der entsprechend modifizierte *String* zurückgeliefert.

```
#> setatt("Text", "UNTERSTREICHEN", 1)
```

liefert z.B. T̲ext.

`index`(*string, zeichen*)
`rindex`(*string, zeichen*)

bestimmen den Index eines Zeichens *zeichen* in einem Stringausdruck *string*. *index* sucht dabei von vorne, *rindex* von hinten (reverse).
Ist die Länge von *zeichen* größer als 1, wird das erste Zeichen von *zeichen* verwendet. Wird das Zeichen nicht gefunden, wird als Return-Wert 0 zurückgeliefert.

```
#> index("hallo", "a")
```
liefert z.B. 2.

Datums-Funktionen

`today`

liefert das aktuelle Datum im internen UNIX-Format als Zahl zurück. Das interne Format entspricht der Darstellung von UNIX, also der Anzahl der seit dem 1.1.1970, 00.00.00 Uhr vergangenen Sekunden.
Es hat den Vorteil, daß damit sehr einfach die Reihenfolge verschiedener Datumsangaben durch Kleiner- bzw. Größer-Vergleich festgestellt werden kann. Für die Konvertierung zwischen internem und externem Format existieren die folgenden zwei Funktionen.

`fdate(`*datum*, "*format*"`)`

liefert das als Zahl (internes UNIX-Format) angegebene *datum* im bezeichneten *Format* als String zurück. Fehlt die Formatangabe, wird das Standardformat (wie bei `#T`) verwendet,
d.h. `#> fdate(today)` hat dieselbe Wirkung wie `#T`.
Bei fehlerhafter *datum*-Angabe wird der Leerstring "" zurückgeliefert.

```
#> fdate (today - 24 * 60 * 60)
```

liefert z.B. das gestrige Datum

`idate("`*datum*`")`

liefert die Interndarstellung des angegebenen Datums als Zahl zurück. *datum* muß folgendes Format besitzen: "J0M0T0h0m0s". Fehlende Stellen werden automatisch mit '0' aufgefüllt, z.B.

```
#> fdate(idate("880526"), "J0M0T")
```

liefert wieder 880526.

Bei fehlerhafter *datum*-Angabe wird -1 zurückgeliefert. Aus technischen Gründen funktioniert `idate()` nur für Datumsangaben, die zwischen 1970 und ca. 1998 liegen.

Konvertierungs-Funktionen

`ctoi("`*zeichen*`")`	liefert den Dezimalwert von *zeichen* zurück. Ist der angegebene String *zeichen* länger als 1 Zeichen, wird der Rest ignoriert; ist er leer, wird 0 zurückgeliefert.
`itoc(`*zahl*`)`	Umkehrfunktion zu `ctoi()`. `itoc()` liefert zu einer Zahl das zugehörige Zeichen als String zurück. Entspricht *zahl* keinem HIT-Zeichen, wird der Leerstring "" zurückgeliefert, z.B. `itoc(ctoi("a"))` liefert wieder "a"; dagegen liefert `ctoi(itoc(2000))` den Wert 0, da es das kein Zeichen mit dem Dezimalwert 2000 gibt.
`xtoi("`*hex-zahl*`")`	liefert den Dezimalwert des angegeben Strings *hex-zahl* zurück; falls eine Konvertierung nicht möglich ist, wird -1 zurückgeliefert.
`itox(`*zahl*`)`	Umkehrfunktion zu `xtoi()`. `itox()` liefert den Hexadezimalwert von *zahl* als String zurück.

Beachten Sie bei den Konvertierungs-Funktionen, daß Zeichen mit Akzenten (2.Teil der T61-Tabelle im Anhang) einem dreistelligen Hexadezimalwert entsprechen; z.B. ä : 0x156 oder ê:0x161.

Signalbehandlungs-Funktionen

Mittels einiger weiterer Standard-Funktionen können mit CLOU Signale analog zur Programmiersprache C verarbeitet werden. Eintreffende Signale werden mit Ausnahme der DEL-Taste und Floiting Point Exceptions standardmäßig ignoriert. Die beiden angegebenen Signale führen wie bisher zum Abbruch mit einer entsprechenden Fehlermeldung.

`signal`(*sig*[,"*CLOU-Anweisungen*"])

ordnet dem Signal mit der Signalnummer *sig* eine neue Signalroutine zu.
Beim Eintreffen eines Signals werden die angegebenen CLOU-Anweisungen ausgeführt. Diese Zuordnung gilt solange, bis entweder für dieses Signal eine neue Routine zugeordnet oder das Einfügen des Bausteins beendet wird.
Wird als Signalroutine ein Leerstring "" übergeben, so wird das angegebene Signal ignoriert.
Fehlt der 2. Parameter ganz, wird wieder die Standardroutine von CLOU eingestellt.

Die Signalnummer *sig* muß zwischen 1 (SIGHUP) und 15 (SIGTERM) liegen. Die Behandlung der Signale 8 (SIGFPE) und 9 (SIGKILL) kann nicht verändert werden.
Als Return-Wert wird die bisherige Signalroutine, im Fehlerfall ein Leerstring zurückgeliefert.
Z.B. geschieht das Ignorieren der DEL-Taste durch den Aufruf:
#$ `signal`(2,"").

`kill(`*pid, sig*`)` sendet das angegebene Signal mit der Signalnummer *sig* an den Prozeß mit der Prozeßnummer *pid*.
Folgende Return-Werte sind möglich:

0	es trat kein Fehler auf
-1	falls der angegebene Prozeß nicht existiert oder einem anderen Benutzer gehört oder falls die Signalnummer nicht korrekt ist.

`pause()` legt CLOU bis zum Eintreffen eines Signals schlafen.

Beispiel:

```
#
#* Testbaustein zur Signalbehandlung in CLOU
#

#* 1. Testfall:
   Es wird solange "Hallo!" in den Text ausgegeben, bis der
   Benutzer die  DEL -Taste (Signal 2) drückt.
#

#D TRUE 1
#D FALSE 0
#D continue TRUE

#$ signal (2, "#= continue FALSE")

#S continue = TRUE:
   Hallo!
   #+
#

#F "weiter:  ↵ "

#* 2. Testfall:
   CLOU ruft ein externes Programm im Hintergrund auf und legt
   sich schlafen. Das Programm schickt ihm nach einiger Zeit
   ein Signal zu und weckt ihn damit wieder auf.
```

```
#

#D cmd "sleep 2; kill -14 " & getpid() & " &"
#$ signal (14, "ok!#^\"RETURN\"")
#! $cmd
#$ pause

#F "weiter:  ↵ "

#* CLOU kann sich auch selbst ein Signal zuschicken.
#

#$ kill (getpid(), 14)

#* 3. Testfall:
   Die DEL -Taste wird während der interaktiven Eingabe
   ignoriert.
#

#$ signal (2, "")
#F "Jetzt wird DEL  ignoriert!"

#* 4. Testfall:
   Die Standard-Signalbehandlung für die DEL -Taste wird
   wieder eingestellt.
#

#$ signal (2)
#F "Wenn jetzt  DEL  + ↵  gedrückt wird, ist es aus!"

#$ exit ("Mit exit ist es auf jeden Fall aus!")
```

Sonstige Funktionen

`getpid()` liefert die Prozeßnummer des CLOU als Zahl zurück.

`setmsgtext(`*anweisung, meldetext*`)`
stellt den Default-Meldetext für einige Anweisungen ein.
Für *anweisung* sind folgende Angaben erlaubt:
"A" für die Alternativenauswahl (#A)
"B" für die Bausteinauswahl (#B)
"L" für die Listenauswahl (#L *LName* A)
Der bisherige Meldetext wird als String zurückgeliefert. Im Fehlerfall ist dies der Leerstring.

`setopt(`*option, modus*`)` stellt CLOU-Optionen ein.
option darf einen der folgenden Werte haben: "N", "O", "v", "8", "7", "j".
Die Beschreibung der einzelnen Optionen finden Sie im Kapitel 5.6.
modus muß entweder "ein" oder "aus" sein.
Der bisherige Wert wird als String zurückgeliefert. Im Fehlerfall ist dies der Leerstring.

3.3 Variablenwerte merken

Falls Sie einmal ermittelte bzw. eingegebenen Werte von CLOU-Variablen nicht nur für die aktuelle Bearbeitung benötigen, sondern auch am nächsten Tag für einen weiteren Prozeßschritt oder zur Weitergabe an einen Kollegen, können Sie CLOU anweisen, sich die Inhalte der Variablen zu merken.

3.3.1 Rechen- und Stringvariablen merken

Die aktuell gültigen Werte aller definierten Rechen- und Stringvariablen - einschließlich Makros und Funktionen - können über die folgende Anweisung "gemerkt" werden:

Syntax

#» "Dokumentname"

Die Variablen werden dabei in Form einer Definitionsanweisung in das angegebene HIT-Dokument *Dokumentname* geschrieben. Von dort können sie später mit der `#B`-Anweisung wieder eingelesen werden.

Beispiel:

```
#= i 13
#= s "hallo"
#» "allevar"
```

ergibt im Dokument `allevar`

```
.
.
.
#D i 13
#D s "hallo"
```

3.3.2 Liste als HIT-Dokument sichern

Listen-Variable werden mit obiger Anweisung nicht übernommen. Für Listen existiert eine Spezial-Anweisung, mit der der Inhalt jeder Liste in ein eigenes Dokument gesichert werden kann.

Syntax

#L ListName » "Dokumentname"

Die angegebene Liste *ListName* wird dabei zeilenweise als HIT-Dokument *Dokumentname* abgelegt. Das Wiedereinlesen von Listen-Werten erfolgt über die im Kap. 2.3.1 beschriebene Anweisung
#L ListName H *"Dokumentname"*.

4 Dateizugriffe

4.1 Zugriff auf HIT-Dateien

Mit der #B-Anweisung können Sie nur ganze HIT-Dokumente als normale Textbausteine einfügen. Sie haben aber keinen Einfluß mehr auf den Einfüge-Vorgang; z.B. können Sie nicht einzelne Teile aus einem Dokument auswählen. Auch ist es mit den bisher beschriebenen Mitteln nicht möglich, in andere Dokumente als das aktuelle Zieldokument selbst hineinzuschreiben.

Einen solchen kontrollierten Zugriff bieten die in diesem Kapitel beschriebenen #H-Anweisungen. Insbesondere können Sie damit ein HIT-Dokument zeilenweise in Stringvariablen einlesen, diese in beliebiger Weise weiterverarbeiten und das Ergebnis in Ihr Zieldokument einfügen.

Alle Anweisungen, die mit dem Zugriff auf HIT-Dateien zusammenhängen, haben den gleichen Aufbau:

#H ["*Dateiname*"] *Kommandoschlüssel* [*Parameter*]

Der *Kommandoschlüssel* besteht aus einem Buchstaben oder Sonderzeichen und besagt, was mit der Datei *Dateiname* geschehen soll. Die Angabe von *Dateiname* kann entfallen; CLOU verwendet dann den Dateinamen aus der zuletzt ausgeführten #H-Anweisung.

CLOU kennt von sich aus, d.h. ohne entsprechende Definitions-Anweisungen, mehrere sog. globale Variablen. Zwei davon spielen im Zusammenhang mit #H-Anweisungen eine Rolle:

- ERRNO

  Die Variable ERRNO wird von CLOU nach jeder #H-Anweisung auf 0 (Null) gesetzt, wenn die Anweisung erfolgreich ausgeführt wurde. Es können beim Zugriff auf Dateien aber diverse Fehler auftreten, z.B. wenn versucht wird, auf eine nicht existierende Datei zuzugreifen oder über das Ende einer Datei hinauszulesen. In solchen Fällen setzt CLOU die Variable ERRNO auf einen Wert ungleich 0. Sie können also im Anschluß an eine #H-Anweisung am Wert der Variablen ERRNO erkennen, ob ein Fehler aufgetreten ist oder nicht.

- OK

  Die Variable OK hat immer den Wert 0 (Null). Sie dient lediglich der besseren Lesbarkeit von CLOU-Bausteinen: Da der Wert 0 für die Variable ERRNO bedeutet, daß alles gutgegangen ist, kann man nach einer #H-Anweisung z.B. schreiben:

```
#? ERRNO = OK :
/N
    #M "Fehler!"
    #;
#
```

Der Einfügevorgang wird dann mit der Meldung Fehler! abgebrochen, wenn ERRNO nicht den Wert OK, also Null hat.

4.1.1 Öffnen einer HIT-Datei

Ähnlich wie Variable vor ihrer Verwendung definiert werden müssen, muß eine Datei "geöffnet" werden, bevor man in ihr lesen oder schreiben kann. Dadurch wird CLOU mit ihrem Namen bekannt gemacht und erfährt gleichzeitig, ob Sie die Datei lesen oder beschreiben wollen:

Syntax

#H ["*Dateiname*"] O ["*Modus*"]

Für *Dateiname* kann der Name eines beliebigen HIT-Dokuments stehen. "O" steht für öffnen (englisch open). Für *Modus* kann einer der beiden folgenden Werte stehen:

r wenn Sie die Datei lesen wollen. In diesem Fall muß die Datei *Dateiname* bereits existieren. Ist dies nicht der Fall, oder ist die angegebene Datei kein HIT-Dokument, so wird keine Datei geöffnet und die globale Variable ERRNO auf einen von Null verschiedenen Wert gesetzt. Dies kann auch geschehen, wenn Sie keine ausreichenden Zugriffsrechte auf die Datei haben.

w wenn Sie in die Datei schreiben wollen. In diesem Fall muß die angegebene Datei noch nicht existieren, sie wird dann neu angelegt. Ist bereits eine Datei *Dateiname* vorhanden, so geht ihr Inhalt verloren. Auch hier kann bei Verletzung der Zugriffsrechte ein Fehler auftreten.

In jedem Fall sollten Sie den Wert der globalen Variablen ERRNO unmittelbar nach dem Öffnen einer Datei abfragen, um sicherzugehen, daß alles gutgegangen ist.

Beispiel:

```
#

#* Öffnen der HIT-Datei "liste" zum Lesen:
#

#H "liste" 0 "r"

#? ERRNO = OK:
/N
    #M "Datei ""liste"" kann nicht geöffnet werden!"
    #;
#

#* Jetzt kann die Datei gelesen werden.
(Kapitel 4.1.3)

#
```

Prinzipiell können Sie in einem Baustein mehrere Dateien nacheinander öffnen und gleichzeitig bearbeiten - jede einzelne ist durch ihren Namen eindeutig identifiziert. Die Gesamtzahl der Dateien, die zur gleichen Zeit offen sein können, ist aber vom System her auf fünf beschränkt. Sie sollten daher nicht mehr benötigte Dateien unbedingt schließen, wie im nächsten Kapitel beschrieben.

4.1.2 Schließen einer HIT-Datei

Wann immer Sie eine zuvor geöffnete Datei nicht mehr benötigen, sollten Sie diese schließen. Das hat zwei Gründe:

- Die Gesamtzahl der Dateien, die gleichzeitig offen sein können, ist vom System her auf fünf beschränkt. Wenn Sie in einem Baustein mit mehreren Dateien arbeiten, dann kann diese Grenze überschritten werden. Dies führt zum Abbruch des Baustein-Einfügens mit einer entsprechenden Fehlermeldung.
- Bei Verwendung der #B-Anweisung kann auch noch ein anderes Problem auftreten. Angenommen, in zwei verschiedenen Bausteinen stehen Öffnungs-Anweisungen für die gleiche Datei. Wenn Sie nun beide Bausteine mit der #B-Anweisung von einem dritten Baustein aus einlesen, so führt die zweite Anweisung zum Abbruch des Einfügevorgangs mit einer entsprechenden Fehlermeldung: die gleiche Datei darf nicht zweimal geöffnet werden.

Aus diesen Gründen sollten Sie jede Datei, die Sie nicht mehr benötigen, sofort explizit schließen:

Syntax

#H ["*Dateiname*"] C

Für *Dateiname* muß der Name einer offenen Datei stehen. Fehlt diese Angabe, so wird der zuletzt in einer #H-Anweisung benutzte Dateiname verwendet. "C" steht für schließen (englisch close). Nach Abschluß eines Einfügevorgangs schließt CLOU alle noch offenen Dateien.

4.1.3 Zeilenweises Lesen

Anders als bei der #B-Anweisung, mit der Sie nur ganze HIT-Dateien einlesen können, gibt Ihnen die Lese-Anweisung die Möglichkeit, auf eine Datei zeilenweise zuzugreifen:

Syntax

#H ["*Dateiname*"] < *VarName*

Für *Dateiname* muß der Name einer zuvor zum Lesen geöffneten Datei stehen. Standardwert ist der Name der zuletzt verwendeten Datei. Für *VarName* muß der Name einer Stringvariablen stehen. Die Anweisung bewirkt, daß eine Zeile aus der Datei *Dateiname* in die Variable *VarName* eingelesen wird.

Die erste Lese-Anweisung nach dem Öffnen der Datei liest die erste Zeile ein, die nächste Lese-Anweisung die zweite usw. Wird versucht, über das Ende der Datei hinauszulesen, so wird die CLOU-Variable ERRNO auf einen Wert ungleich Null gesetzt; der Wert der Variablen *VarName* bleibt dabei unbeeinflußt.

Beispiel:

```
#
#* Die Datei "liste" wird ab der 5. Zeile in das
   Zieldokument eingefügt.
#
#* Öffnen der Datei zum Lesen:
#
#H "liste" O "r"

#? ERRNO = OK:
/N
    #M "Datei ""liste"" kann nicht geöffnet werden!"
    #;
#

#* Definition der Stringvariablen, in die eingelesen
   werden soll:
#
#D zeile ""

#* Überlesen der ersten 4 Zeilen:
#
#W 5
    #H "liste" < zeile
#
#* Die 5. Zeile steht jetzt in "zeile". Es folgt das
   Einfügen dieser und der restlichen Zeilen:
#
#S ERRNO = OK:
    #> zeile
    #^"RETURN"
    #H < zeile
    #* der Dateiname kann weggelassen werden #
#

#* Schließen nicht vergessen:
#
#H C
```

Zwischen dem Einlesen einer Zeile und ihrer Ausgabe ins Zieldokument kann diese natürlich noch verändert werden; z.B. kann man sie mit Hilfe von Teilstrings in einzelne Worte zerlegen und diese in Stringausdrücken neu zusammensetzen. Beispiele für solche Techniken finden Sie im Kapitel 4.1.5.

4.1.4 Zeilenweises Schreiben

Mit der Schreib-Anweisung können Sie Strings oder Stringvariablen in andere Dokumente als das aktuelle Zieldokument schreiben. Allerdings können Sie diese nicht nachträglich in ein vorhandenes Dokument einfügen. Wenn Sie eine bereits existierende Datei zum Schreiben öffnen, wird ihr alter Inhalt nämlich automatisch gelöscht. Die Schreib-Anweisung hat die Form:

Syntax

#H ["*Dateiname*"] > *VarName*

oder

#H ["*Dateiname*"] > "*String*"

Dateiname muß der Name einer zuvor zum Schreiben geöffneten Datei sein. Standardwert ist wieder der Name der zuletzt verwendeten Datei. Für *VarName* muß der Name einer Stringvariablen stehen. Der Wert der Variablen *VarName* wird in die Datei *Dateiname* ausgegeben. Wahlweise kann anstelle eines Variablenwerts auch ein *String* ausgegeben werden.

4.1.5 Beispiele

Umwandlung von Umlauten

Eine HIT-Datei soll auf einem Drucker ausgegeben werden, der keine Umlaute ('ä', 'ö', 'ü') darstellen kann. Zu diesem Zweck wird die Originaldatei mit dem Namen "brief" zeilenweise gelesen, die Umlaute durch 'ae', 'oe' bzw. 'ue' ersetzt und die Ergebnisse zeilenweise in eine neue Datei "brief.u" geschrieben:

```
#
#* Umwandlung von Umlauten
#

#D org "brief"              #* Originaldatei #
#D neu "brief.u"            #* neue Datei #
#D org_zeile ""             #* Puffer für Originalzeile
                            #
#D neu_zeile ""             #* Puffer für umgewandelte
                               Zeile #
#D zeichen ""               #* aktuelles Zeichen #
#D i 0                      #* Zeichenzähler #
#H $org 0 "r"
#? ERRNO = OK :
/N
    #M "Originaldatei 'brief' nicht gefunden!"
    #;
#

#H $neu 0 "w"
#? ERRNO = OK :
/N
    #M "Datei 'brief.u' kann nicht geöffnet werden!"
    #;
#

#* Jetzt sind beide Dateien offen.
#
```

```
#S ERRNO = OK :
    #H $org < org_zeile

    #? ERRNO = OK :
    /J

        #= neu_zeile ""

        #= i 1
        #S org_zeile[i] <> "":
            #= zeichen org_zeile[i]

            #C zeichen
            / "ä" :
                #= zeichen "ae"
            / "ö" :
                #= zeichen "oe"
            / "ü" :
                #= zeichen "ue"
            #

            #= neu_zeile  neu_zeile & zeichen
            #= i i+1
        #

        #H $neu > neu_zeile
    #
#

#* Schließen beider Dateien:
#

#H $org C
#H $neu C
```

4.2 Zugriff auf SINIX-Dateien und Pipes

Für das Verständnis der folgenden Kapitel werden gewisse Kenntnisse des Betriebssystems SINIX, in Spezialfällen auch der Programmiersprache C vorausgesetzt. Wenn Sie schon mit SINIX Shell-Kommandos wie z.B. `ls`, `cp` usw. gearbeitet haben, sollten Sie aber auf jeden Fall Kapitel 4.2.1 lesen. Dort erfahren Sie, wie man diese Kommandos von CLOU aus aufrufen kann.

Die weiteren Kapitel befassen sich mit der Bearbeitung von SINIX-ASCII-Dateien sowie dem Anschluß von Shell-Kommandos an CLOU über Pipes. Sollten Ihnen diese Begriffe nichts sagen, so können Sie diese Kapitel sowie den Rest dieser Einleitung übergehen.

Ähnlich wie die Anweisungen für den Zugriff auf HIT-Dateien in Kapitel 4.1 folgen auch die hier beschriebenen Anweisungen einem einheitlichen Aufbau:

#X ["Name"] Kommandoschlüssel [Parameter]

Das "X" steht für "SINIX". Für *Name* kann der Name einer Datei oder eines über Pipe angeschlossenen Shell-Kommandos (Kapitel 4.2.3!) eingesetzt werden. Standardwert ist wieder der zuletzt verwendete Name. Der *Kommandoschlüssel* besteht aus einem Buchstaben, der die gewünschte Funktion angibt: Öffnen, Schließen, Lesen etc.

Die globale Variable `ERRNO` dient wie beim Zugriff auf HIT-Dateien der Fehlererkennung. Ihr wird nach jeder `#X`-Anweisung ein Wert zugewiesen. Ist dieser gleich Null, so ist alles gutgegangen. Im Fehlerfall ist der Wert ungleich Null. Um die Lesbarkeit Ihrer CLOU-Bausteine zu verbessern, können Sie statt auf Null wieder auf die globale Variable `OK` abfragen, die stets den Wert Null hat.

4.2.1 SINIX-Shell-Kommandos aufrufen

Wenn SINIX für Sie Shell-Kommandos ausführen soll, dann sagen Sie ihm das so:

Syntax

#! "Kommando"

Das *Kommando* sieht so aus, wie Sie es bei direkter Benutzung der Shell auch schreiben würden. Es kann aus beliebig vielen Einzel-Kommandos bestehen. Wenn CLOU diese Anweisung erreicht, löst er das *Kommando* aus, wobei er die Standardausgabe des Shell-Kommandos in das Zieldokument einfügt.

Der Rückgabe-Code des Shell-Kommandos, der in der Regel 0 (Null) ist, wenn die Ausführung erfolgreich abgelaufen ist, wird hinterher der vordefinierten globalen Variablen `RC` zugewiesen. Die Variable `RC` kann wie üblich mit der Variablen `OK` verglichen werden, deren Wert immer 0 ist.

Hinweis:

- Da HIT und CLOU während der Ausführung des Shell-Kommandos weiterhin aktiv bleiben und Eingaben von Tastatur verarbeiten, dürfen keine interaktiven Kommandos aufgerufen werden, die ihrerseits Eingaben von der Tastatur erwarten!
- Ist das letzte Zeichen des Shell-Kommandos ein '&' (Hintergrund-Prozeß), dann wird die Ausgabe des Kommandos nicht eingefügt.
- Tritt bei der Ausführung des Shell-Kommandos ein Fehler auf, so übernimmt CLOU die Fehlermeldung des Kommandos und zeigt sie in der Hinweiszeile an.

Beispiele:

- Die Anweisung

```
#! "ls"
```

  fügt die Namen der Dateien im aktuellen Verzeichnis ins Zieldokument ein.

- Der folgende Baustein benutzt das Shell-Kommando `test`, um die Existenz einer vom Benutzer angegebenen Datei zu überprüfen:

```
#

#D name ""
#D cmd ""

#< name "Bitte Dateinamen angeben:"
#= cmd "test -f " & name

#! $cmd
#? RC = OK:
/N
    #M "die Datei existiert"
    #;
#
```

4.2.2 Öffnen einer Datei

Bevor Sie eine Datei lesen, schreiben oder verändern können, müssen Sie sie öffnen:

Syntax

#X ["*Dateiname*"] O ["*Modus*"]

Der Kommandoschlüssel "O" steht für "open". Als Modus können Sie angeben:

r	wenn Sie die Datei nur Lesen wollen (read);
w	wenn Sie die Datei neu schreiben wollen (write), der ursprüngliche Inhalt der Datei geht dabei verloren;
rw	wenn Sie die Datei lesen und verändern wollen, der ursprüngliche Inhalt der Datei bleibt dabei zunächst erhalten;
a	wenn Sie etwas an das Dateiende anfügen wollen (append).

Fehlt die Modus-Angabe, so wird die Datei zum Lesen eröffnet.

Es können maximal fünf Dateien gleichzeitig offen sein. Bedenken Sie aber - besonders beim Arbeiten mit verschachtelten Textbausteinen - daß die Gesamtzahl offener Dateien vom System her begrenzt ist.

Beispiel:

```
#
#* Öffnen einer Datei mit dem Namen "hallo"
   zum Schreiben
#

#D datei "hallo"
#D meldung ""

#X $datei O "w"

#? ERRNO = OK:
/N
    #= meldung datei & " kann nicht geöffnet werden"
    #M $meldung
    #;
#
```

4.2.3 Anschluß eines SINIX-Kommandos über Pipes

Alle Lese- und Schreiboperationen, die mit Dateien durchgeführt werden können, sind auch mit Pipes möglich. Sie können ein beliebiges SINIX-Programm über eine oder zwei Pipes an CLOU anschließen:

Syntax

#X ["*Shellkommando*"] ! ["*Modus*"]

Mit dieser Anweisung wird der durch *Shellkommando* bezeichnete Prozeß gestartet und in Abhängigkeit von *Modus* über Pipes an CLOU angeschlossen. *Modus* steht für einen der Werte:

r	Eine Pipe vom Prozeß zu CLOU wird angelegt. Die Standard-Ausgabe des Prozesses kann nun von CLOU gelesen werden.
w	Eine Pipe von CLOU zum Prozeß wird angelegt. CLOU kann nun direkt auf den Standard-Eingabekanal des Prozesses schreiben.
rw	Es werden zwei Pipes - in jede Richtung eine - angelegt.

Im *Shellkommando* können auch Aufrufparameter enthalten sein. Kann der Prozeß nicht gestartet werden, oder sind schon zu viele Dateien bzw. Pipes offen, so wird die CLOU-Variable `ERRNO` auf einen entsprechenden Wert gesetzt. Die Gesamtzahl der gleichzeitig offenen Dateien oder Pipes darf die Zahl 5 nicht überschreiten.

Beispiel:

```
#
#* Das SINIX-Kommando "ls" soll mit der l-Option auf-
   gerufen und seine Ausgabe von CLOU gelesen werden:
#
#D kommando "ls -l"

#X $kommando !
#? ERRNO = OK:
/N #M "Pipe kann nicht geöffnet werden"
   #;
#
#* Hier folgt die eigentliche Leseanweisung (wie in den
   folgenden Kapiteln beschrieben)
```

Für alle weiteren Operationen mit einer einmal geöffneten Pipe muß das *Shellkommando* Buchstabe für Buchstabe so wie beim Aufruf des Prozesses angegeben werden - einschließlich aller Parameter und Leerzeichen. Sie sollten daher - um Schreibfehler auszuschließen - für jedes Kommando eine Stringvariable verwenden, die Sie dann in allen weiteren Anweisungen wieder einsetzen können. In allen Beispielen dieses Kapitels wird diese Technik verwendet, wenn der Name nicht weggelassen werden kann. Standardwert für *Shellkommando* ist der zuletzt verwendete Name.

4.2.4 Schließen einer Datei oder Pipe

Wenn Sie eine Datei oder eine Pipe nicht mehr benötigen, so sollten Sie sie explizit schließen:

Syntax

#X ["*Name*"] C

Das "C" steht für "close". Für *Name* muß bei Dateien der Dateiname, bei Pipes das Shellkommando eingesetzt werden. War das Shell-Kommando mit zwei Pipes aufgerufen worden, so werden beide geschlossen. CLOU schließt automatisch alle noch offenen Dateien, wenn er beendet wird. Trotzdem sollten Sie in der Regel am Ende jedes CLOU-Bausteins alle dort geöffneten Dateien explizit schließen, da Sie sonst beim Arbeiten mit verschachtelten Textbausteinen schnell an die Grenze von maximal fünf offenen Dateien stoßen können.

4.2.5 Lesen eines Variablenwertes

Wenn Sie eine Datei oder Pipe zum Lesen eröffnet haben, können Sie mit der Leseanweisung einzelne Worte, Buchstaben oder Zahlen einlesen. Für zeilenweises Lesen steht eine eigene Anweisung zur Verfügung (Kapitel 4.2.7)

Syntax

#X ["*Name*"] R *VarName*

Der Kommandoschlüssel R steht für "read"; für *Name* muß der Name einer offenen Datei oder eines über Pipe angeschlossenen Shell-Kommandos eingesetzt werden. Steht *VarName* für eine Stringvariable, so wird diese in ihrem bei der Definition angegebenen Format eingelesen; bei Rechenvariablen wird generell das Standardformat verwendet.

Wollen Sie z.B. eine Datei Zeichen für Zeichen lesen, so können Sie eine Stringvariable `Zeichen` definieren, die das Format "1" hat und somit immer genau ein Zeichen aufnehmen kann:

```
#D Zeichen "" "1"
```

Programmierern, die Erfahrungen mit der Sprache C haben, ist die Bedeutung der Formate beim Lesen von der Funktion `fscanf` her bekannt.

Nach einem erfolgreichen Lesevorgang wird die Variable `ERRNO` auf 0 gesetzt. Konnte nicht gelesen werden, weil z.B. das Ende der Datei bereits erreicht war, so erhält `ERRNO` den Wert -1.

Beispiel:

```
#
#* Wortweises Lesen aus einer bereits geöffneten
   Datei:
#

#D wort ""


#* Jede Stringvariable hat automatisch das Format "Wort"
#

#S ERRNO = OK :
    #X R wort
    #? ERRNO = OK :
    /J
        #* ausgeben in das Zieldokument #
        #> wort #^ "RETURN"
    /N
        #M "Ende der Datei erreicht"
    #
#
#X C
```

4.2.6 Schreiben eines Variablenwertes

Wenn Sie eine Datei oder Pipe zum Schreiben eröffnet haben, können Sie mit der Schreibanweisung Variablenwerte auf sie ausgeben.

Syntax

#X ["Name"] W VarName

oder

#X ["Name"] W "Text"

Der Kommandoschlüssel W steht für "write"; für *Name* muß der Name einer offenen Datei oder eines über Pipe angeschlossenen Shell-Kommandos eingesetzt werden. In der ersten Form wird der Wert der Variablen *VarName* in ihrem eigenen Format ausgegeben. Im zweiten Fall wird der zwischen den Anführungszeichen stehende Text ohne weitere Formatierung ausgegeben.

Wird auf eine Datei geschrieben, so erfolgt die Ausgabe gepuffert, d.h. die Daten stehen erst dann mit Sicherheit in der Datei, wenn diese wieder geschlossen wird. Bei Pipes hingegen wird der Ausgabepuffer von CLOU automatisch nach jeder Schreibanweisung geleert, also tatsächlich auf die Pipe geschrieben. Dadurch ist gewährleistet, daß der angeschlossene Prozeß die Daten sofort erhält, was die Gefahr eines sog. "Deadlock" vermeidet: Dieser kann eintreten, wenn zwei über Pipes kommunizierende Prozesse beide auf Daten vom jeweils anderen warten, weil diese noch im Puffer stehen.

Beispiel:

```
#
#D dname "beispiel"
#D preis 3 ".2"

#X $dname O "w"
#X $dname W "Preis: "
#X $dname W preis
#X $dname C

#* in der Datei "beispiel" steht jetzt 'Preis: 3.00'#
```

Hinweis:

- Ein über Pipes angeschossenes C-Programm muß ebenfalls sicherstellen, daß der Ausgabepuffer nach jeder Ausgabe, die CLOU lesen soll, automatisch geleert wird. Dies kann durch den Aufruf von `fflush` (stdout) erreicht werden.
- Stringkonstanten können auch (dreistellig) hexadezimal ausgegeben werden.

  Beispiel:

  ```
  #X W "\00a"
  ```
  erzeugt eine neue Zeile

4.2.7 Lesen einer Zeile

Neben zeichen- oder wortweisem Lesen ist auch das Einlesen einer ganzen Zeile von einer Datei oder Pipe möglich:

Syntax

#X ["*Name*"] < *VarName*

Für *Name* muß wie üblich der Name einer Datei oder eines Shell-Kommandos stehen. *VarName* ist der Name der Stringvariablen, in die die Zeile aus der Datei übernommen werden soll. Ist die Länge der Zeile größer als 300 Zeichen, so wird der Rest abgeschnitten. Wird versucht, über das Ende der Datei hinauszulesen, so wird wie beim Einlesen eines Variablenwertes die CLOU-Variable ERRNO auf -1 gesetzt.

Beispiel:

```
#
#* Der Inhalt der Datei "beispiel" wird zeilenweise in das
   Zieldokument eingefügt.
#
#D dname "beispiel"
#D zeile ""
#X $dname O "r"
#? ERRNO <> OK:
/J
    #M "Datei kann nicht geöffnet werden"
    #;
#
#S ERRNO = OK:
    #X $dname < zeile
    #? ERRNO = OK:
    /J
       #> zeile
       #^ "RETURN"
    #
#
#X $dname C
```

4.2.8 Schreiben einer Zeile

In einer mittels #X-Anweisung geöffneten Ausgabedatei können Sie mit den bisher beschriebenen Schreibanweisungen keine Zeilenvorschübe (Zeilenende-Markierungen; "newline-characters") erzeugen. Dazu benötigen Sie diese Anweisung:

Syntax

#X ["*Name*"] > *VarName*

oder

#X ["*Name*"] > "*Text*"

Der einzige Unterschied zu der im Kapitel 4.2.6 "Schreiben eines Variablenwertes" beschriebenen Anweisung besteht darin, daß nach der Ausgabe des Variablenwertes oder Textes ein Zeilenvorschub geschrieben wird. Wollen Sie z.B. nur eine Leerzeile erzeugen, so können Sie schreiben:

```
#X $dname > ""
```

4.2.9 Dateizeiger abfragen und positionieren

Für jede offene SINIX-Datei stellt das Betriebssystem einen Datei-Zeiger zur Verfügung, der die aktuelle Position beim Lesen oder Schreiben in der jeweiligen Datei angibt. Diese Position ist bestimmt durch die Anzahl der Zeichen, die sich zwischen der aktuellen Position und dem Datei-Anfang befinden.

Unmittelbar nach dem Öffnen einer Datei steht dieser Zeiger vor dem ersten Zeichen, also auf 0. Bei jedem Lesen (bzw. Schreiben) wird er dann um die Anzahl der gelesenen (bzw. geschriebenen) Zeichen hochgezählt.

Soll nicht sequentiell gelesen oder geschrieben werden, so kann der Zeiger vor dem Lesen oder Schreiben explizit auf einen bestimmten Wert gesetzt werden:

Syntax

#X ["*Dateiname*"] S *Distanz*

Für *Distanz* kann eine Zahl oder der Name einer Rechenvariablen eingesetzt werden. Der Dateizeiger wird um die angegebene *Distanz* verschoben.

Mit der vordefinierten CLOU-Variablen BASE können Sie festlegen, von welcher Position aus der Zeiger verschoben werden soll. Dazu weisen Sie der Variablen einen der folgenden Werte zu:

0	Die *Distanz* wird vom Datei-Anfang an gerechnet.
1	Der Zeiger wird um die angegebene *Distanz* von der aktuellen Position aus verschoben.
2	Die *Distanz* wird vom Datei-Ende an gerechnet.

Die globale Variable BASE ist mit dem Wert 0 vorbelegt.

Sie können den aktuellen Stand des Dateizeigers auch abfragen. Dies geschieht mit folgender Anweisung:

Syntax

#X ["*Dateiname*"] T *VarName*

Für *VarName* muß der Name einer Rechenvariablen stehen. Dieser weist CLOU die aktuelle Position des Dateizeigers zu.

Beispiel:

```
#* Zurücksetzen des Dateizeigers an den Datei-Anfang:
#
#= BASE 0
#X S 0

#* Zurücksetzen des Zeigers um 10 Zeichen:
#
#= BASE 1
#X S -10

#* Positionieren ans Ende der Datei:
#
#= BASE 2
#X S 0
```

4.2.10 "Eingebaute" SINIX-Kommandos

Mit der CLOU-Anweisung `#!` können Sie - wie beschrieben - beliebige SINIX Shell-Kommandos aufrufen. Zusätzlich sind aber einige häufig benötigte Kommandos in CLOU "eingebaut" worden. Diese können Sie direkt über CLOU-Anweisungen ansprechen; sie laufen dann erheblich schneller ab als im Falle eines Shell-Aufrufs. Den Erfolg eines Aufrufs können Sie wieder anhand des Wertes von `ERRNO` überprüfen. Falls kein Fehler aufgetreten ist, ist dieser gleich 0.

Löschen von Dateien

Syntax

#X ["*Name*"] U

Die Datei *Name* wird gelöscht. Der Kommandoschlüssel U steht für unlink.

Beispiel:

Anstelle des Shell-Aufrufs

```
#! "rm temp"
```

können Sie also schreiben:

```
#X "temp" U
```

Modus einstellen

Syntax

#X ["*Name*"] M "*Modus*"

Die Datei *Name* bekommt den Zugriffsschlüssel *Modus* zugewiesen. *Modus* muß oktal als String angegeben werden.

Beispiel:

Anstelle des Shell-Aufrufs

```
#! "chmod 777 temp"
```

können Sie also schreiben:

```
#X "temp" M "777"
```

Link einrichten

Syntax

#X ["*Name*"] L "*NeuName*"

Auf die Datei *Name* wird ein Link eingerichtet, d.h. auf die Datei kann fortan auch unter dem Namen *NeuName* zugegriffen werden.

Beispiel:

Die gleiche Wirkung wie der Shell-Aufruf:

```
#! "ln temp neu"
```

hat also die CLOU-Anweisung:

```
#X "temp" L "neu"
```

Dateiverzeichnis wechseln

Syntax

#X ["*Dateiverzeichnis*"] D

Das aktuelle Dateiverzeichnis wird *Dateiverzeichnis*.

Beispiel:

Wenn Sie schreiben

```
#X "/usr/tmp" D
```

so werden alle weiteren Dateien, die Sie mit der `#X`- oder `#H`-Anweisung öffnen, unter `/usr/tmp` gesucht.

Hinweis:

Der Shell-Aufruf:

```
#! "cd /usr/tmp"
```

hat nicht die gleiche Wirkung! Der Wechsel in das Dateiverzeichnis `/usr/tmp` gilt in diesem Fall nur für die mit dem `#!`-Kommando eröffnete Sub-Shell; nachfolgende `#X`-Anweisungen sind davon nicht betroffen.

4.2.11 Beispiele

Datei kopieren

Eine Datei wird zeilenweise eingelesen und in eine andere Datei ausgegeben (geht natürlich viel einfacher mit dem SINIX-Kommando "cp").

```
#
#* Kopieren einer Datei #
#D buf ""          #* Zeilenpuffer                        #
#D from ""         #* Quelldatei                          #
#D to ""           #* Zieldatei                           #
#D msg ""          #* Hilfsvariable für Fehlermeldung     #
#< from "Kopieren von:"
#< to "nach:"
#X $from O "r"     #* Datei $from zum Lesen öffnen        #
#? ERRNO <> OK:
/J
    #= msg "Datei "&from&" kann nicht eröffnet
            werden, Fehlernummer: "&ERRNO
    #M $msg
    #;
#
#X $to O "w"       #* Datei $to zum Schreiben eröffnen #
#? ERRNO <> OK:
/J
    #= msg "Datei "&to&" kann nicht eröffnet
            werden."
    #M $msg
    #;
#
#S ERRNO = OK:     #* zeilenweise lesen und schreiben  #
    #X $from < buf
    #? ERRNO = OK:
    /J
        #X $to > buf
    #
#
#X $from C         #* Dateien ordnungsgemäß schließen  #
#X $to C
```

Liste von Namen einlesen und sortiert ausgeben

Eine Liste von Schulkinder-Namen wird in einem Dokument im Dialog eingegeben und gleichzeitig in eine Hilfsdatei geschrieben. Diese Datei wird anschließend durch das UNIX-Kommando "sort" sortiert. Dazu wird "sort" über eine Pipe angeschlossen. Anschließend wird die Hilfsdatei mit dem UNIX-Kommando "rm" wieder gelöscht.

```
#
#* Sortieren #
#K

#D buf " "                    #* Puffer für Namen      #
#D i 0                        #* Zählvariable          #
#D file "x"                   #* Name der Hilfsdatei   #
#D cmd "sort " & file         #* Kommando              #
#D msg ""                     #* für Fehlermeldung     #

#D ALLES_OK "#? ERRNO <> OK:
             /J
                 #= msg ""Abbruch Baustein Einfügen,
                         Fehlernummer: "" & ERRNO
                 #M $msg
                 #;
             #"      #* Makro zum Überprüfen von ERRNO #

#X $file O "w"                 #* Hilfdatei eröffnen   #
#$ ALLES_OK

#S buf <> "":
    #< buf "Vorname des Schulkinds (RETURN bricht ab):"

    #? buf <> "":
    /J
        #X > buf             #* in Hilfsdatei schreiben #
        #> buf               #* in den Einfügetext      #
        #^ "RETURN"
        #= i i+1
    #
#

#X C                         #* Schließen der Hilfsdatei #

#= i i+1
```

```
#^i"Z_LÖSCHEN"
                    #* Unsortierte Liste wieder löschen #

#X $cmd !                       #* sort über Pipe anhängen #
#$ ALLES_OK

#S ERRNO = OK:
    #X < buf                    #* zeilenweise die
                                   Ausgabe des sort-Kommandos
                                   lesen und in den Einfüge-
                                   text schreiben          #
    #? ERRNO = OK:
    /J
        #> buf
        #^"RETURN"
    #
#

#X C                            #* Pipe schließen        #
#=cmd "rm "&file
#! $cmd                         #* Hilfsdatei löschen    #
```

4.3 Datenbank-Zugriffe

Sie können von einem CLOU-Baustein aus auch auf Datenbanken zugreifen. Unterstützt werden die INFORMIX Versionen 2.1 und 4.0.

Datenbank-Anweisungen in CLOU-Bausteinen haben alle den gleichen Aufbau:

Syntax

#Q "Datenbank-Anweisung"

Das Q steht für "query" (engl. für Befragung, Abfrage). Die eigentliche Datenbank-Anweisung ist grundsätzlich nicht nach der CLOU-Syntax aufgebaut, sondern nach der des zugrunde liegenden Datenbank-Systems. CLOU-spezifisch sind lediglich folgende Punkte:

- Die ganze Datenbank-Anweisung steht in Anführungszeichen ("). Sollen diese selbst in der Anweisung vorkommen, müssen sie wie üblich doppelt geschrieben bzw. durch das Fluchtsymbol "\" gekennzeichnet werden.
- CLOU-Variable in Datenbank-Anweisungen werden durch ein vorangestelltes $-Zeichen kenntlich gemacht.

Voraussetzung für das Verständnis der einzelnen Anweisungen sind Kenntnisse des jeweiligen Datenbanksystems. Wie bereits bemerkt, hört der Gültigkeitsbereich der CLOU-Syntax nach dem `#Q` auf: alles, was dahinter zwischen den Anführungszeichen steht, muß der Syntax des jeweiligen Datenbank-Systems gehorchen. In den folgenden Abschnitten dieses Kapitels wird daher - soweit nicht CLOU-spezifische Details eine Rolle spielen - jeweils auf die einschlägige Literatur verwiesen.

INFORMIX unterstützt die Abfragesprache SQL (für "Structured Query Language"). Der große Sprachumfang von SQL würde den Rahmen dieser Dokumentation sprengen. Wir verweisen daher auf die im Anhang aufgeführten Werke /1/ und /3/.
/3/ dient als Referenz-Handbuch, während /1/ eine ausführliche Einführung in relationale Datenbanksysteme und speziell in SQL enthält.

Alle SQL-Anweisungen können in `#Q`-Anweisungen verwendet werden. Parameter für das Eintragen, Ändern oder Löschen von Datensätzen können aus CLOU-Variablen übernommen werden; umgekehrt werden bei Suchanfragen Daten aus der Datenbank in CLOU-Variablen geschrieben.

Der Datenbank-Anschluß wurde über die C-Schnittstelle von INFORMIX realisiert (ESQL/C). Dadurch ist sowohl lesender als auch schreibender Zugriff auf Datenbanken möglich. Die Verarbeitung erfolgt über Zeiger.

Fehler werden dem Benutzer durch Angabe der Fehlerstelle und einer Fehlernummer gemeldet. Diese Fehlernummern sind identisch mit denen der C-Schnittstelle und können in /3/ nachgeschlagen werden.

Die Verwendung von CLOU-Variablen innerhalb dieser Anweisungen entnehmen Sie bitte den ausführlichen Beispielen am Ende dieses Kapitels. Im übrigen folgen Sie genau der SQL-Syntax.

Einschränkungen:

- Die C-Programmierern bekannten ESQL/C-Anweisungen `PREPARE`, `EXECUTE` und `DESCRIBE` können Sie nicht verwenden. Sie werden auch nicht benötigt, da der CLOU-Benutzer mit Hilfe der `#F`-Anweisung ohnehin die Möglichkeit hat, dynamisch Datenbank-Anweisungen einzugeben.
- Ebenfalls nicht unterstützt werden Scroll- und `INSERT`-Satzanzeiger mit den zugehörigen Operationen `OPEN`, `PUT`, `FLUSH` und `CLOSE`. Sie sind in der X/OPEN-Definition von SQL nicht enthalten. Dies bedeutet allerdings keine Einschränkung der Funktionalität, da Datensätze auch über die normale `INSERT`-Anweisung eingefügt werden können.
- ESQL/C kennt bei der Deklaration eines Cursors für die `SELECT`-Anfrage die Option `FOR UPDATE`, die es erlaubt, sich in nachfolgenden `DELETE`- oder `UPDATE`-Anweisungen auf den aktuellen Datensatz zu beziehen. Diese Möglichkeit wird von CLOU nicht unterstützt.

4.3.1 Voraussetzungen für Datenbank-Zugriffe

Damit CLOU auf Datenbanken zugreifen kann, müssen folgende Voraussetzungen unbedingt erfüllt sein:

- Für Informix muß der erforderliche Datenbank-Server installiert sein.
- Die Shell-Variable `DBPATH` muß mit dem Dateiverzeichnis besetzt werden, in dem sich die benötigten Datenbank-Dateien befinden. Wie bei der Shellvariablen `PATH` können auch mehrere Dateiverzeichnisse angegeben werden; diese müssen dann durch : getrennt sein. Fehlt die Angabe von `DBPATH`, so wird im aktuellen Dateiverzeichnis gesucht.

Weitere Shell-Variablen, mit denen Sie das Verhalten von INFORMIX beeinflussen können, finden Sie in /3/.

4.3.2 Datenbank-Abfragen und Zeiger

Datenbank-Abfragen folgen einem bestimmten Schema, das in diesem Kapitel erläutert wird. Auf die einzelnen hier erwähnten Anweisungen wird dann in den folgenden Kapiteln genauer eingegangen.

Datenbank-Abfragen von einem CLOU-Baustein aus haben letztlich zum Ziel, daß die Daten eines Satzes entsprechenden CLOU-Variablen als Werte zugewiesen werden. Dabei stellt sich das Problem, daß im allgemeinen mit einer Suchanfrage mehrere, oft sehr viele Datensätze gefunden werden. Die Datensätze sollen dann einzeln, der Reihe nach auf die CLOU-Variablen abgebildet werden.

Grundlegend für das Arbeiten mit Datenbanken ist daher der Begriff des **Zeigers**. Jede Suchanfrage an das Datenbanksystem wird an einen Zeiger gekoppelt. Mit Hilfe einer speziellen Anweisung kann dann Datensatz für Datensatz einzeln gelesen werden: der Zeiger rückt dabei jedesmal zum nächsten Satz vor.

Zeiger haben folgende Eigenschaften:

- Jeder Zeiger besitzt einen Namen, den Sie frei wählen können. Er muß der üblichen Syntax für CLOU-Variablennamen entsprechen, darf aber höchstens aus 30 Zeichen bestehen. Dieser Name wird in der DECLARE-Anweisung mit einer Suchanfrage verknüpft: für INFORMIX ist dies die SELECT-Anweisung (wie unten näher erläutert). Der so definierte Name wird in allen weiteren Datenbank-Anweisungen benutzt. Es können maximal fünf Zeiger definiert werden.

- Die DECLARE-Anweisung löst den eigentlichen Suchvorgang noch nicht aus. Dies geschieht erst, wenn der Zeiger mit der OPEN-Anweisung "geöffnet" wird. Nach dieser Anweisung "zeigt" der Zeiger auf den ersten gefundenen Datensatz.

- Um die Daten des ersten Datensatzes in CLOU-Variablen zu übernehmen, geben Sie die FETCH-Anweisung. Diese bewirkt zweierlei:

  - Die Daten des ersten Satzes werden auf die in der FETCH-Anweisung angegebenen CLOU-Variablen abgebildet.

  - Der Zeiger rückt auf den nächsten Datensatz vor, so daß die nächste FETCH-Anweisung die Daten des nächsten Satzes an die CLOU-Variablen zuweist.

  Durch aufeinanderfolgende FETCH-Anweisungen lesen Sie also die verschiedenen Datensätze der Reihe nach in die CLOU-Variablen ein. Üblicherweise geschieht dies in Form einer Schleife (#S-Anweisung). Die Abbruch-Bedingung für diese Anweisung liefert der Zeiger selbst: Durch die DECLARE-Anweisung wird mit jedem Zeiger implizit eine CLOU-Variable gleichen Namens definiert. Dieser weist CLOU nach jeder FETCH-Anweisung einen der beiden folgenden Werte zu:

  0 Alles in Ordnung: die CLOU-Variablen haben die neuen Werte.

  100 Kein Datensatz mehr vorhanden.

- Nicht mehr benötigte Zeiger sollten mit der CLOSE-Anweisung explizit geschlossen werden.

Näheres zu Zeigern können Sie /1/ und /3/ entnehmen.

Hinweis:

- Bei Beendigung eines Einfügevorgangs, z.B. durch Abbruch mit der DEL-Taste, wird eine u.U. noch offene Transaktion mit **rollback work** zurückgesetzt. D.h. alle Änderungen, die diese Transaktion auf der Datenbank bewirkt hat, werden zurückgesetzt.
- Beim Starten eines Korrekturlaufs (Hintergrundverarbeitung) wird eine ggf. noch offene Transaktion mit **rollback work** zurückgesetzt. Damit läßt sich der Korrekturlauf auch bei Änderungen der Datenbank durch den CLOU-Baustein sinnvoll einsetzen.

"Satztechnisch bedingte Leerseite"

4.3.3 Anweisungen

4.3.3.1 Die DATABASE-Anweisung

Mit der DATABASE-Anweisung wählen Sie eine Datenbank zur Bearbeitung aus. Alle nachfolgenden #Q-Anweisungen beziehen sich auf diese Datenbank, bis Sie mit einer weiteren DATABASE-Anweisung eine andere auswälen:

Syntax:

#Q "DATABASE *Datenbank-Name*"

4.3.3.2 Die DECLARE-Anweisung

Beim Abarbeiten einer Datenbank-Abfrage werden im allgemeinen mehrere Datensätze gefunden. Jeweils ein Datensatz wird in CLOU-Variablen zwischengespeichert. Um mehrere Datensätze der Reihe nach weiterverarbeiten zu können, benutzen Sie Zeiger. Diese werden mit Hilfe der DECLARE-Anweisung an eine Suchanfrage gekoppelt:

Syntax:

#Q " DECLARE *Zeigername* CURSOR FOR
SELECT..."

Die drei Punkte stehen für die SELECT-Anweisung, die die eigentliche Suchanfrage enhält. Die Syntax für die SELECT-Anweisung ist in /3/ nachzulesen.

Die DECLARE-Anweisung zeigt erst Wirkung, wenn mit der OPEN-Anweisung (siehe unten) der Suchvorgang ausgelöst wird. Eventuelle Fehlermeldungen erfolgen in der Regel ebenfalls erst zu diesem Zeitpunkt.

In der SELECT-Anweisung können beliebig viele CLOU-Variablen, gekennzeichnet durch ein vorangestelltes $-Zeichen, enthalten sein. Diese werden vor dem Öffnen des Zeigers durch ihre aktuellen Werte ersetzt.

4.3.3.3 Die OPEN-Anweisung

Mit der OPEN-Anweisung wird der in der DECLARE-Anweisung angegebene Suchvorgang ausgelöst:

Syntax:

#Q "OPEN *Zeigername*"

Die Ausführung dieser Anweisung benötigt im allgemeinen einige Zeit - je nachdem, wieviele Datensätze gefunden werden. Danach steht der Zeiger vor dem ersten Datensatz. Die Daten können dann in nachfolgenden FETCH-Anweisungen in CLOU-Variablen übernommen werden, was unabhängig von der Anzahl der gefundenen Datensätze recht schnell geht.

4.3.3.4 Die FETCH-Anweisung

Mit der FETCH-Anweisung übernehmen Sie die Daten eines Satzes in CLOU-Variablen, die Sie zu diesem Zweck vorher definieren müssen. Optional ist die Angabe von Indikatorvariablen möglich. Diese müssen ebenfalls vorher als CLOU-Variablen definiert werden.

Syntax:

#Q "FETCH *Zeigername*
INTO $*VarName1*[:*IndName1*], $*VarName2*[:*IndName2*], ...,
$*VarNameX*"[:*IndNameX*]"

Die Anzahl der CLOU-Variablen *VarName1* bis *VarNameX* muß mit der Anzahl der in der DECLARE-Anweisung angegebenen Felder genau übereinstimmen. Die Datentypen von INFORMIX werden soweit möglich den Typen der CLOU-Variablen angepaßt. Wie bei der automatischen Typkonvertierung (Kapitel 2.2.6) ist die Umwandlung nur dann unmöglich, wenn eine Zeichenkette, die nicht mit einer Zahl beginnt, einer Rechenvariablen zugewiesen werden soll.

Falls die Ausführung der FETCH-Anweisung erfolgreich abgelaufen ist, wird in die Indikatorvariablen einer der folgenden Werte eingetragen:

kleiner 0 NULL-Value wurde selektiert

0 Wert wurde selektiert

Bei der Selektion eines NULL-Values wird der Wert der Variablen auf 0 (bei Rechenvariablen) bzw. auf "" (bei Stringvariablen) gesetzt.

Mit jedem Zeiger definiert CLOU auftomatisch eine Rechenvariable gleichen Namens, die nach jedem FETCH-Aufruf mit einem der folgenden Werte besetzt wird:

0 Aufruf erfolgreich

100 Kein Datensatz mehr vorhanden

Wie Sie diesen Wert abfragen können, um in einer #S-Anweisung der Reihe nach alle Datensätze zu verarbeiten, ersehen Sie aus folgendem Bausteinschema.

"Satztechnisch bedingte Leerseite"

4.3.3.5 Bausteinschema für Datenbank-Abfragen

```
#D Name1
   .
   .
   .

#D NameX

#Q "DECLARE z  CURSOR FOR
    SELECT feld1, ..., feldX
    ..."

#* CLOU kennt nun einen Zeiger mit dem Namen "z".
   Jetzt kann der Suchvorgang ausgelöst werden:
#
#Q "OPEN z"
#* Der erste Datensatz wird eingelesen:
#
#Q "FETCH z
    INTO $Name1, ..., $NameX"
#* In einer Schleife werden die Datensätze der Reihe nach
   eingelesen und verarbeitet:
#
#S z = OK :
    #* Die CLOU-Variablen Name1 bis NameX enthalten Werte
       aus der Datenbank. Diese können hier weiterver-
       arbeitet - z.B. ausgegeben - werden.
    #
    #* Am Ende des Schleifenkörpers steht erneut die FETCH-
       Anweisung, mit der der jeweils nächste Datensatz ge-
       lesen wird. Dies wiederholt sich so lange, bis kein
       Datensatz mehr vorhanden ist: die Zeigervariable "z"
       bekommt dann den Wert 100, und die Schleife wird ab-
       gebrochen.
    #
    #Q "FETCH z
        INTO $Name1, ..., $NameX"
#
#Q "CLOSE z"
```

4.3.3.6 Die CLOSE-Anweisung

Wird ein Zeiger nicht mehr benötigt, oder soll er für eine Suchanfrage mit geänderten Parametern erneut deklariert werden, so muß er mit der CLOSE-Anweisung geschlossen werden:

Syntax

#Q "CLOSE *Zeigername*"

Daten, die noch nicht mit einer FETCH-Anweisung geholt worden sind, werden dabei gelöscht, um den Zeiger für eine neue Suchanfrage freizumachen. Am Ende des Einfügevorgangs schließt CLOU automatisch alle noch offenen Zeiger. Dies gilt auch, wenn der Einfügevorgang z.B. mit der DEL -Taste abgebrochen wird.

4.3.3.7 Verarbeitung von INFORMIX-Fehlermeldungen

Standardmäßig wird beim Erkennen von Fehlern in einer Datenbank-Anweisung abgebrochen und die entsprechende INFORMIX-Fehlernummer in der Meldezeile ausgegeben. Soll ein Datenbank-Fehler nicht unmittelbar zum Abbruch führen, sondern im CLOU-Baustein weiterverarbeitet werden, so kann der Abbruch-Modus durch folgende Anweisung ausgeschaltet werden:

Syntax

#Q "ignore"

Nach dieser Anweisung wird bei fehlerhaften SQL-Anweisungen nicht abgebrochen; stattdessen wird die entsprechende Fehlernummer in der globalen CLOU-Variable SQLCODE hinterlegt. Ist kein Fehler aufgetreten, enthält sie den Wert 0. Die Variable SQLCODE können Sie dann in Ihrem CLOU-Baustein auswerten.

Verwenden Sie die Anweisung `#Q "ignore"` nur, wenn Sie sichergestellt haben, daß alle Datenbank-Fehler in Ihrem CLOU-Baustein durch entsprechende Abfragen berücksichtigt werden!

Das Rücksetzen in den Abbruch-Modus erfolgt durch die Anweisung:

#Q "noignore"

4.3.4 Anschluß an INFORMIX 4.0

INFORMIX 4.0 kennt einige zusätzliche Feldtypen.
Diese werden mit CLOU folgendermaßen verarbeitet.

4.3.4.1 Feldtyp VARCHAR

Die Verarbeitung erfolgt analog zu CHAR(n). CLOU-String-Variablen besitzen ohnehin keine Längenbeschränkung.

4.3.4.2 Feldtypen DATETIME und INTERVAL

Beim Einfügen des Inhalts einer CLOU-Variablen in ein entsprechendes Feld mittels `INSERT/UPDATE ... $CLOUVARIABLE ...` muß die CLOU-Variable den Wert in Form einer Zeichenkette enthalten. Die alternativen Notationen (Angabe der Anfangs- und Endkomponenten bzw. von UNIT´s) werden nicht unterstützt. Beim Selektieren wird der Wert entsprechend formatiert in einer Stringvariablen abgelegt.

4.3.4.3 Feldtypen BYTE und TEXT

Diese BLOB-Typen ermöglichen es, beliebig lange Daten (z.B. ganze UNIX-Dateien) in Datenbankfeldern abzulegen. Die Ablage erfolgt binär (BYTE) oder als ASCII-String (TEXT). Von INFORMIX-SQL werden beide Datentypen nur sehr stiefmütterlich unterstützt, lediglich die Komponenten 4GL, PERFORM und natürlich ESQL/C bieten brauchbare Schnittstellen hierfür an.

Um BLOB-Felder bequem über CLOU verarbeiten zu können, wurde deshalb eine **SQL-Spracherweiterung** vorgenommen. Damit ist es möglich:

- Inhalte von UNIX-Dateien oder von CLOU-Variablen in BLOB-Felder einzutragen (INSERT/UPDATE) und
- Inhalte von BLOB-Feldern in UNIX-Dateien abzuspeichern oder in CLOU-Variablen zu übernehmen (SELECT).

Einfügen von BLOB-Feldern

Beim Einfügen/ Ändern von BLOB-Feldern werden zwei Möglichkeiten unterstützt: Das Einfügen des Inhalts einer UNIX-Datei und das Einfügen von Stringausdrücken.

Einfügen eines Dateiinhalts

Dazu wird in der betreffenden INSERT-/UPDATE-Anweisung anstelle der Angabe einer CLOU-Variablen (wie üblich gekennzeichnet durch ein vorangestelltes ´$´- Zeichen) die neu eingeführte Builtin-Funktion **BLOBFILE** aufgerufen. Diese erwartet als Argument den Namen der UNIX-Datei, deren Inhalt eingefügt werden soll (Beispiel s.u.).

Einfügen eines Stringausdrucks

Dies erfolgt analog zu oben durch Aufruf der neuen Builtin-Funktion **BLOBTEXT**. Als Argument erwartet diese den einzufügenden Stringausdruck. Dieser wird vor Weitergabe an INFORMIX entsprechend dem aktuellen Konvertierungsmodus (7- / 8-Bit-ASCII) nach ASCII konvertiert.

Achtung!

BYTE- und TEXT-Felder werden hierbei identisch behandelt. Üblicherweise ist diese Möglichkeit jedoch nur bei TEXT-Feldern sinnvoll, während bei BYTE-Feldern hauptsächlich die Datei-Variante angebracht ist. Die Überprüfung des Feldtyps ist etwas zeitaufwendig, deshalb überläßt CLOU die Verantwortung lieber dem Benutzer.

Maximal 5 BLOB-Felder können hierbei pro SQL-Anweisung verarbeitet werden. Werden die neu eingeführten Funktionen außerhalb einer SQL-Anweisung aufgerufen, so liefern sie lediglich ihr Argument als Returnwert zurück.

Beispiel:

```
#
#Q "CREATE DATABASE BLOBBASE"
#Q "CREATE TABLE BLOBTABLE (
        I    INTEGER,
        B    BYTE,
        T    Text
)"
#D i 1
#D t ""
...
#< t "Kommentar:"
#^"SICHERN"
#+

#Q "INSERT INTO BLOBTABLE VALUES
          ($i, $BLOBFILE(DOKUMENT()), $BLOBTEXT(t))"
```

Dieser Baustein speichert das aktuelle Dokument, nachdem es gesichert wurde, mit einem zugehörigen Kommentar in der Datenbank ab. Die (bereits bekannte) Builtin-Funktion **DOKUMENT** liefert hierfür den aktuellen Dokumentnamen.

Selektieren von BLOB-Feldern

Beim Selektieren von BLOB-Feldern werden analog zum Einfügen wieder zwei Möglichkeiten unterstützt:

- Abspeichern in einer Datei

  Dazu muß analog zu oben bei der FETCH-Anweisung anstelle einer CLOU-Variablen der Aufruf der Builtin-Funktion **BLOBFILE** mit dem Namen der Datei als Parameter erfolgen.

- Zuweisung an eine CLOU-Variable

  Dazu wird der Name der CLOU-Variablen direkt wie gewohnt (ohne Aufruf einer Builtin-Funktion) bei der FETCH-Anweisung angegeben (s. Beispiel).

Achtung!

Dies funktioniert sowohl bei BYTE- als auch bei TEXT-Feldern. Der Feldinhalt wird in beiden Fällen als ASCII-String interpretiert und ins HIT-Format konvertiert. Zeichen, die nicht zum HIT-Zeichensatz gehören, werden durch Blanks ersetzt (insbesondere auch Newlines). Bei BYTE-Feldern wird der Benutzer folglich die Ablage in einer Datei (s.o.) vorziehen.

Diese Konvertierung erfolgt auch beim Einlesen des Ergebnisses einer SELECT-Anweisung in eine Liste mit der Listenanweisung #L... Q... .

Beispiel:

```
#
#Q "DATABASE BLOBBASE"
#D i 0
#D t ""
#D tmp "tmpfile"
#< i "Index?"

#Q "DECLARE C CURSOR FOR
        SELECT B, T FROM BLOBTABLE
        WHERE I = $i"
#Q "OPEN C"
#Q "FETCH C INTO $BLOBFILE(tmp), $t"
#Q "CLOSE C"

#? C = 0:
/J
     #^"WECHSELN" $tmp
     #M $t
#
```

Dieser Baustein holt eine Datei aus der Datenbank und legt sie unter dem Namen `tmpfile` ab. Anschließend wird in dieses Dokument gewechselt und der zugehörige Kommentar in der Kommandozeile gemeldet.

5 Übersicht

5.1 CLOU-Anweisungen

Einlesen eines Textbausteins

#B [:][*Ordner/*]*Baustein*	Ein ':' vor dem Bausteinnamen bewirkt, daß der Baustein im Rohformat eingefügt, also in ihm enthaltene Anweisungen nicht ausgeführt werden.

Kommentare

#* [*Kommentar*]#	Als *Kommentar* kann beliebiger Text eingesetzt werden, in dem jedoch kein '#' enthalten sein darf. Der Text kann über beliebig viele Zeilen gehen.

Manuelle Texteinfügung

#F ["*Meldetext*"]	Freie Texteinfügung in der Kommandozeile; der Einfügetext darf weitere CLOU-Anweisungen enthalten.
#V ["*Meldetext*"]	Variable Texteinfügung beliebiger Länge direkt in den Text; CLOU-Anweisungen werden nicht interpretiert.

Umschalten der automatischen Absatzformatierung

#K ["*Modus*"]	Für "*Modus*" kann "ein" bzw. "aus" zum Ein- oder Ausschalten der automatischen Absatzformatierung stehen. Fehlt die *Modus*-Angabe, so wird in den jeweils anderen Zustand umgeschaltet.

Block einfügen

#[*Text* #	Für *Text* kann ein beliebiger Text stehen, der formatgetreu eingefügt werden soll. Das abschließende #-Zeichen muß allein in einer Zeile stehen.

Ausgabe von Datum und Uhrzeit

#T ["[+]*Format*"]	Das ' + ' an erster Stelle der Formatangabe bewirkt rechtsbündige Ausgabe in der aktuellen Zeile.
Format	Standardwert für "*Format*" ist: "T. MMM_JJ"
T	Kalendertag
0T	Kalendertag zweistellig
TT	Name des Wochentags (Abk.)
TTT	Name des Wochentags
M	Kalendermonat
0M	Kalendermonat zweistellig
MM	Monatsname (Abk.)
MMM	Monatsname
J	Jahr zweistellig
JJ	Jahr vierstellig
h	Stunde
0h	Stunde zweistellig
m	Minute
0m	Minute zweistellig
s	Sekunde
0s	Sekunde zweistellig
W	Wievielter Tag im Jahr
0W	Tag im Jahr dreistellig
! oder \	Flucht-Symbole

Aktivieren von HIT-Funktionen

#^ [*Zahl*] "*Tastenname*"	Standardwert für *Zahl* ist 1. Eine Liste aller *Tastennamen* finden Sie unter Punkt 5.3 - "Funktions- und Steuertasten".
#^ [*VarName*]"*Tastenname*"	*VarName* muß für eine Rechenvariable stehen.
#^ "*Tastenname*" "*Parameter*"	Manche Tastennamen erfordern zusätzliche Angaben (*Parameter*).
#^	Kurzform für #^ "VERSTÄRKER".

Ausgabe von Meldungen

#M "*Meldetext*"	Der *Meldetext* erscheint für ca. drei Sekunden in der Kommandozeile.

Alternativen-Auswahl

#A *Alternative1* / ... / *AlternativeN* #	Maximal 20 Alternativen sind möglich. Die jeweils ersten 80 Zeichen jeder Alternative werden dem Benutzer am Bildschirm gezeigt.

Positionieren der Schreibmarke

#G *Spalte* #J *Spalte Zeile*	Für *Zeile* bzw. *Spalte* kann jeweils eine Zahl oder eine Rechenvariable angegeben werden. Positive Werte zählen vom linken, negative vom rechten Rand aus. Der Wert 0 steht jeweils für die aktuelle Spalte bzw. Zeile; Standardwert für *Zeile* ist die aktuelle Zeile.

Position der Schreibmarke abfragen

#Z Spalte Zeile	Für *Spalte* bzw. *Zeile* müssen die Namen zweier Rechenvariablen angegeben werden.

Bildschirmausgabe verlassen

#+	Der Bildschirmpuffer wird ausgegeben, erst dann wird mit dem Einfügen fortgefahren.
#-	Der Bildschirmpuffer wird ausgegeben, mit dem Einfügen wird ohne Unterbrechung fortgefahren.

Einfügevorgang anhalten

#% Wartezeit	Der Bildschirmpuffer wird ausgegeben, danach wird der Einfügevorgang für die angegebene Anzahl von Sekunden angehalten. Für *Wartezeit* muß eine Zahl oder eine Rechenvariable eingesetzt werden.

Abbruch der Bearbeitung

#;	Das Einfügen des Bausteins wird beendet.

Definition von Variablen

#D VarName Ausdruck ["Format"]

Der Name der Variablen *VarName* darf aus bis zu 30 Buchstaben, Ziffern, Unterstrich und Metablank bestehen; das erste Zeichen darf keine Ziffer sein.

Ausdruck

Für *Ausdruck* kann ein Rechenausdruck oder ein Stringausdruck eingesetzt werden. Dadurch wird eine Rechen- bzw. String-Variable definiert.

Rechenausdruck

Ein Rechenausdruck besteht aus einer oder mehreren Zahlen, Variablennamen, Rechenzeichen ' + ', '-', '*', '/', % sowie Klammern '(' und ')'.

Stringausdruck

Ein Stringausdruck besteht aus einer oder mehreren Stringkonstanten (Text in Anführungszeichen) und Variablennamen, die mit dem Aneinanderreihungs-Zeichen '&' verknüpft sind.

Format

"[-][*Feldbreite*][.*Nachkomma*]"

für Rechenvariablen

"[-][*Feldbreite*][.*MaxLänge*]"

für Stringvariablen
Das '-' bewirkt linksbündige Ausgabe. Für *Feldbreite*, *Nachkomma* bzw. *Max-Länge* müssen jeweils Zahlen mit maximal zwei Stellen eingesetzt werden.

Ausgabe eines Variablenwertes

#> ["Format"] VarName

Die Variable *VarName* wird im angegebenen Format ausgegeben. Fehlt die Formatangabe, so wird das Format aus der #D-Anweisung verwendet.

Manuelle Eingabe eines Variablenwertes

#< ["*Bedingung:*"] *Name* ["*Text*"]	Der Benutzer wird mit der Meldung *Text* aufgefordert, einen Wert für die Variable *Name* einzugeben. Ist eine *Bedingung* angegeben, so muß diese nach der Eingabe erfüllt sein, sonst wird der Benutzer erneut aufgefordert.
#« ["*Bedingung:*"] *Name* ["*Text*"]	Zusätzlich wird der letzte Wert von *Name* als Voreinstellung angeboten.
Bedingung	Eine Bedingung besteht aus zwei Rechen- bzw. zwei String-Ausdrücken, zwischen denen ein Vergleichszeichen steht:
<	kleiner
>	größer
< =	kleiner oder gleich
> =	größer oder gleich
< >	ungleich
=	gleich
= =	Stringgleichheit m. Attributen
	Mehrere solcher Bedingungen können mit "und" bzw. "oder" verknüpft sein.
jn	ja/nein-Abfrage

Wertzuweisung

#= VarName Ausdruck	Die Variable *VarName* erhält den Wert von *Ausdruck*.

Teilstring

VarName [*Index1,Index2*] *VarName* [*Index*]	Für *VarName* muß eine Stringvariable eingesetzt werden; für die Indices können Rechenausdrücke stehen.

Definition von Listen-Variablen

#L *ListName* D {*String1*, ...} [& ...]
#L *ListName* D *Liste* [& ...]

ListName darf aus max. 30 Zeichen bestehen. Für *String1* usw. darf ein beliebiger Stringausdruck stehen. *Liste* ist der Name einer bereits definierten Listen-Variable.

#L *ListName* H "*Dokumentname*"
#L *ListName* X "*Dateiname*"
#L *ListName* ! "*shell-Kommando*"
#L *ListName* Q "*Anweisung*"

ListName wird als Inhalt des HIT-Dokuments *Dokumentname* bzw. der SINIX-Datei *Dateiname* oder als Ausgabe von *shell-Kommando* bzw. der Datenbank-Anfrage *Anweisung* definiert.

Liste erweitern

#L *ListName* & {*String1*, ... } [& ...]
#L *ListName* & *Liste* [& ...]

ListName und *Liste* sind die Namen bereits definierter Listen-Variablen. Für *String1* usw. darf ein beliebiger Stringausdruck stehen.

Liste sortieren

#L *ListName* S

Die Elemente von *ListName* werden alphabethisch sortiert.

Zugriff auf Listen-Elemente

ListName [*Index*]

Für *Index* muß ein Rechenausdruck eingesetzt werden.

Länge einer Liste

#L *ListName* L *i*

An Stelle von *i* muß eine Rechenvariable stehen.

Listen-Auswahl-Menü

#L *ListName* A *Indexliste*
#L *ListName* A *i*

Für *Indexliste* steht der Name einer Listen-Variable, für *i* eine Rechenvariable.

Makro-Aufruf

#$ *VarName* bzw.
#$ *VarName*(*Wert1*, ...)

Für *VarName* muß eine Stringvariable eingesetzt werden. *Wert1*... geben Sie an, wenn das Makro *VarName* Platzhalter enthält. Im Makro enthaltene CLOU-Anweisungen werden ausgeführt.

Rückkehr-Anweisung

#R *Wert*

Wert kann ein Rechen- oder ein String-ausdruck sein. Der Typ des ersten Return-*Wertes* bestimmt den Typ einer Funktion.

Fallunterschiedung

#? *Bedingung:*
/J
Ja-Zweig
/N
Nein-Zweig
#

Einer der beiden Zweige kann entfallen, die Reihenfolge umgekehrt werden.

Bedingte Wiederholung

```
#S Bedingung:
   Schleifenkörper
#
```

Ist die *Bedingung* von Anfang an nicht erfüllt, so wird der *Schleifenkörper* übersprungen.

Gezählte Wiederholung

```
#W Anzahl
   Schleifenkörper
#
```

Für *Anzahl* darf eine Zahl oder der Name einer Rechenvariablen, jedoch kein Rechenausdruck eingesetzt werden.

Mehrfachauswahl

```
#C VarName
/Wert1:
   Zweig1
...
/WertN
   ZweigN
[/:
   SonstZweig]
#
```

Für *Wert1* bis *WertN* dürfen Zahlen, Strings oder Variablen, jedoch keine Ausdrücke eingesetzt werden. Stimmt keiner der Werte mit dem der Variablen *VarName* überein, so kommt der *SonstZweig* zur Anwendung.

Variablenwerte merken

```
#» "Dokumentname"
```

schreibt Variablenwerte in das HIT-Dokument *Dokumentname*.

```
#L ListName » "Dokumentname"
```

Schreibt Listen-Elemente in das HIT-Dokument *Dokumentname*. Für *ListName* steht eine Listen-Variable.

Zugriff auf HIT-Dateien

#H ["*Dateiname*"] O ["*Modus*"]
Öffnen einer HIT-Datei mit dem Namen "*Dateiname*". "*Modus*" steht für einen der Werte:
"r" lesen (read)
"w" schreiben (write)
Standardwert für "*Modus*" ist "r".

#H ["*Dateiname*"] C Schließen einer HIT-Datei.

#H ["*Dateiname*"] < *VarName*
Einlesen einer Zeile aus einer HIT-Datei in die Stringvariable *VarName*

#H ["*Dateiname*"] > *Zeile* Schreiben einer Zeile in eine HIT-Datei. Für *Zeile* kann eine String-Variable oder -Konstante angegeben werden.

Aufruf von SINIX-Kommandos

#! "*Kommando*"
Die Standardausgabe des Shell-Kommandos wird in das Zieldokument übernommen; eventuell darin enthaltene CLOU-Anweisungen werden nicht interpretiert. Der Return-Code des Shell-Kommandos wird in der CLOU-Variablen "RC" hinterlegt

Zugriffe auf SINIX-Dateien und Pipes.

#X ["*Dateiname*"] O ["*Modus*"]
Öffnen einer Datei mit dem Namen "*Dateiname*". "*Modus*" steht für einen der Werte:
"r" lesen (read)
"w" schreiben (write)
"a" anfügen (append)
"rw" lesen und schreiben
Standardwert für "*Modus*" ist "r".

#X ["*Shellkommando*"] ! ["*Modus*"]
Aufruf des SINIX-Shell-Kommandos "*Shellkommando*" und Einrichten einer oder zweier Pipes in Abhängigkeit von "*Modus*":
"r" Pipe zum CLOU
"w" Pipe zum *Shellkommando*
"rw" Pipes in beide Richtungen
Standardwert für "*Modus*" ist "r".

#X ["*Name*"] C
Schließen der Datei *Name* / der Pipe zum Shellkommando *Name*.

#X ["*Dateiname*"] R *VarName*
Lesen eines Variablenwertes von einer Datei oder Pipe.

#X ["*Dateiname*"] W *VarName*
#X ["*Dateiname*"] W "Text"
Schreiben eines Variablenwertes oder einer Stringkonstanten auf eine Datei oder Pipe.

#X ["*Dateiname*"] < *VarName*
Einlesen einer ganzen Zeile in eine Stringvariable.

#X ["*Dateiname*"] > *VarName*
#X ["*Dateiname*"] > "Text"
Schreiben eines Variablenwertes oder einer Stringkonstanten mit Zeilenvorschub.

#X ["*Dateiname*"] T *VarName*
Dateizeiger abfragen. *VarName* muß der Name einer Rechenvariablen sein.

#X ["*Dateiname*"] S *Distanz* Dateizeiger positionieren. Siehe auch die Beschreibung der globalen Variablen BASE.

"Eingebaute" SINIX-Kommandos

#X ["*Name*"] U	Datei *Name* löschen.
#X ["*Name*"] M "*Modus*"	Datei-Modus einstellen.
#X ["*Name*"] L "*Link*"	"Link" auf Datei einrichten.
#X ["*Dateiverzeichnis*"] D	Dateiverzeichnis wechseln.

Datenbank-Zugriffe

#Q "*Abfrage*"	Für Abfrage kann eine beliebige Datenbank-Abfrage eingesetzt werden.

5.2 Standard-Funktionen

Dokument- und Bausteinnamen

`DOKUMENT`	Name des aktuellen Dokuments
`BAUSTEIN`	Name des aktuellen Textbausteins
`BAUSTEINPFAD`	Pfadname des aktuellen Textbausteins

Löschpuffer

`C_LÖPU`	zuletzt gelöschtes Zeichen
`TW_LÖPU`	zuletzt gelöschtes Teilwort
`W_LÖPU`	zuletzt gelöschtes Wort
`TZ_LÖPU`	zuletzt gelöschte Teilzeile
`Z_LÖPU`	zuletzt gelöschte Zeile

Suchen und Ersetzen

suchen (*suchtext, ersatztext, ersetzungen,*
großschreibung, suchattribute, nachformatieren,
suchrichtung, textattribute, bestätigen)

Einträge:

suchtext, ersatztext	Suchmuster und Ersatztext als Strings.
ersetzungen	gewünschte Anzahl oder `"*"`
großschreibung *such-, textattribute*	`"gültig"` oder `"ungültig"`
nachformatieren *bestätigen*	`"ja"` oder `"nein"`
suchrichtung	`"vorwärts"` oder `"rückwärts"`.

Return-Werte:

ersetzungen:0	1 Suchwort gefunden 0 Suchwort nicht gefunden
ersetzungen: > 0	Anzahl der ersetzten Wörter

Variablen-Funktionen

`strlen(`*string*`)` liefert die Länge des Stringausdrucks *string*.

`isdef("`*VarName*`")` prüft, ob die Variable *VarName* bereits definiert wurde.

Return-Werte:

0 nicht definiert
1 Rechenvariable
2 Stringvariable
3 Listen-Variable
4 Funktion mit Return-Wert *zahl*
5 Funktion mit Return-Wert *string*
6 Standard-Funktion mit Return-Wert *zahl*
7 Standard-Funktion mit Return-Wert *string*

`SHELLVARIABLE("`*Name*`")` liefert den Inhalt der Shell-Variablen *Name*

`extern("`*Shell-Kommando*`")` ruft das angegebene *Shell-Kommando* auf und liefert als Return-Wert dessen Exit-Status zurück.

`exit("`*Meldetext*`")` beendet CLOU und gibt den *Meldetext* aus.

`listlen("`*LVarName*`")` liefert die Länge der Listenvariablen *LVarName*.

`setatt("`*String*`", "`*Attribut*`"[,`*l*`[,`*r*`]])`
setzt das Attribut im Stringausdruck *String*.

`delatt("`*String*`", "`*Attribut*`"[,`*l*`[,`*r*`]])`
löscht das Attribut im Stringausdruck *String*.

l und *r* bezeichnen den Index des ersten bzw. letzten Zeichens, dessen Attribut verändert werden soll.

`index`(*string, zeichen*) `rindex`(*string, zeichen*)	bestimmen den Index eines Zeichens *zeichen* in einem Stringausdruck *string*. *index* sucht dabei von vorne, *rindex* von hinten.

Datums-Funktionen

`today`	liefert das aktuelle Datum im internen UNIX-Format als Zahl
`fdate`(*datum*, "*format*")	liefert das als Zahl (internes UNIX-Format) angegebene *datum* im bezeichneten *Format* als String
`idate`("*datum*")	liefert die Interndarstellung des angegebenen Datums als Zahl

"Satztechnisch bedingte Leerseite"

Konvertierungs-Funktionen

`ctoi("`*zeichen*`")`	liefert den Dezimalwert von *zeichen*
`itoc(`*zahl*`)`	Umkehrfunktion zu `ctoi()`.
`xtoi("`*hex-zahl*`")`	liefert den Dezimalwert des Strings *hex-zahl*
`itox(`*zahl*`)`	Umkehrfunktion zu `xtoi()`. "

Signalbehandlungs-Funktionen

`signal(`*sig*`[,"`*CLOU-Anweisungen*`"])`
ordnet dem Signal mit der Signalnummer *sig* eine neue Signalroutine zu.

`kill(`*pid*`,`*sig*`)` sendet das Signal mit der Signalnummer *sig* an den Prozeß mit der Prozeßnummer *pid*.

`pause()` legt CLOU bis zum Eintreffen eines Signals schlafen.

Sonstige Funktionen

`getpid()` liefert die Prozeßnummer des CLOU als Zahl zurück.

`setmsgtext(`*anweisung*, *meldetext*`)`
stellt den Default-Meldetext für einige Anweisungen ein.
anweisung darf folgende Werte annehmen:
"A" (Alternativenauswahl),
"B" (Bausteinauswahl) und
"L" (Listenauswahl).

`setopt(`*option*, *modus*`)` stellt CLOU-Optionen ein. Für *option* darf "j", "N", "O", "v", "8" oder "7" stehen, für *modus* "ein" bzw. "aus".

5.3 Funktions- und Steuertasten

Die in eckigen Klammern dargestellten Tastennamen entsprechen keinen einzelnen Funktionstasten, sondern Tasten-Kombinationen. Diese Tasten-Kombinationen dürfen nicht zusammen mit den entsprechenden einzelnen Tastennamen im gleichen Baustein verwendet werden (z.B. nicht gleichzeitig "BLOCKSATZ" und "BLK"!).

Bewegen der Schreibmarke (Cursor)

"C_OBEN"	Cursor nach oben
"C_UNTEN"	Cursor nach unten
"C_RECHTS"	Cursor nach rechts
"C_LINKS"	Cursor nach links
"W_RECHTS"	Wort nach rechts
"W_LINKS"	Wort nach links
"A_OBEN"	Absatz nach oben
"A_UNTEN"	Absatz nach unten
"T_RECHTS"	Positionierung auf Tabulator rechts bzw. in den Randbereich
"T_LINKS"	Positionierung auf Tabulator links bzw. in den Randbereich
"W_ENDE"	Wortende
"A_ENDE"	Absatzende
"VERSTÄRKER"	Verstärkertaste
"S_ENDE"	Satzende

(Text-) Attribute

"UNTERSTREICHEN"	Unterstreichen ein/aus
"FETT"	Fettschrift ein/aus
"DURCHGESTRICHEN"	Durchstreichen ein/aus
"KURSIV"	Kursivschrift ein/aus
"HOCH"	Hochstellen ein/aus
"TIEF"	Tiefstellen ein/aus
"ATTR_LÖSCHEN"	Attribut löschen
"ATTR_SETZEN"	Attribut setzen

Absatzattribute

"LBD"	linksbündig
"RBD"	rechtsbündig
"BLK"	Blocksatz
"ZNT"	zentriert
"GES"	geschützt

Zeilenlineale und Steuerzeilen

"ZL_NEU"	Zeilenlineal einfügen
"ZL_BEARBEITEN"	Zeilenlineal bearbeiten
"L_RAND"	linke Randmarke
"R_RAND"	rechte Randmarke
"T_LBD"	linksbündiger Tabulator
"T_RBD"	rechtsbündiger Tabulator
"T_ZNT"	zentrierter Tabulator
"T_KOMMA"	Dezimaltabulator
"T_PUNKT"	Tausender-Trennstelle
"KLINGELZEICHEN"	Klingelzeichen
"SZ_NEU"	Steuerzeile einfügen
"SZ_BEARBEITEN"	Steuerzeile bearbeiten

Löschen und Einfügen

"C_LÖSCHEN"	Zeichen löschen
"C_EINFÜGEN"	Zeichen einfügen
"W_LÖSCHEN"	Wort löschen
"W_EINFÜGEN"	Wort einfügen
"Z_LÖSCHEN"	Zeile löschen
"Z_EINFÜGEN"	Zeile einfügen
"BACKSPACE"	Zeichen links löschen

Funktionstasten ohne Parameter

"EFG_MODUS"	Einfügemodus umschalten
"WDH_SUCHEN"	Suchen wiederholen
"ABS_FORMATIEREN"	Absatz formatieren
["BLOCKSATZ"	Blocksatzmodus umschalten]
["ZENTRIEREN"	Zentrieren]
"TRENNEN"	Trennstelle setzen
"ABS_TRENNEN"	Absatz trennen
"MARKIEREN"	Markieren und Bearbeiten
"RECHTECK"	Rechteck markieren
"BEREICH_EINFÜGEN"	Markierten Bereich einfügen
"UNDO"	Aktion zurücknehmen
"MODUS"	Modus einstellen
"LINIENGRAPHIK"	Linien-Graphik ein/aus
"BILDSCHIRM"	Bildschirm aufbauen
"MENU"	Taste MENU
"BLANK"	Leertaste
"RETURN"	Taste ↵
"ENDE"	Taste ENDE

Funktionstasten mit Parameter

"SUCHEN" "*Suchwort*" Suchen nach *Suchwort*

"DOK_FORMATIEREN" "*Formatiertabelle*"
Dokument mit *Formatiertabelle* formatieren. Wie beim Drücken der entsprechenden Taste kommen Sie mit:
#^"DOK_FORMATIEREN" ""
wieder in das unformatierte Dokument zurück.

"DOK_WECHSELN" "*Dokumentname*"
In das Dokument mit dem Namen *Dokumentname* wechseln.
Vorsicht! Das alte Dokument wird nicht gesichert; das muß explizit mit der Taste "SICHERN" geschehen:

"SICHERN" "*Dokumentname*"
Das aktuelle Dokument unter dem Namen *Dokumentname* sichern.
Vorsicht! Das Sichern geschieht ohne die gewohnte Rückfrage.

"DRUCKEN" "*Druckername*" Aktelles Dokument auf dem Drucker *Druckername* ausdrucken.

5.4 Globale CLOU-Variable

ERRNO	wird von CLOU automatisch auf 0 (Null) gesetzt, wenn eine Anweisung bezüglich einer HIT- oder SINIX-Datei erfolgreich durchgeführt wurde. Andernfalls bekommt ERRNO einen von 0 verschiedenen Wert.
RC	bekommt den Rückgabe-Code (return code) eines SINIX-Kommandos zugewiesen, das von CLOU aus gestartet wurde. Üblicherweise ist RC = 0, wenn das Kommando fehlerfrei ausgeführt wurde.
OK	hat immer den Wert 0 (Null). Um CLOU-Bausteine lesbarer zu gestalten, kann man z.B. als Bedingung "ERRNO = OK" anstelle von "ERRNO = 0" schreiben.
BASE	muß auf einen der Werte 0, 1 oder 2 gesetzt werden. Die #X-Anweisung "Dateizeiger positionieren" berücksichtigt diesen Wert wie folgt:
0	Die in der Anweisung angegebene Distanz bezieht sich auf den Datei-Anfang.
1	Die Distanz bezieht sich auf die aktuelle Position.
2	Die Distanz bezieht sich auf das Datei-Ende.
SQLCODE	enthält den Fehlercode bei Datenbank-Anfragen. Üblicherweise ist SQLCODE = 0 , wenn die Anfrage fehlerfrei ausgeführt wurde.

5.5 Beispiel

5.5.1 Erstellen einer Rechnung ohne Datenbank

Der folgende Textbaustein ist ein Beispiel für die automatisierte Rechnungserstellung mit CLOU-Bausteinen. Er wird zunächst im eingefügten Zustand (ergänzt durch entsprechende Benutzereingaben) gezeigt und anschließend in seinem "Rohformat", d.h. in dem Format, in dem er erstellt wurde.

Einfügetext

```
●
Sehr geehrte Damen und Herren,●
●
Ihre Bestellung vom 1.9.85 haben wir dankend erhalten.
Entsprechend unseren Vereinbarungen erlauben wir uns,
Ihnen folgendes zu berechnen (Stand der Preisliste:
September_85):●
●
●
Pos.  Menge/Bezeichnung              Einzel-   Gesamtpreis
1        25 Kilo Tomaten              3.80        95.00 DM
2        80 Kilo Pfirsiche            5.50       440.00 DM
3       100 Kilo Bananen              4.60       460.00 DM
4        20 Kilo Erdbeeren            7.00       140.00 DM

Summe:                                          1135.00 DM
abzügl. 5.0 % Rabatt:                             56.75 DM

                                                1078.25 DM
zuzügl. 7.0 % Mehrwertsteuer:                     75.48 DM
Gesamt:                                         1153.73 DM
                                                ==========
●
Wir  bitten Sie, den berechneten Betrag innerhalb von
14 Tagen auf unser Konto zu überweisen.●
●
Mit freundlichen Grüßen,●
```

Rohformat

```
#D menge              0      "4.0"   #* verkaufte Menge #
#D kilopreis          0      "6.2"   #* Kilopreis       #
#D erm_mwst           7      ".1"    #* ermäßigte
                                     Mehrwertsteuer  #
#D rabatt             5      ".1"    #* Rabattsatz für
                                        Großkunden
                                        (in Prozent)    #

#D pos                1      "-6.0"  #* laufende Nummer #
#D rand               -12            #* Einrückung der
                                        Gesamtpreis-
                                        Spalte          #
#D ZWS                0      "8.2"   #* Zwischensumme   #
#D SUM                0      "8.2"   #* Gesamtsumme     #
•
#^"BLK"
Sehr geehrte#A _Frau #F "Anrede:"
             / r Herr #F "Anrede:"
             / _Damen und Herren#,•

•
Ihre Bestellung vom #F "Bestelldatum (Tag und Monat):"
#T "J" haben wir dankend erhalten. Entsprechend unseren
Vereinbarungen erlauben wir uns, Ihnen folgendes zu
berechnen (Stand der Preisliste: #T"MMM_J"):•
•
#K  #*Automatische Absatzformatierung bei Tabellenbearbeitung
      ausschalten! #
•
```

```
Pos.  Menge/Bezeichnung                Einzel-   Gesamtpreis
•
#* Alle Artikel werden in einer Schleife eingelesen.
   Diese wird dann verlassen, wenn als Menge 0
   eingegeben wird. Die Preise der so eingegebenen
   Artikel werden interaktiv abgefragt.
#

#< menge "Menge des Artikels (0: Beenden):"

#S menge > 0 :
    #> pos
    #= pos  pos + 1
    #> menge Kilo
    #F "Bitte Artikelbezeichnung eingeben:"
    #< kilopreis "Bitte Kilopreis eingeben:"
    #= ZWS  menge * kilopreis
    #= SUM SUM + ZWS
    #G 34   #> kilopreis
    #G rand #> ZWS DM

    #* Neue Menge einlesen:
    #
    #< menge "Menge des Artikels (0: Beenden):"

#              #* Abschluß der bedingten Wiederholung #

#G rand
Summe: #G rand  #> SUM DM
```

```
#* Bei Gesamtsummen über 1000 DM gibt es Rabatt:
#
#= ZWS  SUM * rabatt / 100

#? SUM>=1000:
/J
     #= SUM SUM - ZWS
     abzügl. #>rabatt % Rabatt: #G rand #>ZWS DM

     #G rand
     #G rand  #>SUM DM•

#   #* Abschluß der Fallunterscheidung #

#= ZWS SUM*erm_mwst/100  #* Mehrwertsteuer berechnen  #
#= SUM SUM+ZWS           #* und zur Summe hinzuzählen #

zuzügl. #>erm_mwst % Mehrwertsteuer: #G rand #>ZWS DM
Gesamt: #G rand #>SUM DM
#G rand==========

#K   #* Automatische Absatzformatierung wieder einschalten #
•
Wir bitten Sie, den berechneten Betrag innerhalb von
14 Tagen auf unser Konto zu überweisen.•
•
Mit freundlichen Grüßen,•
```

5.5.2 Verwaltung der Datenbank 'obstladen'

Die Verwendung einer INFORMIX 2.1 Datenbank illustrieren fünf CLOU-Bausteine:

- Erzeugen der Datenbank "obstladen"

  Diese Datenbank enthält nur eine Relation "artikel" mit den Feldern:

  - Artikelnummer,
  - Artikelbezeichnung,
  - Preis und
  - Lagermenge.

  Diese Beispiel-Datenbank dient als Grundlage für die restlichen vier Bausteine.

- Eintragen von neuen Datensätzen

  Neue Artikelbezeichnungen und Preise werden vom CLOU-Benutzer interaktiv abgefragt und in die Datenbank aufgenommen. Gleichzeitig werden die neu erstellten Datensätze im Zieldokument protokolliert.

- Ändern von Datensätzen

  Preis- und Lagermengen-Änderungen werden in die Datenbank eingetragen und gleichzeitig protokolliert.

- Erstellen einer Rechnung

  Eine Rechnung wird erstellt, indem der Benutzer der Reihe nach verschiedene Artikelnummern sowie die jeweils gelieferte Menge eingibt. Alle weiteren benötigten Angaben werden der Datenbank entnommen, wobei die Datei auch auf den neuesten Stand gebracht wird (gelieferte Menge wird "abgebucht").

- Ausdrucken einer Artikel-Übersicht

  Alle in der Datenbank verzeichneten Artikel werden alphabetisch nach der Bezeichnung sortiert und mit den zugehörigen Daten in das Zieldokument ausgegeben. Ist der Lagerbestand kleiner als 100 kg, so wird er fett gedruckt, um nachzubestellende Artikel hervorzuheben.

Erzeugen der Datenbank "obstladen"

Rohformat

```
#
#* Dieser Textbaustein fügt keinen Text ins aktuelle
   Dokument ein, sondern erzeugt lediglich eine neue
   Datenbank "obstladen".
#

#Q "CREATE DATABASE obstladen"

#Q "CREATE TABLE artikel (
        a_nummer        SERIAL,
        a_bezeichnung   CHAR(20),
        a_preis         DECIMAL(32,2),
        a_bestand       INTEGER )"

#* Obige Relation enthält zu jedem Artikel folgende
   Angaben: Artikelnummer, Artikelbezeichnung (max.
   20 Zeichen), Artikelpreis und Lagerbestand in Kilo
#

#* Da häufig über die Artikelnummer auf einen Datensatz
   zugegriffen wird, wird ein Index über die
   Artikelnummer angelegt:
#
#Q "CREATE INDEX i_nummer ON artikel(a_nummer)"

#* Artikelbezeichnungen müssen eindeutig sein:
#
#Q "CREATE UNIQUE INDEX i_bezeichnung ON
    artikel (a_bezeichnung)"
```

Eintragen von neuen Datensätzen

Einfügetext

```
Neu aufgenommmene Artikel vom 25. 9. 85●
●
Nummer   Bezeichnung                    Preis●
●
1        Tomaten                         3.80●
2        Pfirsiche                       5.50●
3        Bananen                         4.60●
4        Erdbeeren                       7.00●
```

Rohformat

```
#
#K
#* Dieser Textbaustein fragt interaktiv die Daten von
   Datensätzen für die Datenbank "obstladen" ab, fügt
   diese in die Datenbank ein und protokolliert die
   eingefügten Daten im aktuellen Dokument. Die
   Artikelnummer wird durch INFORMIX selbst generiert
   (Typ "serial"), der Bestand wird mit 0 vorbesetzt.
#

#Q "DATABASE obstladen"

#Q "DECLARE c CURSOR FOR
    SELECT a_nummer
    FROM artikel
    WHERE a_bezeichnung = $bezeichnung"

#D bezeichnung ""
#D nummer      0
#D preis       0  "8.2"
#D meldung     ""
Neu aufgenommmene Artikel #T "vom T. M. J"●
●
```

```
Nummer   Bezeichnung                    Preis●
●
#<bezeichnung "Bezeichnung des Artikels eingeben
               (<RETURN> bricht ab):"

#S bezeichnung <> "":

    #= meldung "Bitte Preis für " & bezeichnung &
               " eingeben:"
    #<  preis $meldung

    #Q "INSERT INTO artikel
        (a-bezeichnung, a-preis, a-bestand)
        VALUES ($bezeichnung, $preis, 0)"

    #Q "OPEN c"
    #Q "FETCH c INTO $nummer"
    #Q "CLOSE c"

    #> nummer
    #G 10
    #> bezeichnung
    #G 36
    #> preis●

    #< bezeichnung "Beichnung des Artikels eingeben
                    (<RETURN> bricht ab):"
#
```

Ändern von Datensätzen

Einfügetext

```
Geänderte Artikel am 25. 9. 85●
●
Nummer   Bezeichung                    Preis  Bestand●
●
alte Daten:●
1        Tomaten                       3.80        0●
neue Daten:●
1        Tomaten                       3.80      200●
●
alte Daten:●
4        Erdbeeren                     7.00        0●
neue Daten:●
4        Erdbeeren                     7.00       50●
●
alte Daten:●
3        Bananen                       4.60        0●
neue Daten:●
3        Bananen                       3.20      300●
●
alte Daten:●
2        Pfirsiche                     5.50        0●
neue Daten:●
2        Pfirsiche                     2.99      250●
●
```

Rohformat

```
#
#K
#* Dieser Textbaustein sucht Datensätze aus der
   Datenbank "obstladen" anhand des Attributs
   "a_bezeichnung" und bietet diese zur Korrektur an.
   Geändert werden können:
   Artikelbezeichnung, Preis und Bestandsmenge.
   Die geänderten Datensätze werden protokolliert.
#

#Q "DATABASE obstladen"

#D bezeichnung ""
#D nummer      0
#D bestand     0  "8.0"
#D preis       0  "8.2"
#D meldung     ""

#Q "DECLARE c CURSOR FOR
    SELECT a_nummer, a_preis, a_bstand
    FROM artikel
    WHERE a_bezeichnung = $bezeichnung"

Geänderte Artikel am #T "T. M. J"●

Nummer   Bezeichnung                    Preis  Bestand●
─────────────────────────────────────────────────────────
●
#<bezeichnung
    "Artikelbezeichnung (<RETURN> bricht ab):"

#S bezeichnung <> "":

     #Q "OPEN c"
     #Q "FETCH c INTO $nummer, $preis, $bestand"
     #? c = OK:
     /J
         alte Daten:●

         #> nummer
         #G 10
         #> bezeichnung
         #G 36
         #> preis
```

```
            #> bestand●
            #= meldung "Bitte neuen Preis für " &
                bezeichnung & " eingeben:"
            #< preis $meldung

            #= meldung "Bitte neuen Bestand für " &
                bezeichnung & " eingeben:"
            #< bestand $meldung

            #Q "UPDATE artikel
                SET a_preis = $preis,
                    a_bestand = $bestand
                WHERE a nummer = $nummer"

            neue Daten:●

            #> nummer
            #G 10
            #> bezeichnung
            #G 36
            #> preis
            #> bestand●
            ●

        /N
            #= meldung "Artikel " & bezeichnung &
                " existiert nicht!"
            #M $meldung
        #

        #< bezeichnung
           "Artikelbezeichnung (<RETURN> bricht ab):"
        #Q "CLOSE c"
    #
```

Erstellen einer Rechnung

Einfügetext

```
●
Sehr geehrte Damen und Herren,●
●
Ihre  Bestellung  vom 1. 9. 85 haben wir dankend erhal-
ten.  Entsprechend unseren Vereinbarungen erlauben  wir
uns,  Ihnen  folgendes zu berechnen (Stand der Preisli-
ste: September_85):●
●
●
Pos.  Menge/Bezeichnung              Einzel-   Gesamtpreis
1        25 Kilo Tomaten              3.80        95.00 DM
2        80 Kilo Pfirsiche            5.50       440.00 DM
3       100 Kilo Bananen              4.60       460.00 DM
4        20 Kilo Erdbeeren            7.00       140.00 DM

Summe:                                          1135.00 DM
abzügl. 5.0 % Rabatt:                             56.75 DM

                                                1078.25 DM
zuzügl. 7.0 % Mehrwertsteuer:                     75.48 DM
Gesamt:                                         1153.73 DM
                                                ==========
●
Wir bitten Sie, den berechneten Betrag innerhalb von
14 Tagen auf unser Konto zu überweisen.●
●
Mit freundlichen Grüßen,●
```

Rohformat

```
#
#* Erstellung einer Rechnung.
   Anhand der Artikelnummer werden die
   benötigten Daten aus der Datenbank gelesen.
   Gleichzeitig wird der Bestand um die verkaufte
   Menge vermindert. #
```

```
#D menge              0     "4.0"     #* verkaufte Menge #
#D bezeichnung        ""              #* Art.-Bez.        #
#D nummer             0               #* Art.-Nummer      #
#D bestand            0               #* Bestandsmenge    #
#D preis              0     "6.2"     #* Kilopreis        #
#D erm_mwst           7     ".1"      #* erm. Mwst.       #
#D rabatt             5     ".1"      #* Rabattsatz       #

#D pos                1     "-6.0"    #* laufende Nummer #
#D ZWS                0     "8.2"     #* Zwischensumme   #
#D SUM                0     "8.2"     #* Gesamtsumme     #

#Q "DATABASE obstladen"

#Q "DECLARE c CURSOR FOR
    SELECT a_preis, a_bezeichnung, a_bestand
    FROM artikel
    WHERE a nummer = $nummer"
#^ "BLK"

Sehr geehrte#A _Frau #F"Anrede:" / r Herr #F"Anrede:" /
              _Damen und Herren#,●
●
Ihre Bestellung vom
#F "Bestelldatum (Tag und Monat):" #T"J" haben
wir dankend erhalten. Entsprechend unseren
Vereinbarungen erlauben wir uns, Ihnen folgendes zu
berechnen (Stand der Preisliste: #T"MMM_J"):●
●
```

```
#K  #* Automatische Absatzformatierung bei Tabellenbearbeitung
       ausschalten! #

Pos.  Menge/Bezeichnung              Einzel-   Gesamtpreis●
●
#< menge "Menge des Artikels (0: Beenden):"

#S menge>0:
    #<nummer "Bitte Artikelnummer eingeben:"

    #Q "OPEN c"
    #Q "FETCH c INTO $preis, $bezeichnung, $bestand"
    #Q "CLOSE c"

    #? c = OK:
    /J
        #>pos #=pos pos+1
        #>menge Kilo
        #=ZWS menge*preis
        #=SUM SUM+ZWS
        #>bezeichnung
        #G 34 #>preis #G -12 #>ZWS DM●
        #* Bestand aktualisieren: #
        #Q "UPDATE artikel
            SET a_bestand = a_bestand - $menge
            WHERE a_nummer = $nummer"
    /N
        #M "Artikelnummer existiert nicht!"
    #

    #< menge "Menge des Artikels (0: Beenden):"

#

#G -12

Summe: #G -12 >SUM DM●
```

```
#* Bei Gesamtsummen über 1000 DM gibt es Rabatt: #
#= ZWS SUM*rabatt/100

#? SUM>=1000:
/J
    #= SUM SUM-ZWS
       abzügl. #>rabatt % Rabatt: #G -12 #>ZWS DM

    #G -12

    #G -12 #>SUM DM ●

#   #* Abschluß der Fallunterscheidung #

#= ZWS SUM*erm_mwst/100 #* Mehrwertsteuer berechnen...#
#= SUM SUM+ZWS          #* und zur Summe hinzuzählen   #

zuzügl. #>erm_mwst % Mehrwertsteuer: #G -12 #>ZWS DM●
Gesamt: #G -12 #>SUM DM
#G -12==========●
●
#K   #* Automatische Absatzformatierung wieder einschalten #

Wir bitten Sie, den berechneten Betrag innerhalb von
14 Tagen auf unser Konto zu überweisen.●
●
Mit freundlichen Grüßen, ●
```

Ausdrucken einer Artikel-Übersicht

Einfügetext

```
Artikel-Liste vom 25. 9. 85●
●
Nummer   Bezeichnung                    Preis  Bestand●
●
3        Bananen                         3.20      200●
4        Erdbeeren                       7.00       30●
2        Pfirsiche                       2.99      170●
1        Tomaten                         3.80      175●
```

Rohformat

```
#
#K
#* Dieser Textbaustein gibt alle Datens tze, die sich
   in der Datenbank befinden, nach
   Artikelbezeichnungen geordnet aus.
   Bei Artikeln, deren Bestand unter 100 Kilo ist,
   wird der Bestand fett gedruckt #

#Q "DATABASE obstladen"

#D bezeichnung ""
#D nummer      0
#D bestand     0  "8.0"
#D preis       0  "8.2"

#Q "DECLARE c CURSOR FOR
    SELECT a_nummer, a_bezeichnung, a_preis, a_bestand
    FROM artikel
    ORDER BY a_bezeichnung"

#D next  "#Q ""FETCH c INTO $nummer, $bezeichnung,
                 $preis, $bestand"""

Artikel-Liste vom #T "T. M. J"●
●
```

```
Nummer   Bezeichnung                          Preis  Bestand●
●
#Q "OPEN c"
#$ next

#S c = OK:

    #> nummer
    #G 10
    #> bezeichnung
    #G 36
    #> preis

    #? bestand <= 100:
    /J
        #^ "FETT"
        #> bestand
        #^ "FETT"●
    /N
        #> bestand●
    #

    #$ next
#
#Q "CLOSE c"
```

5.6 Aufruf-Optionen

Über die Shell-Variable `CLOUOPT` können bestimmte Eigenschaften von CLOU geändert werden. Diesbezügliche Wünsche teilen Sie CLOU mit, imdem Sie der Variable `CLOUOPT` CLOU-Aufruf-Optionen zuweisen. Die Zuweisung muß für jeden Benutzer in der Datei `.profile` erfolgen und gilt dann für jede CLOU-Anwendung. Folgende Optionen können übergeben werden:

-A	Bausteinarchiv-Schlüssel
-S	Protokoll-Datei
-O	Wiederholte Bausteinabfrage
-N	Abschließende Meldung unterdrücken
-v	nationale Konvertierung bei `#X`
-j	Automatisches Runden
-i	RC-Baustein
-8	Konvertierung von/nach 8-Bit-ASCII beim Datenaustausch mit INFORMIX
-7	Konvertierung von/nach 7-Bit-ASCII beim Datenaustausch mit INFORMIX

Beispiel:

```
CLOUOPT="-N -v"
export CLOUOPT
```

Bausteinarchiv-Schlüssel

Außer dem von HIT-MENÜ übergebenen Bausteinarchiv, kann CLOU weitere Archive nach Bausteinen durchsuchen, wenn Sie ihm diese Archive über die Option `-A` bekanntgeben.

Syntax

-A *X*:*Stufe*:*Pfadname*:[*Std.ordner*]

Mehrere (max. 8) solcher Einträge können - durch ':' getrennt - angegeben werden. *X* ist der Kennbuchstabe für das Archiv, der später im CLOU-Baustein bei der `#B`-Anweisung angegeben werden muß. Der bei `#B` angegebene Baustein wird im Dateiverzeichnis *Pfadname* gesucht.

Entspricht *Pfadname* nicht dem Pfad eines Ordners, sondern dem eines Archivs oder einer Benutzerkennung, muß mit *Stufe* angegeben werden, in welcher Hierarchiestufe sich die Bausteine unter dem angegebenen Dateiverzeichnis *Pfadname* befinden; d.h. bei 1 werden Bausteine im angegebenen Dateiverzeichnis gesucht, bei 2 wird im Dateiverzeichnis eine Ordner/Baustein-Struktur erwartet... Im CLOU-Baustein muß dann bei der #B-Anweisung die erwartete Struktur angegeben werden.

Std.ordner definiert einen Ordner, in dem gesucht wird, falls im CLOU-Baustein nicht die erwartete Struktur angegeben ist (z.B. nur ein Bausteinname anstelle von Ordner/Bausteinname). Falls kein *Std.ordner* eingetragen ist, wird in allen Ordnern gesucht bzw. der jeweils letzten Wert verwendet.

Wurde im CLOU-Baustein noch kein Archivschlüssel verwendet, wird standardmäßig in dem über HIT-MENÜ eingestellten Archiv gesucht. Die Angabe eines Archivschlüssels macht dann das entsprechende Archiv zum Standard-Archiv. Soll danach wieder das über HIT-MENÜ eingestellte Archiv verwendet werden, muß dies CLOU durch den Archivschlüssel "." mitgeteilt werden.

Beispiel:

```
CLOUOPT="-A Z:2:/usr/zentral::d:1:/usr/$USER/Baustein/clou:"
```

Mögliche #B-Anweisungen sind dann

```
#B "hallo"
#B "d baustein7"
#B "Z briefkopf/intern3"
#B ". hallo"
```

Protokoll-Datei

Durch Setzen der Option `-S` können die Namen aller verarbeiteten Textbausteine in einer Datei protokolliert werden. Die Protokollierung erfolgt zeilenweise mit Uhrzeit und Pfadnamen.

Syntax

-S *datei*[.*X*]

Durch Erweiterung von *datei* (getrennt durch einen Punkt), kann die Protokollierung auf diejenigen Bausteine beschränkt werden, deren Archivschlüssel nach dem Punkt aufgeführt ist. Die Protokollierung erfolgt dann ohne Zeitangabe, jedoch mit Archivschlüssel (falls mehr als einer angegeben ist).

Wiederholte Bausteinabfrage

Nach Setzen der Option `-O` fragt CLOU nach Beendigung eines Einfügevorgangs nach weiteren Bausteinen. Beim Einfügen mehrerer Bausteine entfällt damit das wiederholte Drücken von F12 **Baustein einfügen.**

Abschließende Meldung unterdrücken

Nach Setzen der Option `-N` gibt CLOU nach Beendigung eines Einfügevorgangs die Meldung `"Textbaustein ... eingefügt!"` nicht aus.

Nationale Konvertierung bei #X

Nach Setzen der Option `-v` gelten die Konventionen der nationalen Konvertierung nicht nur für Datenbank-Anfragen, sondern auch für die Dateibearbeitung mittels `#X`-Anweisung.

RC-Baustein

CLOU-Anweisungen, die oft benötigt werden (z.B. Definition bestimmter Funktionen oder Makros), können in einem **RC-Baustein** hinterlegt werden. Dieser wird bei jedem Aufruf von CLOU vor den eigentlichen Bausteinen abgearbeitet.

Syntax

-i *bausteinname*

Konvertierung beim Datenaustausch mit INFORMIX

Standardmäßig wird beim Austausch von Daten mit INFORMIX von/nach 7-Bit-ASCII (ISO 646) konvertiert, falls für das HIT-System der 7-Bit-Modus eingestellt ist, andernfalls von/nach 8-Bit-ASCII (ISO 8859-1).

Durch Setzen der Option -7 bzw. -8 wird der Konvertierungsmodus für den Zugriff auf INFORMIX unabhängig davon explizit eingestellt.

6 Das Menü für den Systemverwalter

Zur Pflege der Druckeraufrufe und der Ausnahmedatei steht dem HIT-Systemverwalter ein eigenes Menü zur Verfügung.
Sie erhalten dieses Menü, indem Sie folgendermaßen vorgehen:

- Melden Sie sich unter der Benutzerkennung `admin` an (die üblicherweise durch ein Kennwort geschützt ist).

  Daraufhin wird automatisch die Bedienoberfläche `COLLAGE` geladen und gestartet.

- Wählen Sie im Menü `Desktop` den Befehl `Anwendungen` aus.

- Wählen Sie `HIT-Menü`.

  Sie sehen nun das Menüsystem von HIT auf dem Bildschirm.

- Mit dem Kommando `FUNKTIONEN` wird im Arbeitsbereich ein Menü eröffnet, das eine Reihe von Funktionen enthält.

- Wählen Sie die Funktion `Sys-admin` aus.

Im Anschluß daran wird dieses Menü ausgegeben:

```
        H I T - S Y S T E M - V E R W A L T U N G
        =========================================

        e - Druckeraufruf erstellen/aendern (Formular)

        a - Druckeraufruf aendern (vi)

        l - Druckeraufruf loeschen

        u - Druckeraufruf umbenennen

        v - Ausnahmedatei (Silbentrennung) pflegen

                    h - Wenn Sie Hilfe brauchen

Ihre Auswahl ==>                          Menue verlassen:<Ende>
```

Die Funktionen `e`, `a`, `l`, und `u` dienen der Pflege der Druckeraufrufe.
Die Funktion `v` führt in die Ausnahmedatei.
Die Funktion `h` gibt Ihnen Hilfe-Informationen aus.

6.1 e - Druckeraufruf erstellen/ändern (Formular)

Mit dieser Funktion können Sie neue Druckeraufrufe erstellen, bzw. vorhandene ändern.

Zu Beginn werden Sie aufgefordert, den Namen für den gewünschten Druckeraufruf einzugeben. Dieser darf nur aus Buchstaben, Zahlen, Punkt und Bindestrich zusammengesetzt werden, wobei Punkt oder Bindestrich nicht als erstes Zeichen stehen dürfen! Fehlerhafte Eingaben dürfen Sie nur mit der Taste ⌫ korrigieren.

Wenn an Ihrer Anlage mehrere Drucker angeschlossen (konfiguriert) sind, so erscheinen danach die Namen dieser Drucker. Angeboten werden dabei nur die vom Betriebssystem unterstützten Siemens-Standarddrucker, für die auch ein Druckertreiber ("backend") vorhanden ist (z.B. 9001, 9022 usw.). Die Auswahl wird zudem auf diejenigen Drucker beschränkt, für die bei der Druckerkonfiguration eine eigene Druckergruppe definiert wurde.

Geben Sie den Namen des Druckertyps ein und schließen Ihre Eingabe mit ↵ ab.

Wurde der ausgewählte Drucker bei der Konfiguration mehreren Druckergruppen zugeordnet, werden Ihnen alle möglichen Druckergruppen angezeigt.

Geben Sie den Namen der gewünschten Druckergruppe ein und schließen Ihre Eingabe mit ↵ ab.

Beispiel

Es wurden folgende Drucker und Druckergruppen konfiguriert:

Drucker:	Druckergruppe:
9001	`GRUPPE1`
9001	`TINTE`
9001, 9012	`STD`
9022	`LASER`

Zur Auswahl angeboten werden nur die Drucker 9001 und 9022, nicht dagegen der Drucker 9012 (für ihn wurde keine eigene Druckergruppe definiert).
Beim Drucker 9001 kann zusätzlich noch zwischen den Druckergruppen `GRUPPE1` und `TINTE` ausgewählt werden.

Danach erscheint folgendes Formular:

```
                    DRUCKERAUFRUF ERSTELLEN / VERAENDERN

Druckertyp: .......  | Gruppe: ..........  | Aufrufname: ............

 Druckertabelle                                  S:   > ............ <

 Papierzuführung (Endlos oder Einzelblatt)       S:   > ........... <

 Papierlänge (1-99)                              E:   > .. <

 Erste Seite aus Schacht                         S:   > . <
 Folgeseiten aus Schacht                         S:   > . <

 Geschuetzten Formulartext drucken (j/n)         S:   > . <

 Heftrand ungerade Seitenzahl (0-99)             E:   > .. <
 Heftrand gerade Seitenzahl   (0-99)             E:   > .. <

 Zeilenlänge (1-250)                             E:   > ... <

 Eingabefeld: E                                  Schalterfeld: S

 Daten übernehmen:<RETURN>                       Menue verlassen:<Ende>
```

Im oberen Teil des Formulars werden Ihnen die bereits erfolgten Eingaben (Druckertyp, Druckergruppe, Aufrufname) angezeigt.

Das Formular unterscheidet zwei Arten von Feldern:

- Schalterfelder
  Diese sind mit dem Buchstaben S gekennzeichnet. Durch Drücken der Leertaste können Sie einen der vorgegebenen Werte auswählen.

- Eingabefelder
  Diese sind mit dem Buchstaben E gekennzeichnet. In ihnen können direkt die gewünschten Zahlenwerte eingetragen werden.

Abhängig vom ausgewählten Druckertyp können manche Felder nicht angesprochen werden. So kann z.B. beim Drucker 9001 nicht zwischen Endlosformular und Einzelblatt gewählt werden.

Welche Bedeutung haben die Felder?

Druckertabelle

Wählen Sie aus den angebotenen Druckertabellen diejenige aus, die für den ausgewählten Drucker paßt. Über die Erstellung von Druckertabellen informiert Sie das Kapitel 7.2.

Papierzuführung

Abhängig davon, wie an dem betreffenden Drucker die Papierzuführung erfolgt, wählen Sie hier `Endlos` oder `Einzelblatt` aus.

Papierlänge (1-99)

Den Zahlenwert, den Sie hier eingeben müssen, entspricht der Anzahl von Zeilen, die maximal auf eine Seite gedruckt werden können (bei einzeiligem Zeilenabstand). Nach Erreichen dieser Zeilenzahl erfolgt, falls im Dokument kein Seitenvorschub vorgesehen ist, in jedem Fall ein Seitenvorschub.

Erste Seite aus Schacht

Eine Eingabe in diesem Feld ist nur sinnvoll, wenn der betreffende Drucker über eine Einzelblattzuführung verfügt. Sie können damit die erste Seite zum Beispiel auf Papier mit Briefkopf drucken. Durch Drücken der Leertaste können Sie für die erste Seite Schacht 1, 2 oder 3 auswählen.

Folgeseiten aus Schacht

Wenn Sie im Feld `Erste Seite aus Schacht` einen Wert eingetragen haben, können Sie durch eine Eingabe in diesem Feld bestimmen, aus welchem Schacht das Papier für Folgeseiten eingezogen werden soll. Mögliche Eingaben sind wieder 1, 2 oder 3.
Wenn Sie keinen Eintrag machen, werden alle Blätter aus demjenigen Schacht eingezogen, der im Feld `Erste Seite aus Schacht` angegeben wurde.

Geschützten Formulartext drucken (j/n)

Wenn Sie die Einstellung `j` wählen, wird beim Ausdruck von HIT-Formularen der geschützte Text gemeinsam mit dem Feldtext ausgedruckt.

Heftrand ungerade Seitenzahl (0-99)

Wenn der Text beim Ausdruck nach rechts verschoben werden soll (z.B. um Platz für eine Lochung oder für späteres Binden zu erhalten), können Sie hier die Anzahl der Schreibstellen für den Versatz eingeben.

Heftrand gerade Seitenzahl (0-99)

Analog `Heftrand ungerade Seitenzahl`; der Text wird beim Ausdruck nach rechts verschoben.

Zeilenlänge (1-250)

Wenn Sie hier einen Wert eintragen, werden nur die angegebenen Zeichen je Zeile ausgedruckt.

6.2 a - Druckeraufruf ändern (vi)

Diese Funktion führt zu einem Menü, in dem Sie unter den vorhandenen Druckeraufrufen wählen können. Wenn Sie danach [↵] drücken, wird der vi mit dem ausgewählten Druckeraufruf aufgerufen.

Mit dieser Funktion können Sie Einträge in die Shell-Skripten machen, die mit der menügesteuerten Funktion *e - Druckeraufruf erstellen/aendern* nicht möglich sind.

Der Aufbau der Druckeraufrufe ist in Kapitel 7.1 beschrieben.

6.3 l - Druckeraufruf löschen

Diese Funktion führt zu einem Menü, in dem Sie den gewünschten Druckeraufruf auswählen können.
Wenn Sie danach [↵] drücken, wird der ausgewählte Aufruf gelöscht.

6.4 u - Druckeraufruf umbenennen

Diese Funktion führt zu einem Menü, in dem Sie den gewünschten Druckeraufruf auswählen können.
Danach erhalten Sie ein Eingabefeld, in das Sie den neuen Namen des Aufrufs eintragen können.
Korrekturen dürfen nur mit der Taste [⌫] erfolgen!

6.5 Ausnahmedatei des Trennprogramms pflegen

In diesem Kapitel ist beschrieben, wie Sie die bestehende Ausnahmedatei des Trennprogramms verändern können. Dies ist vor allen Dingen bei fremdsprachigen Wörtern von Vorteil.

Sie gehen dabei folgendermaßen vor:

- Rufen Sie innerhalb des Menüs `'HIT-SYSTEM-VERWALTUNG'` die Funktion `v - Ausnahmedatei pflegen` auf.
- Sie werden aufgefordert, das entsprechende Sprachkennzeichen einzugeben, wobei die möglichen Werte in Klammern angegeben sind. Machen Sie Ihre Eingabe und schließen sie diese mit (↵) ab.
- Es öffnet sich ein HIT-Dokument mit dem Namen `dhtemp`, in dem alle gespeicherten Ausnahmen enthalten sind.
- Sie können jetzt mit den üblichen Editorfunktionen auch unsortiert die Ausnahmen ändern, löschen oder ergänzen. Wie die Ausnahmen eingetragen werden müssen, können Sie dem Abschnitt 'Ausnahmeschreibweise' (siehe unten) entnehmen.
- Wenn Sie alle Änderungen in der Ausnahmedatei gemacht haben, drücken Sie (ENDE) . Sie erhalten die Meldung
  `Wollen Sie Ihr Dokument sichern (ja=j, nein=n)?.`
  Geben Sie `j` ein, und im fehlerfreien Fall werden die Ausnahmen sortiert in die Datei `dhtemp` für den direkten Zugriff gespeichert. Sie stehen damit für die Silbentrennlogik zur Verfügung.

Ausnahmeschreibweise

Für den Eintrag von Ausnahmen gibt es ein paar Regeln, die Sie unbedingt beachten müssen:

1. Eine Ausnahme muß aus mindestens 2 Zeichen bestehen und darf nur maximal 49 Zeichen lang sein. Bei einer fehlerhaften Eingabe erhalten Sie nach der Überprüfung die Meldungen

   `<<<<<T001: Eintrag zu kurz` bzw.
   `<<<<<T004: Eintrag zu lang`

2. Die ersten zwei Zeichen einer Ausnahme dürfen keine Sonderzeichen sein, sonst Fehlermeldung

   `<<<<<T003: Unerlaubtes Zeichen am Wortanfang`

3. Folgende Sonderzeichen mit ihren Bedeutungen sind in Ausnahmen erlaubt:

   - Trennzeichen: Das Zeichen '-' wird bei jeder erlaubten Trennstelle eingetragen.
     z.B. Tren-nun-gen

   + Trennzeichen mit Einfügung:

     Das Zeichen ' + ' bewirkt, daß das vorhergehende Zeichen zusätzlich eingefügt wird;
     z.B.: voll + a-den ergibt voll-la-den

     Hinweis
     Das Zeichen ' + ' wirkt nur bei Sprachen mit einer entsprechenden grammatikalischen Regelung, z.B. im Deutschen.

   # Trennzeichen mit Eliminierung:

     Das Zeichen '#' bewirkt, daß der davorstehende Buchstabe im Falle einer Worttrennung nicht geschrieben wird.
     Z.B. holländisch Taxie#tje ergibt Taxi-tje

     Hinweis
     Die Zeicheneliminierung '#' wirkt nur in der holländischen Sprache.

   * Ausnahmenverkürzung:

     Das Zeichen '*' bewirkt, daß bis zu dieser Stelle wie angegeben getrennt werden soll. Alle weiteren Buchstaben werden von der Trennlogik bearbeitet,
     z.B. bewirkt lau-f* folgende Trennungen:
     lauf, lau-fe, lau-fen, lau-fend, lau-fen-den.
     Achtung: lau-fsper-re!!

     Ein Eintrag ohne '*' wird in Worten wiedererkannt, die bis zu 3 Buchstaben länger sind als der Eintrag:
     z.B. lau-f erzeugt Trennstellen in:
     lau-fe, lau-fen, lau-fend;
     aber nicht: laufende!

. steht für beliebigen Vokal:

Das Zeichen '.' bewirkt, daß an dieser Stelle ein beliebiger Vokal stehen kann:
z.B.: Zei-t.n ergibt
Zei-ten, Zei-tung

, steht für beliebigen Konsonanten:

Das Zeichen ',' bewirkt, daß an dieser Stelle ein beliebiger Konsonant stehen kann
z.B.: lo-,en bewirkt
lo-ben, lo-ten

Als übrige Zeichen sind sämtliche Klein- und Großbuchstaben incl. der nationalen Sonderzeichen erlaubt.

Hinweis

Werden bei Ausnahmen andere Zeichen als die oben beschriebenen angegeben, so bekommen Sie die Fehlermeldung:

```
<<<<<T002: Unerlaubtes Zeichen im Wort.
```

Sämtliche Ausnahmeregeln lassen sich untereinander kombinieren. Die folgende Tabelle zeigt dazu einige Beispiele:

Beispiel-Ausnahme	bewirkt folgende Trennergebnisse		
vill + ad	vill-laden aber:	vill-laden vil-la-den-de	(drittes l wird eingefügt) (mehr als 3 Buchstaben folgen !)
vall + ad*	vall-la-den-der		(drittes l wird eingefügt) (danach weiter mit Trennlogik)
taxie#tje	taxie-tje	taxie-tje	(Eliminierung des e)

Fehlerbehandlung

Sollte Ihnen beim Eintrag von Ausnahmen ein Fehler unterlaufen sein, so erhalten Sie einen entsprechenden Hinweis, und die Ausnahmedatei wird nochmals in HIT geladen.
Die fehlerhaften Zeilen sind am Ende mit der Zeichenkette < < < < < und einer Fehlernummer (T001-T004) gekennzeichnet. Über die Funktion **Suchen** können Sie die fehlerhaften Stellen direkt anspringen und korrigieren.
Beachten Sie dabei bitte, daß die Fehlernummern als Text eingetragen sind und entfernt werden müssen!
Die HIT-Datei wird so oft gestartet, bis keine Fehler mehr vorhanden sind.

7 Druckersteuerung

7.1 Aufbau der Druckeraufrufe

Druckeraufrufe sind Shell-Prozeduren, die Programmaufrufe zur Druckaufbereitung von HIT-Dokumenten enthalten. Die Ausgabe der Programme wird dem SINIX-Kommando `lpr` übergeben.

Anstelle einer Druckaufbereitung können auch beliebige andere Operationen wie z.B. Konvertierung des HIT-Dokuments in ASCII-Format und anschließender File-Transfer (BS2000) durchgeführt werden.

Die Namen der Druckeraufrufe müssen den üblichen SINIX-Konventionen für Dateinamen entsprechen.

Es besteht die Möglichkeit, interaktive Druckerskripten zu erstellen. Damit können die Shell-Prozeduren variabel gestaltet werden und erst beim Ablauf durch den Benutzer mit konkreten Werten versorgt werden.
Der Steuerungsmechanismus benützt hierzu die Suffixe `.1` und `.s`.

Druckeraufrufe mit dem Suffix `.1` erlauben die Ausgabe von Abfragen in der 25. Zeile des Bildschirms.
Typischerweise wird ein derartiger Druckeraufruf für die Ausgabe von HIT-Dokumenten auf einen Drucker mit manueller Einzelblattzufuhr oder für die Ausgabe auf eine Schreibmaschine verwendet. (Siehe hierzu Kap. 7.1.1 Druckprogramm `filter`, Option `-a`, und Kap. 7.2.2.2 Schlüsselworte `BESTAET_NEUE_SEITE`, `REQUEST` und `DEL_REQUEST`).

Bei Druckeraufrufen mit dem Suffix `.s` steht der gesamte Bildschirm für Benutzereingaben zur Verfügung. Z.B. kann das Druckerskript so aufgebaut sein, daß der Benutzer seine Eingaben maskengesteuert betätigen kann. Nach Abarbeitung des Skriptes wird der Bildschirm automatisch wieder aufgebaut.

Die beiden genannten Arten von Druckeraufrufen laufen nicht im Hintergrund ab!

Sollen interaktive Druckerskripten von HIT-COL aus aufgerufen werden, müssen die Prozedurnamen in der Datei `/usr/lib/hit/scripts-D/hit_pif` eingetragen werden.

Das Kommando

```
trap "" 1 2 3 15
```

das am Anfang des Druckeraufrufs stehen sollte, verhindert, daß ein Druckauftrag eines Benutzers abgebrochen wird, wenn er z.B. seine Benutzerkennung verläßt oder die DEL -Taste gedrückt wird.

Die weiteren Programmaufrufe sind in den nachfolgenden Kapiteln beschrieben.

7.1.1 Druckprogramm filter

`filter` ist eine spezielle HIT-Komponente, die das übergebene Dokument für den Ausdruck aufbereitet. Zum Beispiel entfernt `filter`

- Zeilenlineale
- Steuerzeilen
- Absatzmarken
- geschützte Leerzeichen

Außerdem ersetzt er HIT-interne Steuerzeichen durch die entsprechenden Drucker-Steuerzeichen.

Syntax für `filter` (analog zur üblichen SINIX-Konvention):

filter -d *druckertabelle*[*-schalter*] *dokumentname*

druckertabelle	Name der Druckertabelle des angesprochenen Druckers (Pflichteingabe)
dokumentname	Name des auszudruckenden Dokuments. Im Druckeraufruf muß für *dokumentname* die Zeichenfolge `$1` stehen, über die der Name des Dokuments von HIT bzw. HIT-MENÜ als Variable übergeben wird.

Schalter

a	Neue Seite bestätigen. Vorgesehen für Drucker mit manueller Einzelblattzufuhr. Kann nicht verwendet werden bei der Funktion **Schreibauftrag** und wenn der Drucker über das SINIX-Spoolsystem betrieben wird.
b *zahl1,zahl2*	Heftrand; der Ausdruck wird bei ungeradzahligen Seiten um *zahl1*, bei geradzahligen um *zahl2* nach rechts verschoben. Ist nur *zahl1* angegeben, so gilt *zahl1* für alle Seiten. Standardwert: 0
B	Bei Texten mit dem Attribut **Durchgestrichen** (früheres Attribut **Breitschrift**) wird jedes zweite Zeichen (wenn es sich um ein geschütztes Leerzeichen handelt) beim Ausdruck unterdrückt, um breit geschriebene Texte korrekt formatieren zu können.
f	Seitenvorschub nach Ausdruck des Dokuments. Standardwert: nein
F *druform*	Das Dokument wird mit dem einseitigen Druckformular *druform* gemischt, d.h. jeder Seite wird das Formular unterlegt. Siehe auch Schalter `-m`.
l *zeilenzahl*	Zeilenzahl pro Seite (entspricht der Papierlänge). Wird die angegebene Zeilenzahl pro Seite überschritten so erfolgt nach *zeilenzahl* grundsätzlich ein Seitenvorschub. Fehlt der Schalter `-l`, wird der Wert von `SEITENLAENGE` aus der Druckertabelle verwendet.
L *zeilenlänge*	Anzahl der Zeichen pro Zeile; überzählige Zeichen werden abgeschnitten. Fehlt der Schalter `-L`, wird der Wert von `ZEILENLAENGE` aus der Druckertabelle verwendet.

m *doku*

Das auszudruckende Dokument wird mit dem Dokument *doku* zusammengemischt. Dabei ist folgendes zu beachten:
Steht in beiden Dokumenten Text, so bestehen keinerlei Einschränkungen. Attribute, Zeichensatz, Zeilenabstand und Zeichenbreite können in beiden Dokumenten beliebig, auch verschieden, gewählt werden. Dadurch kann es natürlich auch zu Überlappen oder Überschreiben von Text kommen.
Kommt in einem Dokument Grafik vor, wird die Ausgabe des anderen solange unterdrückt. Text neben Grafik und Grafik neben Grafik ist also nicht möglich. Es können nur zwei Dokumente gleichzeitig gemischt werden.

n

Bei Fehlermeldungen wird der Dokumentname `dokuname` als Name des auszudruckenden Dokuments eingesetzt. (Siehe Kap. 7.1.2: Stellungsparameter $5 bei der Übergabe an `lpr`).

p

Der geschützte Formulartext wird mit ausgedruckt.
Standardwert: nein

P *druckaufruf*

Aufbereitung von Zeichnungen erfolgt über *druckaufruf*. *druckaufruf* ist ein Druckprogrammaufruf aus der Graphik-Konfigurationsdatei `gra_liste`.
Existiert für das Graphik-Paket, mit dem eine Zeichnung erstellt wurde, kein Aufruf *druckaufruf*, so werden anstelle der Zeichnung Leerzeilen ausgegeben. Dasselbe passiert, wenn das Graphikpaket selbst nicht installiert ist.

r *j/n*

Das Dokument wird nach dem Ausdruck gelöscht (Angabe `j`) bzw. nicht gelöscht (Angabe `n`). `j/n` wird von HIT- bzw. HIT-MENÜ durch den Stellungsparameter `$3` übergeben (erforderlich, weil beim Ausdruck aus dem HIT nur eine temporäre Datei gedruckt wird, die anschließend wieder gelöscht werden muß).

s *nr1*[*nr2*] — Schachtnummer, aus der bei Einzelblattzuführung der Papiereinzug erfolgen soll. Mögliche Werte sind 1, 2 oder 3. Werden zwei Nummern hintereinander eingetragen, so erfolgt der Eintrag des ersten Blattes aus Schacht *nr1*; alle Folgeblätter werden aus Schacht *nr2* eingezogen.
Standardwert: 0

S *zahl1, zahl2* — Seitenbereich, der ausgedruckt werden soll. Ein # an Stelle von *zahl2* steht für "bis zum Ende des Dokuments". *zahl1, zahl2* wird durch den Stellungsparameter *$2* übergeben.

u *zeichenkette* — Bei einem evtl. notwendigen, erzwungenen Zeilenumbruch wird in der umgebrochenen Zeile zuerst die Zeichenfolge *zeichenkette* ausgegeben. Die maximale Länge von *zeichenkette* beträgt 15 Zeichen bzw. darf die maximale Zeilenlänge -10 nicht überschreiten. Damit haben immer mindestens 10 normale Druckzeichen in einer Druckzeile Platz.
zeichenkette darf beliebige Zeichen aus dem 7-Bit-ASCII-Zeichensatz enthalten.

7.1.2 SINIX-Kommando lpr

`filter` übergibt das aufbereitete Dokument über "pipe" an das SINIX-Druckprogramm `lpr`. Über die Bedeutung der hier standardmäßig gesetzten Schalter informiert Sie das Manual "Betriebssystem SINIX, Kommandos, Teil1".

Folgende Stellungsparameter werden über den Druckerauftrag dem `lpr` übergeben:

$4 Anzahl der auszudruckenden Dokumente
$5 Dokumentname
(notwendig für richtige Anzeige bei den Druckaufträgen)
$6 Priorität (wie von HIT-MENÜ übergeben)

Der Eintrag `> /dev/null` verhindert, daß nicht lesbare Fehlermeldungen am Bildschirm erscheinen.

Beachten Sie bitte:

Die `lpr`-Schalter `-font=` und `-pb[123]` dürfen nicht verwendet werden, da die Einstellung von Schriftart und Zeichenbreite in der Druckertabelle erfolgen muß (Schlüsselwort `INIT_DRUCKER`, siehe Kapitel 7.2.2.3).

7.1.3 Proportionalschrift-Formatierer proform

Für HIT-Dokumente, die in Proportionalschrift ausgedruckt werden sollen, muß ein spezieller Druckeraufruf erstellt werden, der den Aufruf des Proportionalschrift-Formatierprogramms `proform` enthält.

Der Aufruf von `proform` muß vor dem des Moduls `filter` erfolgen. Ein Druckeraufruf für Proportionalschrift hat demnach grundsätzlich folgendes Aussehen (Schalter und Argumente sind nicht berücksichtigt):

```
proform ¦ filter ¦ lpr
```

Syntax für `proform`

proform -d *drutab*[*-schalter*] [-a *datei2*] [*datei1*]

d *drutab* — Name der Druckertabelle. Sie muß Verweise auf die einzelnen Dicktentabellen enthalten.

e *datei1* — Angabe der Eingabedatei (Standardwert: `stdin`).
Die Eingabedatei muß eine Textdatei im HIT-Format sein.
Im Druckeraufruf muß für *datei1* die Zeichenfolge `$1` stehen, über die der Name des Dokuments von HIT bzw. HIT-MENÜ als Variable übergeben wird.

a *datei2* — Name der Ausgabedatei (Standardwert: `stdout`).

Schalter

E *errdat* — Angabe einer Fehlerdatei (Standardwert: `stderr`).
Falls Fehler oder Warnungen aufgetreten sind, dann werden diese Meldungen in die angegebene Fehlerdatei ausgegeben.

A — Falls Fehler oder Warnungen aufgetreten sind, dann werden diese Meldungen an das Dokument angehängt.

f *wert*

Setzen des Auffüll-Modus.
Entspricht der Eingabe einer Steuerzeile `Druckaufber.` mit der Option `fill+`.
Absätze, die durch die Formatierung kürzer geworden sind werden durch Leerzeilen aufgefüllt (nützlich bei Dokumenten mit fester Seitenlänge). Das numerische Argument gibt an, daß nur dann aufgefüllt werden soll, wenn eine Seite um mehr als *wert* Zeilen kürzer wurde.

n *dokuname*

Bei Fehlermeldungen wird der Dokumentname *dokuname* als Name des auszudruckenden Dokuments eingesetzt. Ansonsten wird der an `proform` übergebene Name, das ist i.d.R. der Name der temporären Arbeitsdatei, ausgegeben. (vgl. Schalter `-n` bei `filter`, Kap. 7.1.1).

t *zahl*

Setzen der Trennstellengewichtung.
Bei einer Formatierung kann es vorkommen, daß an einer Trennstelle umgebrochen werden muß. Durch viele Trennstellen in aufeinanderfolgenden Zeilen wird die Qualität des Absatzlayouts jedoch erheblich gemindert.
Über *zahl* kann das Umbrechen an Trennstellen beeinflußt werden. Je höher die Wertangabe, desto mehr Trennstriche werden beim Formatieren erzeugt.
Möglich sind die Werte `1` bis `10`. Bei Gewichtung `0` werden Trennstellen überhaupt nicht berücksichtigt; bei Gewichtung `10` alle Trennstellen.

S *zahl*

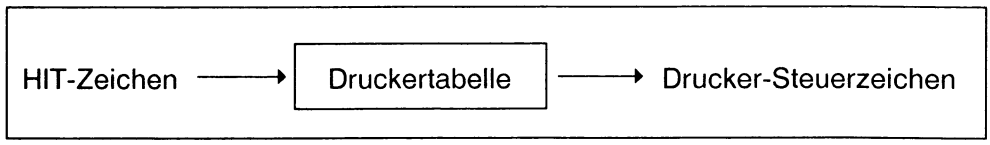
Setzt die Standard-Schriftart (Angabe in der Dicktentabelle).
zahl gibt die Schriftart an, auf die beim Umschalten in die Standard-Schriftart (Tabellenbereiche etc.) zurückgegriffen wird. Mögliche Werte: `1` bis `16`.

b *zahl*

Beginnt in der angegebenen Schriftart.
Dazu wird eine entsprechende Steuerzeile `Druckaufber.` vor die erste Zeile der Ausgabedatei geschrieben und auf die angegebene Schriftart umgeschaltet. Mögliche Werte: `1` bis `16`.

7.2 Druckertabellen

Die meisten Drucker unterscheiden sich grundlegend in der Art der Steuerzeichen, die zu ihrem Betrieb notwendig sind (z.B. Seitenvorschub, Umschalten auf Unterstreichungs-Modus usw.). Deshalb muß bei der Druckaufbereitung von HIT-Dokumenten berücksichtigt werden, auf welchem Drucker das Dokument ausgegeben werden soll.
Dies geschieht in Form von Druckertabellen, über die die HIT-internen Zeichen in Drucker-Steuerzeichen umgesetzt werden.



Selbst für einen bestimmten Drucker müssen die Druckertabellen in Abhängigkeit von seiner Ausbaustufe modifiziert werden, weil zum Beispiel eine Einzelblattzuführung andere Steuerzeichen erfordert als ein Endlosformular-Traktor.

Hinweis:

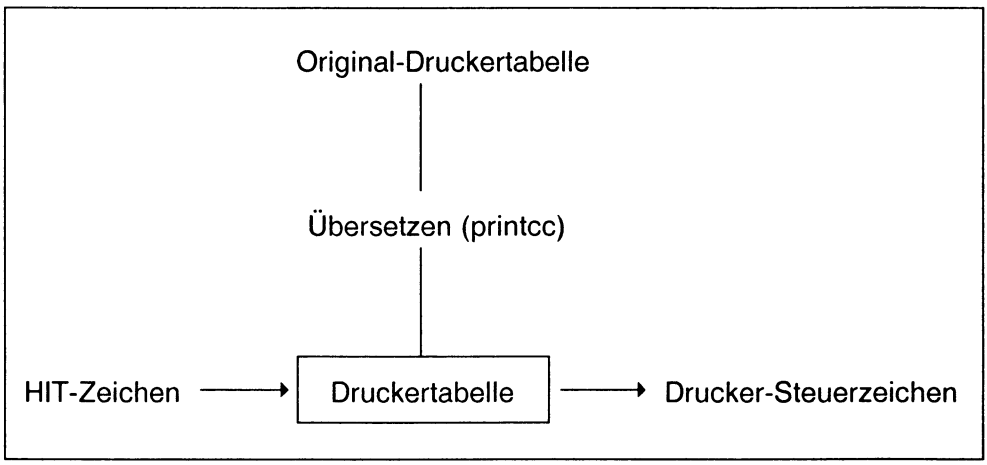
Zum Umgang mit Druckertabellen benötigen Sie gute bis sehr gute Shell- und Systemkenntnisse!

7.2.1 Wie erzeugt man eine Druckertabelle?

Um Ihnen die Anpassung eigener Druckertypen zu ermöglichen, können Sie eigene Druckertabellen erzeugen. Wie gehen Sie dabei vor?

- Mit HIT stellen Sie eine Tabelle auf, die jedem Zeichen ein entsprechendes Drucker-Steuerzeichen zuordnet (Original).
- Diese Tabelle muß, um sie in eine für das Programm `filter` verständliche, also ablauffähige Form zu bringen, übersetzt (compiliert) werden. Dies geschieht mit dem Printcap-Compiler `printcc`.

`printcc` erzeugt eine ablauffähige Druckertabelle, die Sie dann mit der Option `-d` an `filter` übergeben können.



7.2.1.1 Wo befinden sich die Druckertabellen?

Im HIT-System werden neben den übersetzten (ablauffähigen) Druckertabellen, die sich im Dateiverzeichnis `/usr/lib/hit/printcap` befinden, deren Originale für eine eventuelle Modifikation ebenfalls ausgeliefert. Die Original-Druckertabellen sind HIT-Dokumente und befinden sich im Dateiverzeichnis `/usr/lib/hit/original/printcap`; zur Unterscheidung von den übersetzten Druckertabellen sind ihre Namen durch das Suffix `.u` gekennzeichnet.

Beispiel

Zur Druckertabelle `pr9001`
gibt es die Original-Druckertabelle `pr9001.u`.

Die Namen der Druckertabellen setzen sich zusammen aus `pr` (für printcap), dem Druckertyp und evtl. einer Erweiterung (z.B. `e` für Endlos oder `-1` für Einzelblattzuführung, Schacht 1).

Zusätzlich zu den druckerspezifischen Druckertabellen gibt es noch eine Druckertabelle `standard.u`. Diese Tabelle können Sie für die Erstellung einer eigenen Druckertabelle verwenden; sie enthält alle verwendbaren Schlüsselworte und Zeichennamen und ist bereits mit den gebräuchlichsten Zeichen vorbelegt.

7.2.1.2 Wie wird eine Original-Druckertabelle bearbeitet?

Zum Bearbeiten einer Original-Druckertabelle gehen Sie wie folgt vor:

- **Melden Sie sich unter der Benutzerkennung `root` an.**
- **Wechseln Sie bei Anlagen, die über eine X/OPEN-Umgebung verfügen, mit dem Kommando `att` in diese Umgebung.**
- **Erweitern Sie den Suchpfad (Systemvariable `$PATH`) auf `/usr/bin/HIT`.**
- **Wechseln Sie mit dem Befehl `cd` in das Dateiverzeichnis `/usr/lib/hit/original/printcap`.**
- **Kopieren Sie eine der Tabellen mit dem Kommando cp unter einem neuen Namen und bearbeiten dann diese Kopie.**
  Dies sollten Sie unabhängig davon tun, ob Sie eine der vorhandenen druckerspezifischen Tabellen ändern oder aber für einen nicht vorgesehenen Drucker XYZ eine neue Tabelle erstellen wollen.

  Beispiel

  ```
  cp pr9022.u pr9022-12.u oder
  cp standard.u prXYZ.u
  ```

- **Rufen Sie die Druckertabelle mit dem Kommando `hit tabellenname` zur Bearbeitung mit dem HIT-Editor auf.**
- **Machen Sie die erforderlichen Einträge.**
  Die Steuerzeichen entnehmen Sie dem Technischen Handbuch für den betreffenden Drucker. Es handelt sich dabei um recht umfangreiche eigene Dokumentationen, die hier - auch in Ausschnitten - nicht wiedergegeben werden können.
- **Verlassen Sie nach Abschluß Ihrer Arbeit wie gewohnt den HIT und sichern die Original-Druckertabelle unter dem angegebenen Namen.**

Bitte beachten Sie:

Um einen Drucker, der von SINIX nicht unterstützt wird, betreiben zu können, müssen Sie dafür sorgen, daß ein entsprechendes "Backend" zur Verfügung steht und der Drucker auf Shell-Ebene konfiguriert wird.

7.2.1.3 Wie wird eine Druckertabelle übersetzt?

Die neu erstellte oder modifizierte Druckertabelle muß, um in eine vom Programm `filter` verständliche, also ablauffähige Form gebracht zu werden, noch übersetzt (compiliert) werden. Die Übersetzung geschieht mit Hilfe des Printcap-Compilers `printcc`.

Die Syntax für den Compiler lautet:

printcc *druckertabelle.u* [*-schalter*] -a *druckertabelle*

druckertabelle.u	ist die mit HIT erzeugte Tabelle
druckertabelle	ist die ablauffähige Druckertabelle

schalter

C	überprüft, ob die Tabelle falsch geschriebene Einträge enthält.
e *name*	leitet Fehlermeldungen in die Datei *name*. Standardmäßig werden Fehlermeldungen auf den Bildschirm (Standardausgabe stdout) ausgegeben.

Beachten Sie bitte:

- Der Aufruf muß im Dateiverzeichnis `/usr/lib/hit/original/printcap` erfolgen.
- Die übersetzte Druckertabelle wird im Dateiverzeichnis `/usr/lib/hit/printcap` abgelegt.

7.2.2 Aufbau einer Druckertabelle

Die Druckertabelle hat die Aufgabe, die Zuordnung von HIT-Zeichen (und bestimmten Steueranweisungen) zu Drucker-Steuerzeichen festzulegen. Die Zuordnungen erfolgen nach den für Tabellen geltenden Konventionen:

#name:wert#

Wobei *name* sein kann:

- Schlüsselwort oder
- Zeichenname (Zeichenname beginnt immer mit P_)

Beispiele

`US_EIN:`	für Einschalten des Unterstreichungs-Modus
`P_a:`	für a

Und *wert* entsprechend sein kann:

- einer der Schalter `JA` und `NEIN`,
- eine Zeichenkette (Buchstabe, Zahl, Steuerzeichen). Die Steuerzeichen kann man in verschiedener Schreibweise eingeben (Kapitel 7.2.2.3).

Beispiele

`#SIM_FETT:JA#`	bedeutet, daß die Simulation für Fettdruck eingeschaltet ist. Beim Drucker 9001 ist das notwendig, da er nicht über einen eigenen Fettdruck-Modus verfügt und deswegen Fettdruck eben nur simuliert werden kann.
`#LEERZ_V_SEITE:6#`	bedeutet, daß zu Beginn jeder Seite zunächst 6 Leerzeilen ausgegeben werden sollen.
`#P_DP:^03Ah#`	weist dem von HIT übergebenen Zeichen "Doppelpunkt" den Wert 3A in Sedezimal-Schreibweise zu.

7.2.2.1 Wie macht man Einträge?

Schlüsselworte mit JA/NEIN-Wertangabe

Soll die Funktion zur Wirkung kommen, tragen Sie `JA` ein; soll sie nicht zur Wirkung kommen, belassen Sie den Eintrag auf `NEIN` (oder löschen die gesamte Zeile).
Wird dem Schlüsselwort ein leerer Wert zugewiesen (kein Eintrag), dann gilt der Standardwert `JA`.

Beispiel

`#SIM_FETT:#`	bedeutet Fettdruck simulieren und ist identisch mit `SIM_FETT:JA#`

Schlüsselworte mit Zeichenkette-Wertangabe

Hier tragen Sie nach Bedarf einen Wert in einer der in 7.2.2.3 aufgelisteten Schreibweisen ein.

Schlüsselworte mit Zahl-Wertangabe

Hier tragen Sie nach Bedarf die gewünschte Zahl ein, z.B. `#SEITENLAENGE:65#`.

Zeichennamen mit Zeichenkette-Wertangabe

Über die Bedeutung der in der standard.u verwandten Zeichennamen informiert Sie die Tabelle im Anhang.
Den ASCII-Zeichennamen sind bereits Werte zugeordnet. Die hier zugeordneten Werte sind für nahezu alle Drucker gültig.

Bei Zeichennamen ordnen Sie dem von HIT übergebenen Zeichen einen druckerspezifischen Wert in einer der in 7.2.2.3 aufgelisteten Schreibweisen zu. Wenn die HIT-Komponente `filter` ein von HIT übergebenes Zeichen empfängt, zu dem in der Druckertabelle keine druckerspezifische Wertzuweisung erfolgen konnte, wird stattdessen ein Schmierzeichen ausgegeben.

7.2.2.2 Bedeutung der Schlüsselworte

Schlüsselworte mit JA/NEIN-Wertangabe

`SIM_FETT` simuliert den Fettdruck; sollten Sie verwenden, wenn der Drucker keinen echten Fettdruck kann (wie z.B. der Drucker 9001).

`SIM_US` simuliert die Unterstreichung.

`BESTAET_NEUE_SEITE` stoppt den Ausdruck am Seitenende; sollten Sie verwenden, wenn Ihr Drucker z.B. eine Schreibmaschine ist.
In diesem Fall muß der Name des Druckeraufrufs auf `.l` enden.

Beispiel: `lpneu.l`

Der Druckeraufruf (shell-Skript) darf nicht mit dem Zeichen & (Hintergrund) enden, da bei der Verarbeitung im Hintergrund keine Rückmeldung an den Benutzer möglich ist. Außerdem darf der Drucker nicht über das SINIX-Spoolsystem betrieben, sondern muß direkt angesteuert werden. Die Funktion kann auch nicht genutzt werden, um einen Schreibauftrag auszudrucken!

Schlüsselworte mit Zeichenkette-Wertangabe

Die folgenden Schlüsselworte betreffen die Attribute. Wenn Ihr Drucker eines der genannten Attribute nicht ausführen kann, können Sie z.B. "Ersatzattribute" zuweisen (andere Druckerfunktionen, z.B. Kleinschrift für das Attribut KURSIV beim 9013).

US_EIN	Unterstreichen ein
US_AUS	Unterstreichen aus
DG_EIN	Durchgestrichen ein
DG_AUS	Durchgestrichen aus
FETT_EIN	Fettdruck ein
FETT_AUS	Fettdruck aus
KUR_EIN	Kursivschrift ein
KUR_AUS	Kursivschrift aus
HOCH_EIN	Hochstellen ein
HOCH_AUS	Hochstellen aus
TIEF_EIN	Tiefstellen ein
TIEF_AUS	Tiefstellen aus

Die genannten Einträge sind gültig für alle Drucker, bei denen nur eine einzige Schriftart verwendet werden kann. Sind bei dem betreffenden Druckern mehrere Schriftarten möglich, so müssen Sie folgendes beachten:

Bei manchen Druckern können Attribute nur durch den Wechsel in eine andere Schriftart eingeschaltet werden.

Um nach dem Ausschalten des Attributs eine definierte Rückkehr in diejenige Schriftart zu ermöglichen, die vor dem Einschalten des Attributs gültig war, gibt es für jede eingetragene Schriftart (FONT01 ... FONT16, s.u.) einen eigenen Eintrag, um das jeweilige Attribut (z.B. Unterstreichen, Kursiv usw.) ein- und auszuschalten.

Für die Einträge gilt folgende Schreibweise:

schriftart_attribut_schalterstellung

schriftart, *attribut* und *schalterstellung* sind Variable und können jeweils einen der folgenden Werte annehmen:

schriftart — Hier ist die jeweilige Schriftart (`FONT01` ... `FONT16`) einzutragen.

attribut — Kurzbezeichnung für das Attribut. Mögliche Werte:

`US`	Unterstreichen
`DG`	Durchstreichen
`FETT`	Fettdruck
`KUR`	Kursivschrift
`HOCH`	Hochstellen
`TIEF`	Tiefstellen

schalterstellung — Möglich sind die Werte `EIN` bzw. `AUS`.

Beispiel:

`FONT01_US_EIN`	Unterstreichen ein (Schriftart 1)
`FONT01_US_AUS`	Unterstreichen aus (Schriftart 1)
.	
.	
.	
`FONT16_US_EIN`	Unterstreichen ein (Schriftart 16)
`FONT16_US_AUS`	Unterstreichen aus (Schriftart 16)

Die folgenden Schlüsselworte legen bestimmte Druckersteuerungen fest:

`SCHMIER` — definiert ein "Schmierzeichen" als ein Alternativzeichen für ein nichtdruckbares Zeichen.

`REQUEST:` *wert text* — fordert mit *wert text* ein neues Blatt an; zu verwenden bei Schreibmaschinen, wenn man das Schlüsselwort `BESTAET_NEUE_SEITE` verwandt hat. Sollte folgendes Aussehen haben:

```
^[[25;1H Bitte neue Seite einlegen und MENÜ druecken...
```

- Die Bildschirmsteuersequenz `^[[25;1H` positioniert die Schreibmarke auf die 1. Spalte in der 25. Zeile.
- Der kommentierende Text darf verändert werden.

`DEL_REQUEST` — löscht die Anforderung. Sollte folgendes Aussehen haben:

```
^[[25;1H^[[2K^[[24;1H
```

- Bildschirmsteuersequenz `^[[25;1H` wie oben.
- `^[[2K` löscht die Zeile.
- `^[[24;1H` positioniert die Schreibmarke auf die 1. Spalte in der 24. Zeile.

`FORMFEED` — Formfeed

`CR` — Carriage Return

`LINEFEED` — Linefeed

SCHACHT1	Steuerzeichen für Papiereinzug aus Standard-Schacht; zu verwenden bei Druckern mit mehreren Papierschächten.
SCHACHT2	Steuerzeichen für Schacht 2
SCHACHT3	Steuerzeichen für Schacht 3
UNTER	Code für Unterstrich, falls SIM_US eingestellt ist.
SPACE	Leerzeichen

Mit den folgenden Initialisierungsmöglichkeiten stellen Sie den Drucker auf verschiedene Modi ein.

INIT_DRUCKER	Initialisierung vor dem Ausdruck der 1. Seite. Zeichensatz einstellen, ggf. Parameter rücksetzen.
END_DRUCKER	Rücksetzen des Druckers nach dem Ausdruck der letzten Seite
VOR_GRAPHIK	Steuersequenzen, die vor dem Druck einer Grafik-Datei ausgegeben werden sollen.
NACH_GRAPHIK	Ausschalten des Grafik-Modus (falls nötig), ggf. auch Teilsequenzen aus INIT_DRUCKER

Wichtiger Hinweis:

Die Steuerzeilen `Zeichenbreite` und `Schriftart` sowie Attribute können nur dann ohne Auftreten von Fehlern verwendet werden, wenn die Einstellungen ausschließlich über die Druckertabelle erfolgen und nicht über die Schalter beim Kommando `lpr`.

Um bei der Verwendung der genannten Funktionen die Rückkehr auf einen definierten Stand zu ermöglichen, müssen die Einträge für die Standard-Schriftart, die Standard-Zeichenbreite und evtl. das Ausschalten von Attributen auch in INIT_DRUCKER und NACH_GRAPHIK erfolgen.

Die folgenden Schlüsselworte betreffen die möglichen Zeilenabstände und Format-Anweisungen. Die Zeilenabstände müssen natürlich "druckergerecht" sein. Der Standard-Abstand ist normalerweise 6 Zeilen pro Zoll (entspricht 70 Zeilen pro DIN A4-Blatt).

ZEILABST0.25	Zeilenabstand 0.25
ZEILABST0.5	Zeilenabstand 0.5 (Halbzeile)
ZEILABST0.75	Zeilenabstand 0.75
ZEILABST1.0	Zeilenabstand 1.0
ZEILABST1.5	Zeilenabstand 1.5
ZEILABST2.0	Zeilenabstand 2.0
ZEILABST2.5	Zeilenabstand 2.5
ZEILABST3.0	Zeilenabstand 3.0
ZEILABST3.5	Zeilenabstand 3.5

Die folgenden Schlüsselworte legen die Zuordnung der möglichen Schriftarten (eingestellt mit der Steuerzeile `Schriftart`) fest. Einträge sind nur sinnvoll, wenn am Drucker über Steuerzeichen verschiedene Schriftarten eingestellt werden können.

```
FONT01
.
.
.
FONT16
```

Mit den folgenden Schlüsselworten vergeben Sie Namen für die mit `FONT01` bis `FONT16` angesprochenen Schriftarten. HIT-Anwender brauchen dann in der Steuerzeile `Schriftart` nicht mehr die Nummer der gewünschten Schriftart einzutragen, sondern können die von Ihnen vergebenen Namen verwenden.

```
FONTNAME01
.
.
.
FONTNAME16
```

Sie können für eine Schriftart auch mehrere Bezeichnungen vergeben, die Sie dann durch Doppelpunkte trennen.

Beispiel:

```
#FONTNAME01:Courier:Courier 10:Courier 10 Portrait#
```

Mit den folgenden Schlüsselworten erfolgt die Zuordnung der mit der Steuerzeile `Zeichenbreite` eingetragenen Werte zu den entsprechenden Druckerfunktionen.
Voraussetzung dafür ist, daß am Drucker durch Steuerzeichen verschiedene Zeichenbreiten eingestellt werden können (z.B. bei Matrixdruckern).

Mögliche Einträge:

`CPI10`	entspricht 10 Zeichen/Zoll
`CPI12`	entspricht 12 Zeichen/Zoll
`CPI15`	entspricht 15 Zeichen/Zoll

Schlüsselworte mit Zahl-Wertangabe

`LEERZ_V_SEITE`	Leerzeilen am Anfang einer Seite.
`BLANKS_V_ZEILE`	Leerzeichen vor einer Zeile.
`SEITENLAENGE`	Seitenlänge in Zeilen (bei Zeilenabstand 1.0)
`ZEILENLAENGE`	Zeilenlänge in Zeichen.

Die folgenden Einträge sind nur für eine Druckaufbereitung in Proportionalschrift maßgebend. Beschreibung im Kapitel 7.3.

```
NR_STANDARDFONT
HMI_EINHEIT
HMI_MINIMALWERT
HMI_MAXIMALWERT
FORMATSTRING
FONTCAP01
.
.
.
FONTCAP16
```

7.2.2.3 Welche Schreibweisen kann man verwenden?

Die Wertzuweisungen können in vier verschiedenen Schreibweisen erfolgen.

- ASCII-Schreibweise
  Beispiel `#P_A:A#`
- dezimale Schreibweise
  Beispiel `#P_A:^065z#`
- hexadezimale Schreibweise
  Beispiel `#P_A:^041h#`
- oktale Schreibweise
  Beispiel `#P_A:^0101o#`

In einer Tabelle dürfen mehrere Schreibweisen gemischt sein.

ASCII-Schreibweise

Bei der ASCII-Schreibweise setzen Sie einfach nach *name:* das entsprechende ASCII-Zeichen ein.

Beispiel

```
#P_A:A#
```

CTRL stellen Sie durch dieses Zeichen dar: ^
ESC wird entsprechend so dargestellt: ^[

Dezimale Schreibweise

Syntax

^0*dezimaler Zahlenwert***z**

Die dezimale Schreibweise wird durch diese Zeichen eingeleitet: ^0 und mit diesem Zeichen beendet: z (das z steht für Zehnersystem).

Beispiel

```
#P_A:^065z#
```

Sedezimale Schreibweise

Syntax

^**0***Sedezimaler Zahlenwert***h**

Die Sedezimale Schreibweise wird durch diese Zeichen eingeleitet: ^0 und mit diesem Zeichen beendet: h.

Beispiel

```
#P_A:^041h#
```

Oktale Schreibweise

Syntax

^**0***oktaler Zahlenwert***o**

Die oktale Schreibweise wird durch diese Zeichen eingeleitet: ^0 und mit diesem Zeichen beendet: o.

Beispiel

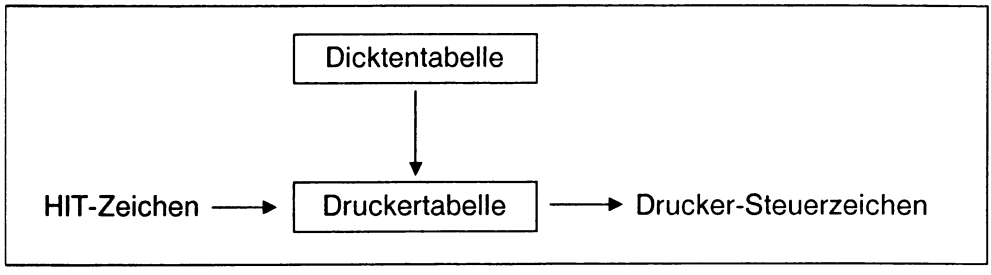
```
#P_A:^0101o#
```

7.3 Dicktentabellen

Bei der Druckaufbereitung in Proportionalschrift werden alle Absätze eines Dokuments neu formatiert und dabei bei jedem Zeichen seine Breite (die sog. "Dicktenweite") berücksichtigt (siehe Kapitel 6.4 im Benutzerhandbuch).

Dazu muß der Proportional-Formatierer jedoch wissen, wie breit die einzelnen Zeichen sind. Diese Informationen entnimmt er **Dicktentabellen** (fontcap's), die für jede verwendete Schriftart zur Verfügung stehen müssen.

Die Umsetzung in Drucker-Steuerzeichen erfolgt dann über die Druckertabelle. Die Druckausgabe kann somit - analog den Druckertabellen - so dargestellt werden:



Damit bei der Druckausgabe die Dicktenweiten berücksichtigt werden können, sind zusätzliche Eintragungen in den Druckertabellen nötig (siehe Kapitel 7.3.3).

7.3.1 Wie erzeugt man eine Dicktentabelle?

Um Ihnen die Anpassung an unterschiedliche Proportional-Schriftarten zur ermöglichen, können Sie eigene Dicktentabellen erzeugen. Wie gehen Sie dabei vor?

- Mit HIT erstellen Sie eine Tabelle, die jedem Zeichen eine entsprechende Zeichenbreite zuordnet.
- Die Tabelle muß, um sie in ein für die Programme `proform` und `filter` verständliche, also ablauffähige Form zu bringen, übersetzt ("compiliert") werden. Dies geschieht mit dem "Fontcap-Compiler" `fontcc`.

`fontcc` erzeugt eine ablauffähige Dicktentabelle, deren Namen Sie dann in die entsprechende Druckertabelle eintragen können.

7.3.1.1 Wo befinden sich die Dicktentabellen?

Wegen der Vielzahl von möglichen Proportionalschriften, die darüber hinaus von den verschiedenen Druckern nicht standardmäßig unterstützt werden, wird mit dem HIT-System nur eine Beispiel-Dicktentabelle (Original) ausgeliefert. Original-Dicktentabellen sind HIT-Dokumente und befinden sich im Dateiverzeichnis `/usr/lib/hit/original/fontcap`. Zur Unterscheidung von den übersetzen Dicktentabellen, die im Dateiverzeichnis `/usr/lib/hit/fontcap` abgelegt werden müssen, sind ihre Namen durch das Suffix `.u` gekenn-zeichnet.

Beispiel

Zur Dicktentabelle `fc9004-1` gibt es die Original-Dicktentabelle `fc9004-1.u`.

Die Namen der Dicktentabellen setzen sich aus `fc` (für fontcap), dem Druckertyp und einer Erweiterung zur Kennzeichnung der Schriftart zusammen.

7.3.1.2 Wie wird eine Original-Dicktentabelle bearbeitet?

Zum Bearbeiten einer Original-Dicktentabelle gehen Sie wie folgt vor:

- **Melden Sie sich unter der Benutzerkennung `root` an.**
- **Wechseln Sie bei Anlagen, die über eine X/OPEN-Umgebung verfügen, mit dem Kommando `att` in diese Umgebung.**
- **Erweitern Sie den Suchpfad (Systemvariable `$PATH`) auf `/usr/bin/HIT`.**
- **Wechseln Sie mit dem Befehl `cd` in das Dateiverzeichnis `/usr/lib/hit/original/fontcap`.**
- **Kopieren Sie die Beispiel-Dicktentabelle und vergeben Sie dabei die von Ihnen gewünschte Bezeichnung.**
- **Rufen Sie die Dicktentabelle mit dem Kommando `hit dicktentabelle.u` zur Bearbeitung mit dem HIT-Editor auf.**

Nun können Sie die erforderlichen Einträge machen. Die Dicktenweiten der einzelnen Zeichen entnehmen Sie dabei bitte der Dokumentation für die betreffende Proportional-Schriftart.

7.3.1.3 Wie wird eine Dicktentabelle übersetzt?

Die neu erstellte oder modifizierte Dicktentabelle muß, um in eine für die Programme `proform` und `filter` verständliche Form gebracht zu werden, noch übersetzt ("compiliert") werden. Die Übersetzung geschieht mit Hilfe des "Fontcap-Compilers" `fontcc`.

Syntax für den Compiler:

fontcc *dicktentabelle.u* -a *dicktentabelle*

dicktentabelle.u ist die mit HIT erzeugte Original-Dicktentabelle.

dicktentabelle ist die ablauffähige Dicktentabelle

Beachten Sie bitte:

- Der Aufruf muß im Dateiverzeichnis `/usr/lib/hit/original/fontcap` erfolgen.
- Die übersetzte Dicktentabelle wird im Dateiverzeichnis `/usr/lib/hit/fontcap` abgelegt.

7.3.2 Aufbau einer Dicktentabelle

Die Dicktentabelle hat die Aufgabe, die Breite der einzelnen Zeichen festzulegen.

Die Einträge erfolgen nach den für HIT-Tabellen geltenden Konventionen:

#name:wert#

Wobei *name* sein kann:

- Zeichenname (Zeichenname beginnt immer im `F_`) oder
- Schlüsselwort

Beispiele

`F_A`	für A
`CHARUNIT`	Schrittweiten-Teilung

Und *wert* entsprechend sein kann:

- eine Zahl
- die Kurzbezeichnung für ein Attribut

Während als *wert* für die Schlüsselwörter `CHARUNIT` und `MITTL_BREITE` nur eine Zahl zulässig ist, können bei den Zeichennamen auch mehrere Werte, die Sie durch Doppelpunkte trennen, eingetragen werden.

Beispiel

```
#F_A:37:39:32#
```

Der zweite und alle folgenden Werte geben dabei die Dicktenweiten für attributierte Zeichen an. Die Dicktenweite kann nämlich bei Verwendung z.B. der Attribute **Kursiv** und **Fettdruck** durchaus andere Werte annehmen.

Die Zuordnung der mehrfachen Werte zu den einzelnen Attributen erfolgt über das Schlüsselwort `ATTRIBUTFONT`, bei dem für *wert* - getrennt durch Doppelpunkte - die Kurzbezeichnungen der verwendeten Attribute einzutragen sind.

Beispiel

```
#ATTRIBUTFONTS:FE:KU#
#F_A:37:39:32#
```

Das Zeichen A hat in Normalschrift die Dicktenweite 37, in Fettdruck 39 und in Kursivschrift 32.

Bedeutung der Schlüsselworte

CHARUNIT	Anzugeben ist eine Einheit in 1/Zoll, in der die folgenden Zeichenbreiten gemessen werden; z.B. CHARUNIT:100, F_n:10 bedeutet, daß "n" die Breite 10/100 Zoll besitzt.
MITTL_BREITE	mittlere Zeichenbreite Durch Änderung des angegebenen Wertes kann der Zeilenumbruch bei Proportionaldruck beeinflußt werden: Verkleinerung des Wertes bewirkt, daß mehr Zeichen in eine Zeile passen; Vergrößerung, daß der Zeilenumbruch früher erfolgt. Entsprechend kann sich die Länge der einzelnen Absätze verändern. Der genaue Wert muß durch Tests ermittelt werden! Die Angabe der mittleren Zeichenbreite ist notwendig, da Proportionalschrift-Zeichen im Mittel schmäler sind als dicktengleiche Zeichen (Bildschirm) und Zeilen dadurch beim Druck kürzer werden. Beachten Sie, daß bei Änderung von MITTL_BREITE auch die Breite der Randbereiche differiert. Wenn in einem proportional-formatierten Dokument Semigrafik-Zeichen verwendet werden, dann muß die mittlere Zeichenbreite der Breite der Semigrafik-Zeichen entsprechen.

`ATTRIBUTFONTS` Hier legen Sie die Reihenfolge der Einträge für attributierte Zeichen fest. Mögliche Werte sind:

`FE`	Fettdruck
`DG`	Durchgestrichen
`US`	Unterstrichen
`KU`	Kursiv
`TI`	Tiefstellen
`HO`	Hochstellen

Über die Bedeutung der Zeichennamen informiert Sie die Tabelle im Anhang.

Die einzelnen Attributnamen können miteinander kombiniert werden. Dazu werden die zu kombinierenden Werte mit dem Zeichen _ verbunden.

Beispiel

```
#ATTRIBUTFONTS:KU:DG:KU_DG#
```

7.3.3 Zusätzliche Einträge in der Druckertabelle

Für die Druckaufbereitung in Proportionalschrift sind neben den Dicktentabellen auch noch zusätzliche Einträge in den Druckertabellen notwendig. Die möglichen Einträge lauten:

`FONTCAP01`
.
.
.
`FONTCAP16`
	Hier ist der Name der Dicktentabelle für die unter `FONT01` ... `FONT16` definierte Schriftart einzutragen.

`NR_STANDARDFONT`	Die Nummer derjenigen Schriftart, in die umgeschaltet wird, wenn in der Steuerzeile `Druckaufber.` mit den Anweisungen `stop+` in die Standard-Schriftart gewechselt wird.
`HMI_EINHEIT`	Anzugeben ist der Kehrwert der Schrittweite des Druckers, d.h. die Zahl der Teilschritte, die zusammengenommen eine horizontale Positionierung des Druckers um 1 Zoll ergeben.
`HMI_MINIMALWERT`	Kleinstmöglicher Positionierungs-Schritt.
`HMI_MAXIMALWERT`	Größtmöglicher Positionierungs-Schritt.
`FORMATSTRING`	Beschreibt die Steuersequenz, die zum Erzeugen eines horizontalen Positionierungs-Schritts an den Drucker geschickt wird (erforderlich für Wortzwischenräume). Der mögliche Aufbau der Steuersequenz wird anschließend ausführlich erläutert.

Formatstring

Die Steuersequenz, die einen horizontalen Positionierungs-Schritt erzeugt, hat i.a. folgende Form:

string1 varwert [*string2*]

string1,2 ist dabei eine feste Zeichenfolge, an der der Drucker erkennt, um welche Steueranweisung es sich handelt. Die festen Zeichenfolgen geben Sie nach den in Kapiteln 7.2.2.3 beschriebenen Konventionen an (z.B. `A` oder `^041h` usw.).

varwert ist der eigentliche Vorschubswert, der vom Druckprogramm `filter` variiert wird. Um die Ausgabe variabler Werte zu ermöglichen, muß *varwert* in einer speziellen Syntax eingetragen werden:

Syntax für varwert

% [*wert*] [*byte*] *format* [(*operator zahl...*)]

Das einleitende %-Zeichen muß immer gesetzt werden. Es dürfen auch mehrere *varwert*-Folgen - jeweils mit `%` beginnend - in `FORMATSTRING` eingetragen werden.

Ebenfalls Pflichteintrag ist *format*. Hier legen Sie fest, in welcher Form der variable Vorschubswert an den Drucker übergeben wird:

- d dezimal
- b binär
- r als Wiederholung des Zeichens *wert*.

Alle anderen Einträge sind optional.

Beim Format `d` legt *wert* fest, daß der Vorschubswert *wert*-stellig ausgegeben wird (eventuell mit führenden Nullen).
Beim Format `r` wird das Zeichen *wert* (standard: 0x00) so oft ausgegeben, bis der nötige Vorschub erreicht ist. Dadurch ist eine Simulation der horizontalen Positionierung möglich (anwendbar z.B. im Grafikmodus).

Standardmäßig werden im Binärformat zwei Byte ausgegeben (zuerst das höherwertige, dann das niederwertige). Über *byte* können Sie diese Ausgabe steuern.

- < nur höherwertiges Byte
- > nur niederwertiges Byte.

Sind der vom Proportionalschrift-Formatierer berechneten Vorschubswert und der vom Drucker erwartete Wert nicht gleich (z.B. wenn sich die vom Drucker erwarteten Werte nicht durch einfache Multiplikation berechnen lassen - vgl. `ZEILABST` in `pr9004.u`), so kann mittels *operator zahl* eine oder mehrere Umrechnungen vorgenommen werden, z.B. `(+1)`. Folgende Operatoren sind möglich:

- \+ SUMME
- – Differenz
- & UND-Verknüpfung
- ¦ ODER-Verknüpfung
- ^ ENTWEDER-ODER-Verknüpfung.

Achtung:

- Für *wert* und *zahl* sind nur dezimale, hexadezimale und oktale Schreibweise möglich und zwar abweichend von sonstigen Konventionen ohne einleitendes ^ (z.B. `051z` oder `033h` oder `063o`).

- Soll das Zeichen `%` als solches auftreten, so muß es doppelt geschrieben werden, d.h. `%%`.

Beispiel: Formatstring für den Drucker 9022

```
#FORMATSTRING:^01ch&c%>b%<b#
```

Für jede horizontale Positionierung erhälte der Drucker also folgende Sequenz:

- das Steuerzeichen 'FS' und die Zeichen '&' und 'c'
- das niederwertige Byte des Vorschubwertes
- das höherwertige Byte des Vorschubwertes

Beispiel: Formatstring für den HP-Laserjet

```
#FORMATSTRING:^01bh*p+%dX#
```

Für jede horizontale Positionierung erhält der Drucker folgende Sequenz:

- das Steuerzeichen 'ESC' und die Zeichen '*' und 'p'
- die relative horizontale Positionierung in $1/300$-Zoll-Schritten in folgender Form: *+zahl*, wobei *zahl* in der ASCII-Darstellung ausgegeben wird.
- das Zeichen X, das die Sequenz abschließt.

8 Konfiguration

Dieses Kapitel gibt Ihnen Hinweise, wie Sie HIT an spezielle Anforderungen anpassen können. Falls erforderlich, können Sie

- Benutzerkennungen so modifizieren, daß Benutzer gemeinsam mit den gleichen Archiven, Ordnern und Dokumenten arbeiten (Kapitel 8.1);
- Zentrale Archive einrichten, auf die alle Benutzer zusätzlich zu ihren eigenen zugreifen können (Kapitel 8.2);
- die Funktionsauswahl in HIT-MENÜ erweitern (Kapitel 8.3);
- für einzelne oder alle Benutzer den Zeitraum festlegen, nach dem Objekte im elektronischen Papierkorb von HIT-MENÜ gelöscht werden sollen (Kapitel 8.4);
- zusätzliche Programme installieren, mit denen über die Funktion **Markieren und Bearbeiten** HIT-Texte bearbeitet werden können (Kapitel 8.6);
- die Liste von Programmen erweitern, die über die Steuerzeile `Anwendung` angeboten werden (Kapitel 8.7).
- für einzelne oder alle Benutzer Posteingangs-Meldungen festlegen bzw. die Postfunktion erweitern.

Wichtige Hinweise

- Die oben erwähnten Anpassungen müssen zum Teil auf Shell-Ebene durchgeführt werden. Dazu sind gute bis sehr gute SINIX-Kenntnisse dringend erforderlich!
- Wenn in den folgenden Kapiteln von Shell-Variablen, deren Einrichtung und Verwendung die Rede ist, so wird hier auf das Handbuch "Betriebssystem SINIX, Buch 1" verwiesen. Falls Sie eine Shell-Variable global (d.h. für alle Benutzer) vereinbaren wollen, sollten Sie die System-Datei `/etc/profile` um den entsprechenden Eintrag erweitern.
- Bei allen Anlagen mit SINIX V5 befinden sich die HIT-Systemdateien im X/OPEN-Universum. Ein Aufruf des HIT-Editors auf Shell-Ebene kann nur erfolgen, wenn die Shell-Variable `$PATH` entsprechend vorbelegt ist (Kapitel 11 "Systemarchitektur").

8.1 Mehrfachbenutzbarkeit

Jeder HIT-Benutzer verfügt normalerweise über seine eigenen Archive, Ordner und Dokumente, die anderen Benutzern nicht zugänglich sind. Oft jedoch macht es der Organisations-Ablauf erforderlich, daß mehrere Benutzer mit den gleichen Archiven, Ordnern und Dokumenten arbeiten. Um dies auch in HIT zu erreichen, gehen Sie folgendermaßen vor (unbedingt Reihenfolge beachten!):

- **Richten Sie über das Standard-Menüsystem für die Benutzer, die gemeinsam Zugriff auf die Dokumente haben sollen, eine neue Benutzergruppe ein.**

- **Richten Sie anschließend (ebenfalls über das Standard-Menüsystem) die benötigten Benutzerkennungen ein.**
  Als Benutzergruppe müssen Sie dabei jeweils den Namen eintragen, den Sie vorher bei der Definition der neuen Benutzergruppe gewählt haben.

- **Legen Sie nun auf Shell-Ebene im HOME-Dateiverzeichnis jedes Benutzers dieser Gruppe eine Datei `.profile` an.**
  Diese muß folgende Anweisungen enthalten:
  - Änderung des HOME-Dateiverzeichnisses (für alle Benutzer den gleichen Eintrag!)
  - Das Kommando `cd`, um den Wechsel in das neue HOME-Dateiverzeichnis zu vollziehen.
  - Änderung der Standard-Einstellung der Zugriffsrechte mit dem Kommando `/bin/umask` (z.B. `/bin/umask 002`). Die Verwendung des Kommandos `/bin/umask` ist im Handbuch "Betriebssystem SINIX, Buch 1" beschrieben.

Beispiel

```
HOME=/usr/alle
export HOME
cd
/bin/umask 002
```

Bitte beachten Sie:

- Ein Anmelden von mehreren Benutzern unter derselben Benutzerkennung darf nicht erfolgen, da dies aufgrund der beschränkten Anzahl von Prozessen je Benutzerkennung zu Fehlern führen würde!

- Wenn eine bestimmte Benutzerkennung nachträglich einer gemeinsamen Gruppe zugeordnet werden soll, muß für alle bereits existierenden Dateien und Dateiverzeichnisse (auch das HOME-Dateiverzeichnis selbst!) mit dem Kommando `/bin/chgrp` die Gruppe geändert werden.

  Dateien und Dateiverzeichnisse, die unter dem "alten" HOME-Dateiverzeichnis angelegt wurden und weiterhin benötigt werden, müssen bei einer Änderung des HOME-Dateiverzeichnisses in dieses übertragen werden.

8.2 Zentrale Archive

Zentrale Archive sind Archive, auf deren Dokumente alle Benutzer des Systems zugreifen können. Im Gegensatz zu der in Kapitel 8.1 beschriebenen Mehrfachbenutzbarkeit, bei der alle Benutzer gemeinsam mit denselben Archiven arbeiten, stehen die Zentralen Archive den Benutzern zusätzlich zu ihren eigenen Archiven zur Verfügung.

Wozu dienen Zentrale Archive?

Die Zentralen Archive erleichtern Ihnen als Systemverwalter die Verwaltung und Betreuung von Dokumenten (z.B. Bausteinen oder Formularen), auf die alle Benutzer Zugriff haben sollen. Eine Änderung oder Ergänzung in einem Zentralen Archiv wird für alle gleichzeitig wirksam.

Wie richten Sie Zentrale Archive ein?

Am einfachsten ist es, wenn Sie für die Verwaltung der Zentralen Archive eine eigene Benutzerkennung einrichten. Das Anlegen und Verwalten von Zentralen Archiven, Ordnern und Dokumenten kann dann über HIT-MENÜ erfolgen.
Falls Sie sich für irgend ein anderes Dateiverzeichnis (zum Beispiel `/usr/lib/hit/Zentrale`) entscheiden, müssen Sie selbst dafür sorgen, daß die entsprechende Archivstruktur (beschrieben in Kapitel 11) angelegt wird.

Achten Sie darauf, daß alle von Ihnen angelegten Archive, Ordner und Dokumente die richtigen Zugriffsrechte besitzen, d.h. daß sie für die Benutzer mindestens lesbar sind (wenn Sie keine Änderungen durch die Benutzer zulassen wollen). Erreichen können Sie dies entweder über die Funktion **Zugriff** in HIT-MENÜ oder das SINIX-Kommando `/bin/chmod` auf Shell-Ebene.

Wie machen Sie Zentrale Archive den Benutzern zugänglich?

Um die eingerichteten Zentralen Archive den Benutzern zugänglich zu machen, sind zwei Schritte erforderlich:

- **Belegen Sie für jeden Benutzer die Shell-Variable `FITCENTRE` mit dem Namen des Dateiverzeichnisses, in dem sich die Zentralen Archive befinden.**

Beispiel

```
FITCENTRE=/usr/alle
export FITCENTRE
```

- **Teilen Sie anschließend den einzelnen Benutzern den oder die Namen der Zentralen Archive mit, mit denen sie arbeiten sollen.**

Wie arbeitet ein Benutzer mit den Zentralen Archiven?

Ein Benutzer kann nun ganz einfach auf ein Zentrales Archiv zugreifen, indem er in seiner eigenen Archivstruktur ein leeres Archiv mit dem von Ihnen vorgegebenen Namen anlegt. Sobald das Archiv angelegt bzw. im Kontrollbereich eingestellt ist, hat er sofort Zugang zu den darin enthaltenen Ordnern und Dokumenten.

Bitte beachten Sie:

Die Shell-Variable für Zentrale Archive wurde für HIT-COL V4.0 umbenannt in `FITCENTER`. Bei Parallelinstallation von HIT V4.0 und HITCOL V4.0 können getrennte Zentrale Archive angelegt werden indem Sie die Variablen `FITCENTRE` und `FITCENTER` belegen.

8.3 Zusätzliche Funktionen in HIT-MENÜ

Wie in Kapitel 4.2.4.15 des Benutzerhandbuchs bereits erwähnt, besteht die Möglichkeit, die Funktionsauswahl in HIT-MENÜ auf Dokumentebene zu erweitern.

Sie können HIT-MENÜ zu einer richtigen Büro-Umgebung erweitern, indem Sie Funktionen wie zum Beispiel Tabellenkalkulation, Terminkalender oder Tischrechner anschließen. Die Vereinbarung von neuen Funktionen erfolgt dabei über die Shell-Variablen `FITUSR1`, `FITUSR2` ... `FITUSR32`.
Die Funktionsauswahl kann also um insgesamt 32 Anwendungen erweitert werden.

Die Variablen `FITUSR25` bis `FITUSR32` werden für die Systemkonfiguration herangezogen und sollten deshalb nicht überschrieben werden.

Die Belegung der Shell-Variablen erfolgt nach folgender Syntax:

FITUSR*zahl* = "[%USROPT: *opt*] *name aufruf*"

FITUSR*zahl* — ist der Name der Shell-Variablen, wobei Sie für *zahl* eine Zahl zwischen 1 und 24 einsetzen müssen.

%USROPT: — ist der Name einer Option, mit der Sie HIT-MENÜ zusätzliche Steueranweisungen übergeben können. %USROPT: muß durch eine oder mehrere der folgenden Optionen ergänzt werden:

- B Der HIT-MENÜ-Bildschirm wird mit blinkender Sanduhr angezeigt, während das Kommando ausgeführt wird. Die Option ist für Anwendungen gedacht, die als Hintergrund-Prozeß ablaufen und keine Ein-/Ausgabe vornehmen.
- R Nach Beendigung des Kommandos wird der HIT-MENÜ-Bildschirm wieder aufgebaut.
- S Vor der Programmausführung kann der Benutzer einen Dokumentnamen eingeben bzw. die Dokumentnamen wie gewohnt im Arbeitsbereich markieren. Beim Aufruf des Kommandos werden die ausgewählten Dokumente als Parameter angehängt.

name ist der Text, der nach Drücken der Taste MENU **Funktionsauswahl** in HIT-MENÜ auf Dokumentebene erscheint. *name* darf nicht länger als 14 Zeichen sein und nicht den gleichen Wortlaut haben wie eine bereits vorhandene Funktion!

aufruf ist das Programm, das nach Auswahl von *name* ausgeführt werden soll. Nach Möglichkeit sollte der absolute Pfadname verwendet werden. Ergänzt werden kann *aufruf* noch um die Option `%s`. Ist diese gesetzt, wird das im Kontrollbereich von HIT-MENÜ eingestellte Dokument dem angesprochenen Anwenderprogramm übergeben. Bei Verwendung der Option %USROPT: darf `%s` nicht eingesetzt werden!

Beispiel

In einer Datei `.profile` sind folgende, zusätzliche Funktionen definiert:

```
FITUSR1="%USROPT: SR Siplan /usr/bin/siplan"
FITUSR2="CED /usr/bin/ced %s"

export FITUSR1 FITUSR2
```

Beim Aufruf der Funktion **Siplan** erhält der Benutzer zunächst ein Eingabefeld für den Dokumentnamen im Arbeitsbereich. Über Suchmuster bzw. F19 **Objektauswahl** können ein oder mehrere Dokumente markiert werden. Nach dem Beenden von SIPLAN wird der Bildschirm von HIT-MENÜ wieder aufgebaut.

Die Funktion **CED** startet den Editor `ced` mit dem im Kontrollbereich eingestellten Dokument (ohne Auswahlmöglichkeit).

Bitte beachten Sie:

Der Name des auszuführenden Programms muß zusätzlich in der HIT-Konfigurations-Datei `hit_pif` eingetragen werden, um Bildschirmausgaben (z.B. Fehlermeldungen) zu ermöglichen.

8.4 Verfallszeit im Papierkorb

Alle Objekte (Archive, Ordner, Dokumente), die sich länger als 24 Stunden im Papierkorb jedes HIT-Benutzers befinden, werden beim Aufruf von HIT automatisch gelöscht.

Sie können diese Verfallszeit nach Bedarf ändern, indem Sie die Shell-Variable `FITTRASH` mit einem anderen Wert (in Stunden) belegen.

Beispiel

```
FITTRASH=48
```

bewirkt, daß der Inhalt des Papierkorbs erst nach 48 Stunden gelöscht wird.

8.5 Benutzerspezifische Druckeraufrufe

Oft sind an einer Anlage mehrere Drucker angeschlossen und konfiguriert, diese stehen aber in verschiedenen Räumen, so daß jeweils nur bestimmte Benutzer Zugang dazu haben. In diesen Fällen ist es nicht sinnvoll, wenn die Benutzer in einem Büro bei der Auswahl der Drucker in HIT auch die Druckeraufrufe eines anderen Büros angeboten bekommen.

Um nun die Auswahl der Druckeraufrufe für bestimmte Benutzer einzuschränken, gehen Sie folgendermaßen vor:

- **Erstellen Sie in der Benutzerkennung `admin` über die HIT-Systemverwaltung alle benötigten Druckeraufrufe.**
- **Richten Sie anschließend im Dateiverzeichnis `/usr/lib/hit/printer` Unter-Dateiverzeichnisse ein,**
  die den organisatorischen Einheiten entsprechen.
- **Übertragen Sie die Druckeraufrufe in die dafür vorgesehenen Unter-Dateiverzeichnisse.**
  Wenn Sie dazu das SINIX-Kommando `/bin/ln` verwenden, können Sie die Druckeraufrufe weiterhin über das Systemverwalter-Menü ändern. Allerdings werden dann den Benutzern, deren Auswahl nicht beschränkt wurde, alle Druckeraufrufe angeboten.
- **Belegen Sie nun für jeden betroffenen Benutzer die Shell-Variable `HITPGROUP` mit dem Namen des Dateiverzeichnisses, das seine Druckeraufrufe enthält.**
  Für die Belegung ist nur der Basisname des Dateiverzeichnisses zulässig (keine Pfadnamen!).

Beispiel

Im Dateiverzeichnis `/usr/lib/hit/printer` haben Sie die Unter-Dateiverzeichnisse `buero-1`, `buero-2` und `buero-3` eingerichtet.

Mit dem Eintrag `HITPGROUP=buero-1` für einen bestimmten Benutzer beschränken Sie dessen Auswahlmöglichkeiten in HIT auf die Druckeraufrufe, die im Dateiverzeichnis `/usr/lib/hit/printer/buero-1` vorhanden sind.

8.6 Erweiterung von 'ext-bearbeiten'

Bei der Funktion **Markieren und Bearbeiten**, **ext-bearbeiten** werden standardmäßig die Funktionen **ext-Speicher**, **rechnen** und **sortieren** angeboten. Sie können nun über Shell-Variablen zusätzliche Funktionen vereinbaren. Dazu verwenden Sie die Shell-Variablen `HITMARK` nach folgender Syntax:

HITMARK*zahl* = "[%EXOPT:opt] *name aufruf*"

HITMARK*zahl* — ist der Name der Shell-Variablen, wobei Sie für *zahl* eine Zahl zwischen 1 und 10 einsetzen müssen.

%EXOPT — ist der Name einer Option, mit der Sie zusätzliche Steueranweisungen übergeben können.
%EXOPT: muß durch eine oder mehrere der folgenden Optionen ergänzt werden:

- E Der markierte Bereich wird gelöscht und an das verarbeitende Programm übergeben. Die Ausgabe dieses Programms wird nicht in das HIT-Dokument eingefügt.
- R Nach Beendigung des Kommandos wird der Bildschirm wieder aufgebaut.

name — ist der Text, der in dem Auswahl-Menü nach `ext-bearbeiten` erscheint.

aufruf — ist das Programm, das die HIT-Daten verarbeitet.

Das verarbeitende Programm muß so strukturiert sein, daß es Daten, die im HIT-Format angeliefert werden, verarbeiten kann. Die Ausgabe des Programms muß ebenfalls im externen HIT-Format erfolgen. Die Datenkonvertierung von/nach HIT-Format erfolgt zweckmäßigerweise mit dem Modul `procon`, dessen Aufrufoptionen im Kapitel 9 ausführlich beschrieben sind.

Bitte beachten Sie:

Der Name des auszuführenden Programms muß zusätzlich in der HIT-Konfigurations-Datei `hit_pif` eingetragen werden, um Bildschirmausgaben (z.B. Fehlermeldungen) zu ermöglichen.

Erweiterung von 'importieren'

Zusätzliche Funktionen für **Bereich einfügen, importieren** können über die Shell-Variablen `HITIMPORT` vereinbart werden. Die Belegung der Shell-Variablen erfolgt nach folgender Syntax:

HITIMPORT*zahl* = "[%IMPOPT:R] *name aufruf*"

HITIMPORT*zahl*	ist der Name der Shell-Variablen, wobei Sie für *zahl* eine Zahl zwischen 1 und 10 einsetzen müssen.
%IMPOPT	ist der Name einer Option, mit der Sie zusätzliche Steueranweisungen übergeben können. %IMPOPT: muß durch folgende Option ergänzt werden: R Nach Beendigung des Kommandos wird der Bildschirm wieder aufgebaut.
name	ist der Text, der in dem Auswahl-Menü nach `importieren` erscheint.
aufruf	ist das Programm, das nach Auswahl von *name* ausgeführt werden soll.

Die Ausgabe des Programms muß im externen HIT-Format erfolgen. Die Datenkonvertierung nach HIT-Format erfolgt zweckmäßigerweise mit dem Modul `procon` (siehe Kapitel 9).

Bitte beachten Sie, daß der Name des auszuführenden Programms zusätzlich in der HIT-Konfigurations-Datei `hit_pif` eingetragen werden muß, um Bildschirmausgaben (z.B. Fehlermeldungen) zu ermöglichen.

"Satztechnisch bedingte Leerseite"

8.7 Anschluß von Anwenderprogrammen

Die Schnittstelle, über die der Anschluß von SIPLAN an HIT realisiert ist, kann auch für andere Anwenderprogramme genutzt werden. Der Aufruf erfolgt jeweils über die Steuerzeile `Anwendung`.

Zur Installation ist folgende Vorgehensweise notwendig:

Zur Erweiterung des Auswahl-Menüs, das beim Erstellen der Steuerzeile `Anwendung` erscheint, muß zunächst im Dateiverzeichnis `/usr/lib/hit/scripts-D` die Datei `application` erweitert werden. Für jede zusätzliche Anwendung muß darin ein Eintrag in der Form
`/usr/lib/hit/scripts-D/APP/Anwendungsname`
gemacht werden.

Darüber hinaus müssen im Dateiverzeichnis `/usr/lib/hit/scripts-D/APP` für jede Anwendung zwei Programme (shell-Prozeduren) vorhanden sein: *Anwendungsname*.con und *Anwendungsname*.app. *Anwendungsname* muß in der Schreibweise identisch mit dem Eintrag in der Datei `/usr/lib/hit/scripts-D/application` sein!

Das Programm *Anwendungsname*.con wird von HIT immer dann aufgerufen, wenn eine bestehende Datei in HIT angezeigt werden soll (beim Einfügen einer Steuerzeile `Anwendung`, beim Aufruf eines Dokuments zur Bearbeitung). Die Schnittstelle in HIT erwartet die Ausgabe der Datei auf die Standardausgabe (stdout) im HIT4-Format. Zum Erzeugen des HIT4-Formats kann das Modul `procon` verwendet werden (siehe Kapitel 9).

Beispiel:

```
# Prozedurname: VI.con
# Konvertiert die Datei von ASCII-Format in das externe
# HIT-Format und schreibt sie auf stdout.

if [ ! -s $1 ]
then /bin/touch $1
fi
procon -h3 $1
```

Das Programm *Anwendungsname*.app dient dazu, die Anwendung mit der dazugehörigen Datei zur Bearbeitung aufzurufen und nach Beendigung der Anwendung HIT die Daten im externen HIT-Format zu übergeben.

Beispiel:

```
# Prozedurname: VI.app
# Ruft den Editor vi auf, konvertiert die Datei in das
# externe HIT-Format und schreibt sie auf stdout.

vi $1 < /dev/tty > /dev/tty
procon -h3 $1
```

> *Hinweis:*
>
> Die Zuweisung von `/dev/tty` für die Ein-/Ausgabe bei der Anwendung ist unbedingt erforderlich, wenn die Ein-/Ausgabe über Tastatur/Bildschirm erfolgt.

Der Name des auszuführenden Programms muß zusätzlich in der HIT-Konfigurations-Datei `hit_pif` eingetragen werden, um Bildschirmausgaben (z.B. Fehlermeldungen) zu ermöglichen.

Konfigurations-Datei hit_pif

Um unter COLLAGE externe Programme richtig aufrufen zu können, muß dem Aufrufer der Applikationstyp (OS/NS) bekannt sein. Alle innerhalb des HIT-Systems aufgerufenen Programme müssen deshalb mit dem jeweiligen Applikationstyp in der Konfigurations-Datei `hit_pif` eingetragen werden:

Programmname:Typ

Typ kann die Werte OS und NS annehmen. Ist ein Programm nicht eingetragen, so wird ihm automatisch der Typ NS (newstyle) zugeordnet. Programme, die mit OS (oldstyle) gekennzeichnet sind, werden mit einer Oldstyle-Emulation gestartet.

Beispiel:

```
hit:NS
sh:OS
ls:OS
```

Die Datei `hit_pif` befindet sich im Dateiverzeichnis `/usr/lib/hit/scripts-D`.

Graphik-Konfigurationsdatei gra_liste

Graphik-Pakete, mit deren Hilfe in HIT-Dokumenten Zeichnungen erstellt werden sollen, müssen in der Konfigurations-Datei `gra_liste` eingetragen sein. Aus der Datei `gra_liste` entnimmt HIT, wie und mit welchen Parametern die benötigten Programme aufzurufen sind.
Die Datei `gra_liste` befindet sich im Dateiverzeichnis `/usr/lib/hit/scripts-D`.

Beispiel:

```
# Schnittstelle fuer SIDRAW V2.0 auf PC MX2+/MX300:
# -------------------------------------------------
SIDRAW,                                          # Paketname
.drw, .drw, .drw,                                # Suffix1-3

drwcap -PID=$2 -PARAM=$0,                        # G-Aufruf
1,                                               # G-Protokoll
-PID=$2\n-WIDTH=$3\n-HEIGHT=$4\n$1\n,            # G-Param1
-DIR=$1\n,                                       # G-Param2

redraw,                                          # Z-Aufruf
2,                                               # Z-Protokoll
S\,$1\,$9\n,                                     # Z-Param1
R\,$1\,$5\,$6\,$3\,$4\,$7\,$8\,$9\,$A\n,         # Z-Param2
Z\,$1\,$3\,$4\,$7\,$8\n,                         # Z-Param3
D\,$1\n,                                         # Z-Param4
W\,$1\n,                                         # Z-Param5
2,                                               # D-Anzahl
9001=hmetapr $1 -l $7 -w $3 -h $4 -p SIE9001 -H,
9002=hmetapr $1 -l $7 -o $8 -w $3 -h $4 -p SIE9022-75 -H,
```

Aufbau und Syntax

Der Eintrag für ein Graphik-Paket zerfällt in vier Blöcke:

- Paketname und Datei-Suffices
- Aufruf des Graphikprogramms
- Aufruf des Zeichnungsprogramms
- Aufruf des Druckprogramms

Das Graphikprogramm wird zur Erstellung der Zeichnung benötigt, das Zeichenprogramm zur Anzeige im HIT-Dokument und das Druckprogramm zur Aufbereitung der Zeichnung für einen bestimmten Drucker.

Die einzelnen Einträge werden durch Komma (,) voneinander getrennt. Zur Strukturierung dürfen nach einem Komma Zeilenende-Zeichen und Leerzeichen bzw. Tabulatorzeichen stehen. Soll eines dieser Metazeichen an das aufgerufene Programm weitergegeben werden, so muß es im Eintrag entwertet werden:

`\,`	Komma ist Bestandteil eines Eintrags
`\n`	Zeilenende (`\n`) ist Bestandteil eines Eintrags
`\Blank`	Falls zu Beginn eines Eintrags ein Leerzeichen übergeben werden soll, muß es auf diese Weise geschrieben werden. Innerhalb eines Eintrags genügt ein normales Leerzeichen.
`\\`	Das Entwertungszeichen (\) ist Bestandteil eines Eintrags.

Im folgenden sind die einzelnen Einträge in der von HIT erwarteten Reihenfolge aufgelistet. Alle Einträge müssen vorhanden und durch Komma getrennt sein, da HIT sonst keine Zuordnung treffen kann.

Bei einigen Einträgen können von HIT bereitgestellte interne Parameter ($1 - $A) an das aufzurufende Programm übergeben werden. Auf die relevanten Parameter wird bei jedem Eintrag hingewiesen.

Paketname und Datei-Suffices

Paketname — Name des Graphikpaketes (`SIDRAW`)

Suffix1 — Suffix, das an die Zeichnungsdatei beim Aufruf des Graphikprogramms (`drwcap`) angeheftet wird. Ist bereits ein Suffix vorhanden, so wird dieses ersetzt.

Suffix2 — Suffix, das in die Zeichnungsdatei beim Aufruf des Zeichnungsprogramms (`redraw`) angeheftet wird.

Suffix3 — Suffix, das in der Zeichnungsdatei beim Aufruf des Druckprogramms (`hmetapr`) angeheftet wird.

Aufruf des Graphikprogramms

G-Aufruf — Aufruf des Graphikprogramms aus HIT heraus. *G-Aufruf* wird verwendet, wenn eine Zeichnung erstellt oder bearbeitet werden soll. Ob *G-Aufruf* jedesmal oder nur beim ersten Auftreten der genannten Funktionen benutzt wird, hängt vom angegebenen *G-Protokoll* ab.

$0 — Name der Parameterdatei, die von HIT zur Kommunikation mit dem Graphikprogramm verwendet wird (Protokoll 1).

$2 — Prozeß-ID des aufrufenden HIT (Protokoll 1).

$1, $3, $4 — können bei Protokoll 0 übergeben werden (siehe *G-Param1*)

G-Protokoll — *G-Protokoll* kann die Werte 0 und 1 annehmen.
0 bedeutet, daß das Graphikprogramm jedesmal neu aufgerufen wird.
Bei 1 wird das Graphikprogramm nur bei der ersten Verwendung über *G-Aufruf* angestoßen; bei den folgenden Aufrufen wird es über ein Signal reaktiviert. In diesem Fall werden über *G-Param1* und *G-Param2* die nötigen Parameter an das Graphikprogramm übergeben.

G-Param1 — Parameter, die das Graphikprogramm in der Parameterdatei erhält, wenn eine Graphik erstellt oder bearbeitet wird.

$1	Name der Graphikdatei
$2	Prozeß-ID des aufrufenden HIT
$3	Breite der Zeichnung in Pixeln
$4	Höhe der Zeichnung in Pixeln

G-Param2 — Parameter, die das Graphikprogramm zusätzlich zu *G-Param1* erhält, wenn seit der letzten Aktivierung ein Directory-Wechsel über HIT-MENÜ stattgefunden hat.

$1	Name des aktuellen Directories

Aufruf des Zeichenprogramms

Z-Aufruf — Aufruf des Zeichenprogramms aus HIT heraus. *Z-Aufruf* wird verwendet, wenn eine Zeichnung im HIT-Dokument angezeigt werden soll. Ob *Z-Aufruf* jedesmal oder nur bei der ersten Zeichenanforderung benutzt wird, hängt vom angegebenen *Z-Protokoll* ab.

$0	Name der Parameterdatei, die von HIT zur Kommunikation mit dem Graphikprogramm verwendet wird (Protokoll 1).
$2	Prozeß-ID des aufrufenden HIT (Protokoll 1)

$1, $3, $4, $5, $6, $7, $8, $9, $A — Können bei Protokoll 0 übergeben werden (siehe *Z-Param2*)

Z-Protokoll

Z-Protokoll kann die Werte 0, 1 und 2 annehmen.

0 bedeutet, daß das Zeichenprogramm jedesmal neu aufgerufen wird. In diesem Fall kann das Zeichenprogramm nur zur Anzeige benutzt werden; eine Existenzprüfung und die Funktion `Größe der Zeichnung ändern` sind nicht möglich. Der Modus `Zeichnung: sichtbar` sollte bei Protokoll 0 nur bei Erstellung/Bearbeitung der Zeichnung eingeschaltet werden.

Bei 1 wird das Zeichenprogramm nur bei der ersten Verwendung über *Z-Aufruf* angestoßen; bei den folgenden Aufrufen wird es über ein Signal reaktiviert. Ergebnisse werden in der Parameterdatei erwartet.

Bei 2 wird das Zeichenprogramm ebenfalls nur bei der ersten Verwendung über *Z-Aufruf* aktiviert. Bei den folgenden Aufrufen werden die Anweisungen an das Zeichenprogramm über Pipe auf Standard-Eingabe geschickt, das Ergebnis wird von der Standardausgabe des Zeichenprogramms gelesen.

Z-Param1

Parameter, die das Zeichenprogramm enthält, wenn die Existenz einer Zeichnung überprüft werden soll.

$1	Name der Graphikdatei
$9	im HIT-Fenster eingestellter Font
$2	Prozeß-ID des aufrufenden HIT (Protokoll 1)

Z-Param2

Parameter, die das Zeichenprogramm erhält, wenn im HIT-Dokument eine Zeichnung ausgegeben werden soll.

	$1	Name der Graphikdatei
	$3	Breite der Zeichnung in Pixeln
	$4	Höhe der Zeichnung in Pixeln
	$5	x-Koordinate im HIT-Fenster in Pixeln
	$6	y-Koordinate im HIT-Fenster in Pixeln
	$7	Verschiebung in x-Richtung in Pixeln, falls linke obere Ecke nicht sichtbar (relativ zu $5)
	$8	Verschiebung in y-Richtung in Pixeln, falls linke obere Ecke nicht sichtbar (relativ zu $6)
	$9	im HIT-Fenster eingestellter Font
	$A	Window-ID des HIT-Fensters
	$2	Prozeß-ID des aufrufenden HIT (Protokoll 1)
Z-Param3		Parameter, die das Zeichenprogramm erhält, wenn die Zeichnung vergrößert oder verkleinert werden soll.
	$1	Name der Graphikdatei
	$3	Breite der Zeichnung in Pixeln
	$4	Höhe der Zeichnung in Pixeln
	$7	neue Breite der Zeichnung in Pixeln
	$8	neue Höhe der Zeichnung in Pixeln
	$2	Prozeß-ID des aufrufenden HIT (Protokoll 1)
Z-Param4		Paramater, die das Zeichenprogramm erhält, wenn seit der letzten Aktivierung ein Directory-Wechsel über HIT-MENÜ stattgefunden hat. Bei Protokoll 1 wird dieser Parameter mit der Parameterdatei übergeben, bei Protokoll 2 sofort über Pipe an das Zeichenprogramm geschickt.
	$1	Name des aktuellen Directories
Z-Param5		Parameter, die das Zeichenprogramm erhält, wenn die Schreibberechtigung für eine Zeichnung überprüft werden soll.
	$1	Name der Graphikdatei

Aufruf des Druckprogramms

D-Anzahl Anzahl der Druckprogramm-Aufrufe

D-Name = D-Aufruf Aufruf des Druckprogramms. Es müssen genau soviele Aufrufe angegeben werden wie bei *D-Anzahl* genannt.

Vom HIT-Druckprogramm `filter` aus erfolgt der Zugriff auf das Zeichnungs-Druckprogramm über den symbolischen Namen *D-Name*. *D-Name* wird dazu bei dem `filter`-Schalter `-P` angegeben. Bei der automatischen Erzeugung von Druckeraufrufen über das Systemverwaltermenü wird vorausgesetzt, daß die symbolischen Namen denen der System-Konfigurationsdatei `CONFIG` entsprechen (z.B. `9001, 9022` usw.).

D-Aufruf ist der eigentliche Aufruf des Druckprogramms mit allen von diesem benötigten Parametern. Das Druckprogramm muß die aufbereitete Zeichnung auf die Standard-Ausgabe schreiben. Zusätzlich muß der Ausgabe die HIT-Graphik-Kennung vorausgehen. Die Graphik-Kennung besteht aus einem kleinen "z" und einer dreistelligen Zeilenangabe (ggf. mit führenden Nullen). Die Zeilenangabe beschreibt, wieviele Zeilen (in 1/6 Zoll) die ausgedruckte Zeichnung in Anspruch nimmt.

$1	Name der Graphikdatei
$3	Breite der Zeichnung in Pixeln
$4	Höhe der Zeichnung in Pixeln
$7	Abstand der Zeichnung vom linken Rand des Zeichenblattes
$8	Abstand der Zeichnung vom oberen Rand des Zeichenblattes

Das Programm `hmetapr`

Das Programm `hmetapr` kann zum Drucken von Zeichnungen eingesetzt werden, die im COLLAGE-Metafile-Format abgelegt sind (z.B. SIDRAW-Zeichnungen, SICHART-Graphiken oder HARDCOPY-Metafiles). Der normale Anwendungsfall wird die Einbindung von `hmetapr` in die Druckausgabe des HIT sein, dazu sind entsprechende Eintragungen in der `gra_liste` vorzunehmen.

Verwendbare Optionen (ähnlich wie bei metaprintc):

[-s] file	Name des Metafiles (muß angegeben werden)
-t treename	Name des Objektbaums im Metafile (Default: leer)
-H	HIT-Graphikkopf ausgeben (Default: nicht ausgeben)
-p device	Name des Druckers (Default: SIE9001, s.u.)
-l n	linke Einrückung um n Bildpunkte (Default: n = 0)
-o n	Abstand zum oberen Rand n Bildpunkte (Default: n = 0)
-x n	x-Offset in der Zeichnung n Bildpunkte (Default: n = 0)
-y n	y-Offset in der Zeichnung n Bildpunkte (Default: n = 0)
-w n	maximale Zeichnungsbreite n Bildpunkte (Default: n = Breite der Zeichnung)
-h n	maximale Zeichnungshöhe n Bildpunkte (Default: n = Höhe der Zeichnung)
-X n	Breite der Ausgabe auf Papier (Default: der bei `-w` angegebene Wert)
-Y n	Höhe der Ausgabe auf Papier (Default: der bei `-h` angegebene Wert)
-f {p | l}	Orientierung der Ausgabe (p für Portrait, l für Landscape, Default: `p`)

`hmetapr` gibt die Druck-Darstellung der Zeichnung nach *stdout* aus, wobei dieser Ausgabe der HIT-Graphikkopf vorausgeht (*z999*, *999* entspricht der Zahl der Zeilen, die durch die Zeichnung belegt werden). Ggf. werden Fehlermeldungen nach *stderr* ausgegeben.

Mögliche Angaben bei der Option `-p`:

SIE9001	Drucker 9001 (Default-Drucker)
SIE9012	Drucker 9012 mit 240 dpi
SIE9012-120	Drucker 9012 mit 120 dpi
SIE9012-80	Drucker 9012 mit 80 dpi
SIE9012-60	Drucker 9012 mit 60 dpi
SIE9013	Drucker 9013 mit 144 dpi
SIE9013-72	Drucker 9013 mit 72 dpi
SIE9022	Drucker 9022 mit 300 dpi
SIE9022-150	Drucker 9022 mit 150 dpi
SIE9022-75	Drucker 9022 mit 75 dpi
laserjet	Drucker HP Laserjet mit 300 dpi
laserjet-150	Drucker HP Laserjet mit 150 dpi
laserjet-75	Drucker HP Laserjet mit 75 dpi
postscript	PostScript-Drucker

Die Option `-o` ist nur für solche Drucker anzugeben, für die kein Bit-Image, sondern z.B. Vektorgraphik oder PostScript-Code erzeugt wird (momentan Drucker 9022 und PostScript-Drucker).

Beispiel für Eintragungen in der `gra_liste`:

```
9001=hmetapr $1 -l $7 -w $3 -h $4 -p SIE9001 -H,
9002=hmetapr $1 -l $7 -o $8 -w $3 -h $4 -p SIE9022 -H,
HP=hmetapr $1 -l $7 -w $3 -h $4 -p laserjet -H,
PS=hmetapr $1 -l $7 -o $8 -w $3 -h $4 -p postscript -H,
```

Hinweis zum Drucken von HARDCOPY-Metafiles:

Bei solchen Metafiles ist die Option `-t hardcopy` anzugeben, da der Objektbaum diesen Namen hat. Wenn die Optionen `-X` und `-Y` zum Verkleinern nicht angegeben werden, ist die Orientierung mittels `-fl` auf Landscape umzustellen. Da sich Breite und Höhe eines Hardcopys zu 720 bzw. 540 Pixeln ergibt, kann durch die Angabe der Optionen `-X 360` und `-Y 270` die Ausgabe auf 1/4 verkleinert werden, z.B. zur Integration in ein HIT-Dokument über die Steuerzeile *Graphik* (dazu ist beim Aufruf von `hmetapr` die Option -H anzugeben). Über die Steuerzeile *Zeichnung* kann ein HARDCOPY-Metafile nicht in ein HIT-Dokument integriert werden!

8.8 Nachrichten an HIT-Benutzer

HIT-MENÜ überprüft laufend, ob der Benutzer Post (SINIX-Kommando `mail`) erhält. Dabei wird der Inhalt der Variablen `MAIL` ausgewertet, die standardmäßig auf den Dateinamen `/usr/spool/mail/<$USER>` gesetzt ist. Wenn sich die Größe und der Zeitpunkt der letzten Modifikation dieser Datei ändert, stellt HIT-MENÜ dies fest und weist den Benutzer auf den "Posteingang" hin.

Wenn gewünscht wird, daß mehrere Dateien hinsichtlich einer Änderung der Größe oder dem Zeitpunkt der letzten Modifikation überprüft werden oder wenn die Ausgabe eines bestimmten Meldungstextes erfolgen soll, kann dies über die Variable `MAILPATH` geschehen. `MAILPATH` ersetzt `MAIL`.

Die shell-Variable MAILPATH verwenden Sie mit folgender Syntax:

MAILPATH = "*dateiname* [*%meldungstext*] [:*dateiname* [*%meldungstext*]]"

dateiname — Der Name der Datei, die überprüft wird. Mehrere Dateinamen müssen durch Doppelpunkt getrennt werden.

%meldungstext — Wenn auf den Dateiname ein Prozentzeichen folgt, wird der anschließende Text (bis zum nächsten Doppelpunkt) als Meldungstext ausgegeben.

Beispiel:

```
MAILPATH="/usr/spool/mail/gast%Post angekommen!:\
/usr/lib/hit/public/Rundschreiben%Rundschreiben angekommen!"
```

Der Benutzer `gast` wird mit der Meldung "`Post angekommen!`" informiert, wenn er über die Post-Funktion Nachricht erhält. Die Meldung "`Rundschreiben angekommen!`" informiert ihn darüber, daß im Verteiler von HIT-MENÜ das Schriftstück `Rundschreiben` eine neue Information bereithält.

Über die shell-Variable `MAILCHECK` kann der Zeitabstand in Sekunden angegeben werden, nach dem die genannten Dateien überprüft werden. Der geringstmögliche Wert dabei ist 60.

Gültige Dateinamen

In der Regel dürfen Dateinamen nur Zeichen aus dem 7-Bit-ASCII-Zeichensatz, also z.B. keine Umlaute, enthalten.

Sie haben die Möglichkeit, Dateinamen, die Zeichen aus dem 8-Bit-ASCII-Zeichensatz enthalten, im gesamten HIT-Textsystem zu verarbeiten. Hierzu zählen alle Eingabefelder in allen Komponenten des HIT-Systems.

Zu diesem Zweck stehen Ihnen folgende Shell-Variablen zur Verfügung:

`OCIS-FILENAME`	Zulässig sind hier die Einstellungen ISO 8859-1 und ASCII.
`LANG`	Zulässig sind die Einstellungen, die auf ".88591" enden.

Wenn beide Variablen gesetzt sind, hat die Variable `OCIS-FILENAME` Vorrang. Ist keine der beiden Shell-Variablen gesetzt, erfolgt eine Minimalkonvertierung in 7-Bit-ASCII-Dateinamen.

Bei der Verwendung von `OCIS-FILENAME` gelten einige Einschränkungen für die Benutzung von Sonderzeichen in Dateinamen (siehe nachfolgende Tabelle). Wenn die Variable `LANG` belegt ist und `OCIS-FILENAME` nicht, dann gelten diese Einschränkungen nicht.

Folgende Sonderzeichen sind bei Verwendung von OCIS-FILENAME nicht in Dateinamen erlaubt:

!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
:	;	<	=	>	?	@	[	\	]	^	`	{	|	}
~	¡	Ç	£	¤	¥	¦	§	¨	©	ª	«	®	¯	°
±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿

9 Konvertierungsprogramm PROCON

Das Modul `procon` erlaubt es, HIT-Dokumente in Dokumente mit anderen Datenformaten umzuwandeln bzw. aus Dokumenten, die nicht im HIT-Format vorliegen, HIT-Dokumente zu erzeugen.

Um Fehler bei der Anwendung zu vermeiden, sind neben den Aufrufoptionen für `procon` die erzeugbaren Formate sowie evtl. Besonderheiten ausführlich beschrieben.

9.1 Aufruf aus der Shell

Syntax:

procon [*Option*] ...

Folgende Optionen können angegeben werden:

Dateiname
: Name der Eingabedatei.
Voreinstellung ist "stdin". Pflichteingabe ist *Dateiname* nur bei Konvertierung aus HIT2-Format (Option `-h1`), da hier 2 Dateien gelesen werden müssen.

-a *Dateiname*
: Name der Ausgabedatei.
Voreinstellung ist "stdout". Plichteingabe ist `-a` *Dateiname* nur bei Konvertierung in HIT2-Format (Option `-p1`), da hier 2 Dateien geschrieben werden müssen.

-e *Dateiname*
: Name einer Fehlermeldungsdatei
In diese werden (im HIT-Format) die bei der Konvertierung aufgetretenen Warnungen und Fehler geschrieben. Standardmäßig werden sie mit `hless` angezeigt. Die Angabe wird ignoriert bzw. kann entfallen wenn der Schalter `-q` gesetzt ist.

-s

Warnungen (nicht Fehler), die bei der Konvertierung aufgetreten sind, werden nicht auf das Fehler-Device geschrieben.

-q

Alle Fehlerarten, die bei der Konvertierung aufgetreten sind, werden unterdrückt. Die Angabe bei der Option `-e` wird ignoriert. Der Ende-Status gibt Auskunft darüber, ob die Konvertierung erfolgreich durchgeführt werden konnte.

-h *pnr*
-p *pnr*

Steuerung der Konvertierungsrichtung und des Ziel- bzw. Quellformates.

pnr	**Format**
1 :	alte HIT-Formate
2 :	Basic-Teletex
3 :	ISO-646
4 :	ISO-6937
5 :	TELEX
6 :	ISO-8859
>9 :	USERPROCON

-h Das ausgewählte Format wird in HIT4- Format konvertiert.

-p Eine HIT4-Format-Datei wird in das ausgewählte Format konvertiert.

Wird weder `-p` noch `-h` angegeben wird von einem unbekannten Format ausgegangen, das in HIT4-Format konvertiert werden soll (siehe Kapitel 9.2.1).

-d *pnr*

procon arbeitet beim Einlesen eines unbekannten Formates per Voreinstellung mit dem durch *pnr* angegebenen Format.

-B *zeichen*[/*attribut*]

Bei der Konvertierung werden nicht konvertierbare Zeichen durch das angegebene Zeichen *zeichen* ersetzt.
zeichen ist ein beliebiges Zeichen aus dem Zeichenvorrat des Zielzeichensatzes und muß in seinem hexadezimalen Wert angegeben werden. (z.B. hat der Buchstabe A den hexadezimalen Wert 41).
zeichen kann mit dem Attribut *attribut* ausgegeben werden.
Dabei sind folgende Werte erlaubt:

0 bzw. N	normal
1 bzw. U	unterstrichen
2 bzw. B	fett
3 bzw. C	durchgestrichen
4 bzw. I	kursiv
5 bzw. H	hochstellen
6 bzw. L	tiefstellen

-M

Angabe einer sprachspezifischen Meldungsdatei.
Voreinstellung ist :
`/usr/lib/hit/messages/procon/messages`

-T

Bei der Konvertierung werden HIT-Tabulatoren in Tabulatoren des Zielformats umgewandelt.

"Satztechnisch bedingte Leerseite"

Optionen für ISO-Formate

-g

Bei der Konvertierung werden die Steuerzeilen `Graphik` und `Zeichnung` berücksichtigt. Die Graphik bzw. Zeichnung wird in die entsprechende Anzahl Leerzeilen konvertiert.
Bei Angabe von `-g` wird automatisch die Option `-G` gesetzt (siehe Optionen für alte HIT-Formate).

-G

Bei der Konvertierung in/aus HIT4-Format werden Liniengraphikzeichen von SIPLAN übernommen. Die Angabe dieser Option ist nur sinnvoll bei ISO 8859 bzw. ISO 6937.

-t

Bei der Konvertierung in ein Nicht-HIT-Format werden Leerzeichen per Voreinstellung nicht in sogenannten Tabulatorschritten zusammengefaßt. Eine Ausnahme bildet hier nur das ISO-646 (ASCII/IA5) - Format.
Mit dieser Option kann (z.B. zur Speicherplatz-Optimierung) diese Voreinstellung geändert werden. Es gilt folgende Regel:
`-t` schaltet die Leerzeichenkontraktion ein
`+t` schaltet die Leerzeichenkontraktion aus.

-NL

Angabe nur bei ISO-646-Format (Protokollnummer 3) möglich.
Mit dieser Option kann die UNIX-ASCII-übliche Interpretation des `LF`-Zeichens als implizites `CR LF` ausgeschaltet werden.

-x

Angabe nur bei ISO-646-Format (Protokollnummer 3) möglich.
Es werden die Tabellen für das Extended-ASCII-Format für die Konvertierung herangezogen.

-D *pnr*

Angabe einer Draft-Nummer *pnr* ist nur bei ISO-8859-Format (Protokollnummer 6) möglich. Standardmäßig wird von "Latin Alphabet I" ausgegangen.

pnr	**Draft**
1	Latin Alphabet I
2	Latin Alphabet II

3	Latin Alphabet III
4	Latin Alphabet IV

-L0 Angabe nur bei Konvertierung in ISO-6937-Format (Option `-p4`).
Für die Einbindung des Electronic-Mail-Formates in die Konvertierung muß die Option `-L0` gesetzt sein. Es wird dann gemäß dem zugehörigen Zeichenvorrat und den Rahmenbedingungen des ISO-6937-Formates konvertiert.

-i, +i Angabe nur bei Konvertierung in/aus ISO-646-Format und HIT2-Format (Protokollnummern `3` und `1`) möglich.
Bei Angabe von `+i` werden bei der Konvertierung die Angaben zur nationalen Konvertierung, wie sie in der Meldungstext-Datei eingetragen sind, herangezogen.
Bei Angabe von `-i` werden diese ignoriert und die Festeinstellungen zur Konvertierung benützt (siehe Kapitel "Nationale Konvertierung").

Optionen für alte HIT-Formate

-Z *znr* Bei der Konvertierung in das HIT4-Format werden die Zeilenlinealbereichs-Attribute auf `linksbündig` eingestellt. Mit der Angabe dieser Option kann diese Voreinstellung auf das bei *znr* eingestellte Attribut umgeschaltet werden.

znr	**Attribut**
1	Linksbündig
2	Blocksatz
3	Zentriert
4	Rechtsbündig
5	Geschützt

-n Angabe nur bei Konvertierung in/aus HIT2-Format (`-p1` oder `-h1`). Bei der Konvertierung werden keine Zeilenlinealdateien erzeugt bzw. ausgewertet.

-V2.0 Angabe nur bei Konvertierung in HIT2-Format (Option `-p`) möglich.
Erzeugen einer Datei gemäß den Rahmenbedingungen des HIT-V2.0-Formates.

-V2.1 Angabe nur bei Konvertierung in HIT2-Format (Option `-p`) möglich.
Erzeugen einer Datei gemäß den Rahmenbedingungen des HIT-V2.1-Formates kombiniert mit denen des HIT-V2.0-Formats.

-V2A Angabe nur bei Konvertierung in HIT2-Format (Option `-p`) möglich.
Das Format HIT-V2.0 wird von der Konvertierung ausgeschlossen. Es werden nur die Rahmenbedingungen des HIT-V2.1-Formates bei der Konvertierung berücksichtigt.

9.1.1 Shell-Variable

Die Aufrufoptionen für `procon` können auch über die folgenden Shell-Variablen übergeben werden:

`PROCONOPT` Einstellung von Aufrufoptionen und deren zugehöriger Paramter.
Zulässig sind alle Optionen außer `-h` und `-p`.

`HITUSRPROC` Name eines Programmes, das beim Aufruf eines USERPROCONS durch procon aktiviert werden soll.

`HITLANGUAGE` Auswahl einer anderssprachigen Meldungsdatei

Allgemein gilt der Grundsatz:

- Voreinstellungen werden nur dann ausgewertet, wenn sie für die entsprechende Konvertierung möglich sind (z.B. kommt `-V2.0` bei den Optionen `-h1` und `-p1` zur Wirkung).
- Voreinstellungen können beim Aufruf von procon aus der Shell durch die entgegengesetzte Option überschieben werden. Es gilt dann die in der Shell gesetzte Option (s. Beispiel).
- Bei gleichem Eingabelevel hebt eine ' + '-Option den Effekt einer '-'-Option auf. In diesem Fall gilt die Standardeinstellung der jeweiligen Option.

Beispiel:

Eintrag in der `PROCONOPT`-Shell-Variable:

```
PROCONOPT=-s
```

Wenn bei einer Konvertierung ausnahmsweise einmal die Warnungstexte ausgegeben werden sollen, so muß dazu nicht der Eintrag in der Shell-Variable geändert werden, sondern es genügt, beim Aufruf von procon die aufhebende Option mit anzugeben. In diesem Beispiel:

```
procon ... +s
```

Wenn der gleiche Effekt über die Menüs aus HIT erreicht werden soll, so ist allgemein die entgegengesetzte Einstellung der entsprechenden Variablen anzuwählen. In diesem Beispiel:

```
Warnungen ein
```

9.1.2 Ende-Status

Die möglichen Ende-Status von procon sind zusammen mit ihrer Bedeutung im folgenden aufgelistet:

0 Es sind keine Fehler bei der Konvertierung aufgetreten.

1 Es sind nur Warnungen aufgetreten.

50 Es sind nur Fehler aufgetreten.

51 Es sind sowohl Warnungen wie auch Fehler aufgetreten.

Ende-Status, die zum Abbruch der Konvertierung führen

100 Nicht näher spezifizierter fataler Fehler.

101 Falsche Aufrufparameter.

102 Protokollnummer fehlt.

103 Unvereinbare Optionen.

104 Unbedingt benötigte Datei konnte nicht geöffnet werden.

105 Speicherplatzproblem

106 Fehler bei der Zuordnung von national definierten Zeichen.

107 Eine Zeile konnte nicht in die Zieldatei geschrieben werden.

108 User-Procon nicht angeschlossen.

109 User-Procon nicht ansprechbar..

110 Nicht näher spezifizierter Fehler bei der Konvertierung mit Dateien des HIT3-Formates.

9.1.3 Fehlerbehandlung

Die möglichen Fehlermeldungen bei der Konvertierung sind drei Fehlerklassen zugeordnet:

Fatale Fehler

Sie führen zum Abbruch des Programmes. Darunter fallen Fehler wie mangelnder Speicherplatz, ungültige Parameterversorgung, und ähnlich Fehler, die einen korrekten Programmablauf unmöglich machen.

Fehler

Es handelt sich um schwerwiegendere Veränderungen in der konvertierten Datei (z.B eine zwangsweise umgebrochenen Zeile). Diese Fehlermeldungen können mit der Option `-q` unterdrückt werden.

Warnungen

Diese Meldungen zeigen Konvertierungsprobleme an, die das konvertierte Dokument nicht wesentlich verändern (z.B. nicht konvertierbare Zeichen). Sie können über die Option `-s` unterdrückt werden.

9.2 Konvertierung in HIT4-Format

9.2.1 Automatische Erkennung des zu konvertierenden Formates

Wird nicht explizit angegeben, von welchem Format eingelesen werden soll, so wird wie folgt vorgegangen:

- Zuerst wird überprüft, ob es sich um eine Datei im Format der HIT-V3.0 bzw. HIT-V3.1 handelt.
- Ist dies nicht der Fall, wird getestet, ob eine Zeilenlinealdatei *.Dateiname*.Z vorhanden ist. Falls ja, wird von einer HIT-Datei der Version 2.0/2.1 ausgegangen.
- Falls ein Userprocon angeschlossen ist, wird dieser als nächstes auf die Datei angewendet.
- Liegt keiner der 3 genannten Fälle vor, wird von dem Standard-Format ISO-646 ausgegangen. Mit dem Schalter `-d` *pnr* kann dieses Standard-Format umdefiniert werden.

Dabei ist folgendes zu beachten:

- Falls eine ISO-646-Datei im "Linefeed Mode" vorliegt, muß dies zusätzlich angegeben werden. Standardmäßig wird immer vom "Newline Mode" ausgegangen.
- Liegt die Datei in keinem der genannten Formate vor, so kann die Konvertierung zu einem unbefriedigenden Ergebnis führen (z.B, wenn eine ISO-8859-Datei in das HIT4-Format konvertiert wird).

9.2.2 Voreinstellungen bei Konvertierung ins HIT-Format

Bei der Konvertierung in HIT4-Format gelten folgende Voreinstellungen, die gegebenenfalls über das Konvertierungs-Menü oder über die Angabe von Optionen geändert werden können.

ISO-6937 Teletex	Linefeed Mode International Optionen `-i` und `-NL` ohne Wirkung
TELEX	Linefeed Mode Optionen `-i` und `-NL` ohne Wirkung
ISO-646	Newline Mode - über Konvertierungs-Menü oder über die Option `-NL` kann der Linefeed Mode eingeschaltet werden (für IA5 Konvertierung). National - über Konvertierungs-Menü oder über die Option `-i` kann auf international umgeschaltet werden. Über Konvertierungs-Menü oder über die Option `-x` kann das Format "Extended ASCII" angesprochen werden.
ISO-8859	Newline Mode International Optionen `-i` und `-NL` ohne Wirkung. Über Konvertierungs-Menü oder über die Option `-D` *pnr* kann zwischen Latin Alphabet I und II unterschieden werden.
alte HIT-Formate	Newline Mode National - über Konvertierungs-Menü oder über die Option `-i` kann auf internationale Konvertierung umgeschaltet werden. Option `-NL` ohne Wirkung.
Unbekanntes Format	Newline Mode - mit der Option `-NL` kann auf Linefeed Mode umgeschaltet werden. National - mit der Option `-i` kann auf internationale Konvertierung umgeschaltet werden.

9.2.3 ISO-6937 und verwandte Formate

Da der Zeichenvorrat des ISO-6937-Formates eine Obermenge der Zeichenvorräte der Formate ISO-646 (ohne Extended ASCII), TELEX und TELETEX darstellt, können diese Protokolle bei der Konvertierung nach HIT zusammengefaßt werden. Unterschiede ergeben sich nur bei den vorne genannten Voreinstellungen.

Zeichenvorrat

Der Zeichenvorrat, der in ISO-6937/2 definiert ist, wird in die entsprechenden HIT-Zeichen konvertiert.

Zusätzlich wird das Zeichen 0xC9 (Umlautmarks) interpetiert wie der Code 0xC8 (für TTX).

Das Zeichen 0x24 wird im HIT immer als Dollar Sign ($) dargestellt, und nicht wie in TTX als das Currency Sign.

Steuerzeichen

Bei der Konvertierung werden alle Steuerzeichenfolgen erkannt, die dem in ISO-6429 definierten Standard entsprechen. Siehe dort:
- Kontrollfunktionen des C1 sets (5.1.1)
- " aus Kontrollsequenzen (5.1.2)
- " wie ESC Fs (5.1.3)

Tabelle der ausgewerteten Steuerzeichen(folgen):

ISO6429	HIT	
TAB	Expansion zu Blanks	
BS	Kombination von Zeichen	<1>
LF	Neue Zeile	<1>
FF	Neue Seite-Lineal	<1>
CR	Wagenrücklauf	<1>
PLD	Attribut Tief	
PLU	Attribut Hoch	
HPR	Horizontale Positionierung	<1>
VPR	entfällt	
RLF	entfällt	
PFS	ohne Bedeutung	
SGR 0	Attribut Normal	
SGR 1	Attribut Fett	
SGR 3	Attribut Kursiv	
SGR 4	Attribut Unterstrichen	
SGR 9	Attribut Durchgestrichen	
SHS 0	Zeichenbreite 10	
SHS 1	Zeichenbreite 12	
SHS 2	Zeichenbreite 15	
SVS 0	Zeilenabstand 1.0	
SVS 1	" 1.5	
SVS 2	" 2.0	
SVS 3	" 0.5	
SVS 4	" 0.75	
SVS 5	" 1.0	
SVS 6	" 1.5	
SVS 7	" 2.0	
SVS 8	entfällt (Standard 1.0)	
SO	entfällt	
SI	entfällt	
SS2	entfällt	
SS3	entfällt	
LS(0,1,2,3)	entfällt	
LS(1,2,3)R	entfällt	
SUB	entfällt	
IGS	entfällt	
PT/DT	entfällt	<2>

<1> Falls durch eines dieser Steuerzeichen zwei Zeichen übereinandergedruckt würden, gilt:

- Die zwei Zeichen sind eine gültige Kombination:
  → entsprechendes Kombizeichen
- Beide Zeichen sind gleich:
  → Attribut Fett wird für dieses Zeichen gesetzt
- Eines der beiden Zeichen ist 0x5F (_) oder 0xCC (Non-spacing _):
  → Attribut Unterstreichen wird für dieses Zeichen gesetzt
- In allen anderen Fällen wird von einer ungültigen Zeichenkombination ausgegangen:
  → das 2. Zeichen wird mit Warnung ausgefiltert.

<2> DT und PT haben in HIT keine Äquivalente und werden deshalb durch Zurücksetzen aller Format-Steuerzeichen auf ihre Standardwerte interpretiert.

9.2.4 Alte HIT-Formate

Zeichenvorrat

Der HIT2/HIT3-Zeichensatz wird in die entsprechenden Zeichen des aktuellen HIT-Formates umgewandelt. Dabei gilt bei Version 2.x die Regelung für nationale Konvertierung (siehe ISO-646) sowie folgende Besonderheiten:

Für HIT-V2.1-Dokumente werden zusätzlich die folgenden Zeichen (nationale Sonderzeichen) entsprechend der in der jeweiligen Meldungsdatei eingestellten Zuordnung konvertiert:

0x02 - 0x08
0x0A - 0x1A
0x1C - 0x1F

- Bindestriche mit gesetztem Trennattribut werden in Trennstriche konvertiert.
- Klammer-Zeilenlineale in Umbruch-Steuerzeilen für den jeweiligen Linealbereich umgewandelt.
- Unterstriche (alte geschützte Leerzeichen) werden in geschützte Leerzeichen konvertiert.
  Eine gesetzte -n Option wird im Header der HIT4-Datei vermerkt.

Attribute

Die Attribute der alten HIT-Formate werden in die entsprechenden Werte des HIT4-Formates umgewandelt.

Zeilenlineale

Umsetzung der Zeilenlineal-Symbole:

Alt	Neu
rechter, linker Rand	rechter, linker Rand
TAB	Linksbündiger Tabulator
Tabellentabulator	Linksbündiger Tabulator und Setzen des ZL-Bereichsattributes Geschützt.
Klingelzeichen	Klingelzeichen
Dezimaltabulator	Dezimaltabulator
Tausender-Trennstelle	Tausender-Trennstelle

Steuerzeilen

Nur für Version 3.0, 3.1:
Vorhandene Steuerzeilen werden unverändert übernommen.

Steuerzeilen `Zeichnung` aus HIT-COL-V3.1-Format werden in HIT-COL-V4.0-Format konvertiert. Eine Rückkonvertierung in HIT-COL-V3.1-Format ist nicht möglich.

9.3 Konvertierung in ein Fremdformat

Ausgehend von HIT4-Format können folgende Formate angesprochen werden:

- ISO 646 (UNIX-ASCII, IA5, Extended ASCII)
- TELEX
- BASIC-TELETEX
- ISO 8859
- ISO 6937
- Alte HIT-Formate

Hinweis:

Im Kapitel 3.19 "Dokument sichern in anderes Format" des Benutzerhandbuchs finden Sie Abbildungen des HIT-Konvertierungs-Menüs mit den jeweiligen Standardeinstellungen.

9.3.1 ISO 646

Unter diese Formatklasse fallen 3 ähnliche Formate:

- IA5: entspricht ISO 646 im Linefeed Mode.
  Linefeed Mode bedeutet, daß eine neue Zeile mit der Sequenz `CR LF` und eine neue Seite mit `CR FF` erzeugt wird.

- UNIX-ASCII: entspricht ISO 646 im Newline Mode.
  Newline Mode bedeutet, daß Dateien erzeugt werden wie sie auf UNIX (trademark of Bell) üblich sind, d.h daß das Steuerzeichen `LF` implizit ein `CR LF` bedeutet.

- Extended ASCII: entspricht einem 8-BIT-ASCII definiert in Anhang F, IBM RT PC, C Language Guide and Reference, AIX V2, November 1985.

Die Auswahl der Formate erfolgt im Konvertierungs-Menü in HIT über die Menüpunkte `Modus UNIX/ISO-FORMAT` und `Extended ASCII ja` bzw. über die Optionen `-NL` und `-x`.

Zeichenvorrat

Der gültige Zeichenvorrat ist der ISO 646-Beschreibung zu entnehmen. Ungültige Zeichen werden mit einer entsprechenden Warnung durch den Code 0x20 (Leerzeichen) ersetzt.

Für die folgenden Zeichen gelten zur Behandlung landessprachlicher Sonderzeichen spezielle Konvertierungsregeln:

Zeichen	Name	Sedezimaler Wert
#	Hash	0x23
$	Dollar	0x24
<	Pfeil nach links	0x3c
@	Paragraph	0x40
[	eckige Klammer auf	0x5B
\	Backslash	0x5C
]	eckige Klammer zu	0x5D
^	Accent circuonflexe	0x5E
{	geschweifte Klammer auf	0x7B
|	Pipe-Zeichen	0x7C
}	geschweifte Klammer zu	0x7D
~	Tilde	0x7E
`	Accent grave	0x60

Die Sonderbehandlung geschieht wie folgt:

Für diese Zeichen wurden in der Meldungsdatei des Konvertierprogrammes Einträge reserviert. Die dort eingetragenen landesspezifischen Zeichen aus dem HIT-Zeichensatz werden bei der Konvertierung durch die entsprechenden ASCII-Zeichen ersetzt. Die internationale Zeichenzuordnung, als Alternative zur nationalen Variante, ist im Konvertierungs-Menü über `international/national` bzw. über die Option `-i` ansprechbar. (Nicht relevant bei Extended ASCII)

Attribute, Zeilenlineale, Steuerzeilen

Siehe dazu die Liste unter ISO 6937 Level 1 (Kap. 9.3.5).

9.3.2 TELEX

Es wird ein TELETEX-Dokument erstellt, das nur Zeichen enthält, die für TELEX gültig sind. Es handelt sich also nicht um eine Konvertierung, die strenggenommen dem Protokoll CCITT Nr. 5 (dem internationalen TELEX-Code) entspricht.

Zeichenvorrat

Zulässig sind:

a-z (0x61 - 0x7A)
0-9 (0x30 - 0x39)

sowie folgende Sonderzeichen:

Zeichen	Name	Sedezimaler Wert
Space	Leerzeichen	0x20
'	Accent aigue	0x27
(	Klammer auf	0x28
)	Klammer zu	0x29
+	Plus	0x2B
,	Komma	0x2C
-	Bindestrich	0x2D
.	Punkt	0x2E
/	Slash	0x2F
:	Doppelpunkt	0x3A
=	Gleichheitszeichen	0x3D
?	Fragezeichen	0x3F

Alle übrigen Zeichen werden mit einer entsprechenden Warnung durch Leerzeichen ersetzt.

Eine Ausnahme hiervon bilden deutsche Umlaute und einige Sonderzeichen. Sie werden bei der Konvertierung in eine Ersatzdarstellung gebracht:

Zeichen	Ersatzdarstellung
Ä bzw. ä	ae
Ö bzw. ö	oe
Ü bzw. ü	ue
ß	ss
"	'
! bzw.;	.
* bzw. &	+
%	%

Attribute, Zeilenlineale und Steuerzeilen

Siehe ISO-6937 Level 1 (Kap 9.3.5).

Hinweise:

- Aufgrund der Ablage als TELETEX (T.61) Dokument, müssen auch die Einschränkungen, die für das TELETEX-Protokoll gelten, mitberücksichtigt werden (z.B Seitenformate).
- Zum Erstellen eines TELEX sollte die mitgelieferte Tastatur-belegungs-Tabelle TELEX verwendet werden, da sonst Zeichen verlorengehen, die für die Lesbarkeit notwendig sind (z.B. alle Großbuchstaben).

9.3.3 Basic-Teletex (T.61)

Basic-Teletex ist in dem Papier "Rahmenwerte für Teletex Endgeräte" (FTZ 111 1R 293 1) im Kapitel 7 "Zeichenvorrat Codierung" beschrieben.

Zeichenvorrat

TELETEX (T.61) Zeichenvorrat.
Die zusätzlich im HIT verfügbaren Zeichen (ISO 6937 + HIT-Erweiterungen, u.a. { und }) werden mit Warnung ausgefiltert.

Attribute, Steuerzeilen, Zeilenlineale

Siehe ISO-6937 Level 2 (Kap. 9.3.5).

Seitenformat

Es sind folgende Seitenformate im internationalen Schriftverkehr üblich:

- DIN A4-Format (210 x 297 mm)
- Nordamerikanisches Format (216 x 280 mm)

Daraus leitet sich für Basic-Teletex folgendes Format ab, das beide Formate beinhaltet.

TTX-Basisformat (210 x 280 mm)

Zum Seitenformat siehe auch:

- SINIX-TTX-PACK V2.0 A3.1
- Teletex Rahmenwerte Kap. 7.3.3.3.4 Seite 246
- Teletex Rahmenwerte Kap. 7 ANNEX E Seite 271
- Teletex Rahmenwerte Kap. 8.3.3 Seite 278
- ISO-6937 Kap. 9 b)

Beachten Sie bitte:

- Header (UTC) sind Teletex-Paket spezifisch, und können deshalb nicht ausgewertet werden.

- Des weiteren können die Zusatzspalten (Anzahl - abhängig von der Schreibdichte (Zeichenbreite) - minimal 5, maximal 7), die bei TELETEX-Dokumenten am linken Rand zusätzlich zum Textbereich (Bezug T63-Protokoll: "Provisions for Verification of TELETEX Terminal Compliance") zur Verfügung stehen, normal angesprochen werden. Bei der Konvertierung nach Basic-Teletex werden alle HIT-Zeichen in den ersten 5 bzw. 7 Spalten als solche Zeichen in den Zusatzspalten inter-pretiert. In der erstellten Datei steht dann die entsprechende Anzahl von Backspace-Zeichen (Hexadezimal 0x08) vor dem eigentlichen Text. Um das zu vermeiden, empfiehlt es sich, bei der Erstellung eines TTX-Dokumentes im HIT mit einem Zeilenlineal zu arbeiten, das einen linken Rand von mindestens 5 bzw. 7 Spalten besitzt.

9.3.4 ISO 8859

Mit dem Format ISO 8859 können momentan 4 verschiedene Zeichensätze angesprochen werden. Sie bestehen aus einem 7-Bit-ASCII-Zeichensatz und den 8-Bit-Erweiterungen der ISO 8859-Drafts Latin Alphabet I, II, III und IV. Die im Menü `Konvertierungssteuerung` mögliche Eingabe für die `Draft`-Nummer (1 bis 4) entspricht dieser Einteilung.

Zeichenvorrat

Die 8-Bit-Erweiterungen zu Latin Alphabet I und II entnehmen Sie bitte dem Anhang.

Zeichen aus dem HIT- bzw. dem ISO 8859-Zeichensatz, die im jeweils anderen keine eindeutige Entsprechung besitzen, werden mit Warnung durch Leerzeichen ersetzt. Dabei ist vor allem Vorsicht bei dem Umgang mit verschiedenen Drafts des ISO8859-Formats geboten, da in den verschiedenen Drafts jeweils nur ein begrenzter Ausschnitt aus dem HIT-Zeichensatz zur Verfügung steht.

Das Sonderzeichen NBSP (No-Breaking-Space 0xa0) wird im HIT durch das geschützte Leerzeichen 0xa0 ersetzt und umgekehrt.

Das Sonderzeichen SHY (Soft-Hyphenation 0xad) wird im HIT durch das Attribut "Trennstelle" dargestellt und umgekehrt. Bei der Konvertierung in das HIT-Format wird beim Auftauchen eines SHY-Zeichens das diesem Zeichen vorangehende mit dem Attribut "Trennstelle" versehen. Entsprechend wird ein im HIT mit dem Attribut "Trennstelle" versehenes Zeichen bei der Konvertierung in das ISO 8859-Format aufgelöst in das nicht attributierte Zeichen, gefolgt von einem SHY-Zeichen.

Attribute, Steuerzeilen, Zeilenlineale

Für die Kontrollsequenzen und das Layoutformat werden die Werte des Conformance-Levels 1 (entspricht ISO 646) herangezogen.

9.3.5 ISO 6937

ISO-6937 ist in den Papieren ISO-6937/1 ISO-6937/2 und ISO-6937/3 beschrieben.

Zeichenvorrat

Siehe ISO-6937/2, zugehörige Tabelle im Anhang.

Zusätzliche, nur in HIT verfügbare, Zeichen (HIT-Erweiterungen, u.a. Liniengraphik) werden mit Warnung ausgefiltert.

Attribute, Zeilenlineale, Steuerzeilen

ISO-6937 definiert 3 "Conformance Levels", welche verschiedene Stufen der Textlayout-Funktionen bieten.

Level 1 — Entspricht den Controll Funktionen, die in ISO-646 definiert sind.

Level 2 — Entspricht den Controll Funktionen, die in T.61 definiert sind.

Level 3 — Entspricht dem vollen Umfang der in ISO-6937 definierten Controll Funktionen.

(Erläuterungen siehe ISO-6937/3 Kap. 9)

Zusätzlich gibt es ein **Level 0** für die Anwendung "electronic mail", siehe auch Shell-Option -L0.

Level 0 — Hier werden nur die Steuerzeichen CR (Carriage Return) und LF (Line Feed) unterstützt.

Zuordnung der HIT-Steuersymbole zu Steuersequenzen, wie sie im Protokoll ISO6429 definiert sind.

HIT-Attribut	Level 0/1	Level 2	Level 3
US	-	SGR 4	SGR 4
FE	-	-	SGR 1
HO	-	PLU	PLU
TI	-	PLD	PLD
DG	-	-	SGR 9
KU	-	-	SGR 3
FELD	-	-	-
NUMERISCH	-	-	-
REBÜND.	-	-	-
EINGABEZW.	-	-	-
TRENN(INS/DEL)	-	-	- <1>
Steuerzeilen			
Zeilenabstand			
0.25	-	SVS 3	SVS 3
0.5	-	SVS 3	SVS 3
0.75	-	SVS 4	SVS 4
1.0	-	SVS 0	SVS 0
1.5	-	SVS 1	SVS 1
2.0	-	SVS 2	SVS 2
2.5	-	-	-
3.0	-	-	-
3.5	-	-	-
Zeichenbreite			
10	-	-	SHS 0
12	-	-	SHS 1
15	-	-	SHS 2
Umbruch			
n	n mal CR FF<3>	n mal CR FF	n mal CR FF
Sonstige	-	-	-

Zeilenlineale	Level 0/1	Level 2	Level 3
Neue Seite Linealsymb.	CR FF <3> -	CR FF -	CR FF -
Sonstiges			
Seitenformate	-	PFS 0/1	PFS 0-9 <2>
gesch. Leerz. Trennstrich Zeilenende Absatzmarke	0x20 0x2D CR LF CR LF	0x20 0x2D CR LF CR LF	0x20 0x2D CR LF CR LF

<1> Bei Trennattributen ist zu beachten, daß damit unter Um-ständen auch die Informationen intelligenter Trennprozessoren, die zusätzlich zum Text vorhanden sind, (z.B wie ein 'ck' zu trennen ist) verloren gehen.

<2> Für Level 1 gibt es keine besonderen Seitenformate, für Level 2 sind laut Basic-Teletex nur 2 Formate zulässig, Level 3 läßt 10 verschiedene Seitenformate zu (siehe ISO-6937 Kap. 7.2.1 Seite 26). Da in HIT keine verschieden Seitenformate existieren, wird bei Level 1 keine Information über das Seitenformat abgelegt, bei Level 2 nur der Defaultwert PFS 0 (entspricht Carriage Return und Formfeed) und bei Level 3 der Wert PFS 2 (DIN A4 hoch).

<3> Bei Level 0 wird das Formfeed-Zeichen für Seitenende unterdrückt.

9.3.6 Alte HIT-Formate

Zu den früheren HIT-Formaten zählen Dateien der HIT-Versionen 2.0, 2.1, 3.0 und 3.1.

Für die Versionen 2.0 und 2.1 werden zwei Dateien erzeugt, eine Zeilenlinealdatei *.name*.Z und das Dokument *name*. Das Ergebnis kann nicht auf die Standardausgabe geschrieben werden (wegen der zwei Dateien).
Mit den Optionen `-V2.0`, `-V2.1` und `-V2A` kann gesteuert werden, ob eine Datei im Format der HIT-Version 2.0 bzw. 2.1 erstellt werden soll.

Zeichenvorrat

Siehe Zeichenbehandlung bei ISO-646-Format.

Zusätzlich werden für Version HIT-V2.1 folgende Zeichen tabellengesteuert erzeugt:

0x02 - 0x08
0x0A - 0x1A
0x1C - 0x1F

Diese Zeichen wurden bei V2.1 sprachspezifisch verwendet.

Das Tabulatorzeichen wird durch die entsprechende Anzahl von Leerzeichen ersetzt.

Zeichen, die in einem der alten HIT-Formate nicht existieren, werden mit einer entsprechenden Warnung durch ein Leerzeichen ersetzt.

Bei Version 2.0 wird nach 80 Zeichen, bei den Versionen 2.1, 3.0 bzw. 3.1 nach 132 Zeichen ein Zwangs-Zeilenumbruch erzeugt, da in diesen Versionen die maximale Zeilenbreite auf die angegebenen Werte beschränkt war.

Attribute

Attribute werden, wenn sie in den alten Formaten bereits vorhanden waren, entsprechend konvertiert und ansonsten mit Warnung übergangen.

Zeilenlineale

Die Zeilenlinealmarken werden in entsprechende alte Marken umgesetzt. Der "rechtsbündige" sowie der "zentrierte" Tabulator werden in einen alten Tabellentabulator umgesetzt. Der Dezimaltabulator sowie die Tausender-Trennstelle werden bei den Versionen 2.0 bzw. 2.1 mit Warnung ausgefiltert.

Steuerzeilen

Für die Version 3.0 und 3.1 werden alle Steuerzeilen übernommen, außer denen für Anwendungen, Druckaufbereitung und Stichwortregister.

Die Steuerzeilen für Zeichnung aus HIT-COL V3.1 werden in HIT-COL V4.0-Format konvertiert. Eine Rückkonvertierung in HIT-COL V3.1-Format ist nicht möglich.

Für die Versionen 2.0 und 2.1 gelten folgende Konvertierungsregeln:

< Umbruch:*zahl* >	ergibt *zahl* Neue-Seite-Lineale + 1 Leerzeile.
< Umbruch:ein >	wird zu einem Klammerlineal (falls nicht zufällig an der aktuellen Stelle bereits ein Lineal steht, wird ein zusätzliches eingefügt), das nur bis zum nächsten Lineal gilt, d.h ein geschützter Umbruch über mehrere Linealbereiche geht verloren.
< Umbruch:aus >	wird zu einem Zeilenlineal, das den klammernden Bereich abschließt, falls seit der Steuerzeile < Umbruch:ein > kein Zeilenlineal gesetzt war.

Sonstige Steuerzeilen werden mit einer entsprechenden Warnung ignoriert.

10 HIT-Werkzeuge

Mit den HIT-Werkzeugen kann der Systemverwalter Dokumente der HIT-Version 4.0 bearbeiten. Im Prinzip werden die HIT-Werkzeuge wie die entsprechenden SINIX-Werkzeuge für ASCII-Dateien eingesetzt. Die HIT-Werkzeuge setzen grundsätzliche Shell-Kenntnisse voraus.

Die Meldungen der HIT-Werkzeuge werden in der eingestellten Sprachvariante ausgegeben (siehe Shell-Variable `HITLANGUAGE`).

Welche HIT-Werkzeuge gibt es?

Folgende HIT-Werkzeuge stehen Ihnen zur Verfügung:

hsort	Sortieren von Zeilen
hindent	spaltenweises Verschieben
hchange	Änderung der Randmarken
hcon	Aneinanderhängen von HIT-Dokumenten
huntrenn	Löschen von Trennstellen
hgrep	Suchen von Mustern in HIT-Dokumenten
hdiff	Vergleichen von HIT-Dokumenten
hnum	Zeilen numerieren
hsed	Stream Editor für HIT-Dokumente
hfile	Dateityp ermitteln
huniq	mehrfache Zeilen einmal ausgeben
hwc	Zählen von Wörtern und Zeilen

Bitte beachten Sie:

Mit Ausnahme von `hfile` und `hwc` erfolgt die Ausgabe aller HIT-Werkzeuge im HIT-Format. Obwohl die Ausgabe standardmäßig auf die Standardausgabedatei `stdout` erfolgt, kann diese im allgemeinen nicht gelesen werden. Die Ausgabe kann allerdings mit SINIX-Mitteln in eine beliebige Datei umgelenkt bzw. über "pipe" an ein anderes Programm weitergegeben werden (z.B. `hless`).

10.1 Sortieren von HIT-Dateien

Wenn Sie Zeilen alphabetisch sortieren möchten, dann benutzen Sie `hsort`. Dieses Programm steht Ihnen standardmäßig über die Funktion **Markieren und Bearbeiten** / **ext-bearbeiten** zur Verfügung. Die Sortierreihenfolge ist im HIT-Benutzerhandbuch beschrieben (Kapitel 3.9.4.8).

Syntax

hsort [-c] [-v] [-b] [-e *errdat*] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

c *zahl* Nummer der zu sortierenden Spalte (Standardwert: 1).

v *zahl* Nummer der ersten zu sortierenden Zeile (Standardwert: 1).

b *zahl* letzte zu sortierende Zeile (Standardwert: letzte Zeile).

datei1 Name der Eingabedatei (Standardwert: `stdin`).

a *datei2* Ausgabedatei (Standardwert: `stdout`).

e *errdat* Fehlerdatei (Standardwert: `stderr`).
Wird keine Fehlerdatei angegeben, so werden aufgetretene Fehler über das Modul `hless` angezeigt.

Bitte beachten Sie:

Die Zeilenanzahl des zu sortierenden Dokuments ist auf 15.000 Zeilen beschränkt.

10.2 Spaltenweises Verschieben

Mit diesem Programm können Zeilen des angegebenen HIT-Dokuments spaltenweise nach rechts oder links verschoben werden. Zeilenlineale werden entsprechend angepaßt.

Syntax

hindent -r *zahl* -l *zahl* [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

r *zahl* Verschieben um *zahl* Zeichen nach rechts.

l *zahl* Verschieben um *zahl* Zeichen nach links.

datei1 Name der Eingabedatei (Standardwert: `stdin`).

a *datei2* Ausgabedatei (Standardwert: `stdout`).

Es kann nur jeweils eine der Optionen `r` oder `l` angegeben werden.

Achtung:

- Beim Einrücken (Verschieben nach rechts) wird der Zeilenanfang mit Leerzeichen aufgefüllt; Zeichen die nicht mehr in die Zeile passen, werden abgeschnitten.
- Beim Ausrücken werden führende Zeichen (auch Nicht-Leerzeichen) entfernt.

10.3 Ändern von Randmarken

Die Randmarken der Zeilenlineale des angegebenen HIT-Dokuments werden entsprechend den Argumenten nach rechts oder links verschoben. Der unter den Zeilenlinealen stehende Text wird nachformatiert.

Syntax

hchange [-l *zahl1*] [-r *zahl2*] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

l *zahl1* Verschieben der linken Randmarke um *zahl1* Zeichen nach rechts (negative und positive Zahlen sind zugelassen).

r *zahl2* Verschieben der rechten Randmarke um *zahl2* Zeichen nach rechts (negative und positive Zahlen sind zugelassen).

datei1 Name der Eingabedatei (Standardwert: `stdin`).

a *datei2* Name der Ausgabedatei (Standardwert: `stdout`).

Beispiel:

```
hchange -l -5 -r 3 -a text.out text
```

Alle Zeilenlineale des HIT-Dokuments `text` werden folgenderweise geändert:

Die linke Randmarke wird um 5 Spalten nach links verschoben, die rechte Randmarke um 3 Spalten nach rechts. Der Text des Dokuments wird nachformatiert.

Die Ausgabe erfolgt in die Datei `text.out`.

10.4 Aneinanderhängen von HIT-Dateien

Mit dem Modul `hcon` können eine oder mehrere HIT-Dateien aneinandergehängt werden. Die Ausgabe von `hcon` ist ein unformatiertes HIT-Dokument.

Darüber hinaus lassen sich formatierte Dokumente als unformatierte kennzeichnen(!) oder Formulare in unformatierte Dokumente überführen.

Syntax

hcon [-n] [-a *datei2*] [*datei1 ...*]

Folgende Aufrufoptionen sind möglich:

n	Bewirkt, daß nach jeder Datei ein Zeilenlineal vom Typ "Neue Seite" eingefügt wird.
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`).

10.5 Löschen von Trennstellen

Es werden alle (sowohl benutzte als auch mögliche) Trennstellen in einer HIT-Datei entfernt. Trifft `huntrenn` auf getrennte Wörter, die nach dem Entfernen der Trennstellen nicht in eine Zeile passen (weil z.B. das Zeilenlineal schmäler als das Wort ist), dann wird an dieser Stelle die Trennstelle nicht entfernt.

Syntax

huntrenn [-n] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

n	benutzte Trennstellen nicht löschen.
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`).

10.6 Suchen von Mustern in HIT-Dateien

Das Modul `hgrep` sucht in den angegebenen HIT-Dateien nach dem jeweiligen Muster. Im Prinzip arbeitet `hgrep` wie das SINIX-Dienstprogramm `fgrep`.

Attribute, Zeilenlineale und Steuerzeilen werden bei der Suche nicht berücksichtigt.

Syntax

hgrep [-ACchlmnvxy] [[-e] *muster*] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

A — Ausgabe erfolgt im ASCII-Format (Achtung: Nicht-ASCII-Zeichen werden nicht korrekt ausgegeben!).

C *vergleich* — Gibt an, wie Suchtext und der Inhalt der Steuerzeile miteinander verglichen werden soll. Der Vergleich erfolgt lexikographisch.
Mögliche Werte sind:

Operator	Bedeutung
=	Suchtext gleich Steuerzeile
< > oder ! =	Suchtext ungleich Steuerzeile
< =	Suchtext kleiner oder gleich Steuerzeile
> =	Suchtext größer oder gleich Steuerzeile
<	Suchtext kleiner Steuerzeile
>	Suchtext größer Steuerzeile
in oder *	Steuerzeile enthält den Suchtext

c — Anzahl der Zeilen ausgeben, die das Muster enthalten.

h — Beim Durchsuchen mehrerer Dateien werden die Dateinamen nicht mit ausgegeben.

l — Namen der Dateien ausgeben, die das Muster enthalten.

m *zahl* — Es wird nur nach Steuerzeile, die durch *zahl* angegeben ist, gesucht. Textzeilen werden nicht berücksichtigt.

zahl steht für eine Steuerzeile. Die Zuordnung können Sie der folgenden Tabelle entnehmen.

Name der Steuerzeile	Nummer der Steuerzeile
Zeilenabstand	11
Kopf	05
Seitennummer	08
Graphik	03
Schachtwechsel	09
Umbruch	10
Fuß	02
Kapitelnummer	04
Zeichenbreite	13
Druckformular	14
Anwendung	17
Fußnote	01
Stichwort	18
Schriftart	12
Druckaufber.	19
***	15
Zeichnung	16

n — Zeilennummer vor jeder Zeile ausgeben.

v — Ausgabe aller Zeilen, die das Muster nicht enthalten.

x — Alle Zeilen ausgeben, die nur das Muster enthalten. Leerzeichen am Anfang der Zeile werden mit berücksichtigt!

y — Groß- und Kleinbuchstaben werden gleich bewertet.

e — Ist anzugeben, wenn das Muster mit dem Zeichen "-" beginnt, z.B sucht `hgrep -e -abc` alle Zeilen die das Muster "`-abc`" enthalten.

muster — eine Zeichenfolge, nach der `hgrep` jede Zeile durchsucht.

datei1 — Name der Eingabedatei (Standardwert: `stdin`). Wenn mehrere Dateien angegeben sind, wird vor jeder gefundenen Zeile der Dateiname mit ausgegeben.

a *datei2* — Name der Ausgabedatei (Standardwert: `stdout`).

10.7 Vergleichen von HIT-Dateien

Dieses Programm vergleicht zwei HIT-Dateien miteinander. Beim Vergleichen werden alle Zeilentypen untersucht (Textzeilen, Zeilenlineale und Steuerzeilen) und die Zeilen, in denen sich die Dateien unterscheiden, ausgegeben.

Syntax

hdiff [-a *datei*] *datei1* [*datei2*]

Wenn *datei2* nicht angegeben ist, wird von der Standardeingabe `stdin` gelesen, wenn -a *datei* nicht angegeben ist, wird auf die Standardausgabe `stdout` geschrieben.

10.8 Zeilen numerieren

Dieses Programm versieht alle Textzeilen einer HIT-Datei mit Zeilennummern. Diese Nummern werden an den Beginn jeder Textzeile eingefügt.

Syntax

hnum [-m] [-z] [-a *datei2*] [*datei1*]

Standardmäßig werden nur die Textzeilen behandelt und gezählt.

Folgende Aufrufoptionen sind möglich:

m	Steuerzeilen werden mitgezählt.
z	Zeilenlineale werden mitgezählt. In beiden Fällen werden die betreffenden Nummern nicht in die Ausgabe-Datei geschrieben.
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`).

Hinweis:

Bei der Numerierung mit der Option -2 werden Zeilenlineale durch Zeilenlineale ohne Marken ersetzt.

10.9 Stream-Editor für HIT-Dateien

Mit `hsed` können in HIT-Dokumenten bestimmte Zeichenfolgen gegen andere ausgetauscht werden.

Syntax

hsed [-n] [[-e] *skript*] [-f *skriptdatei*] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

n	Es werden nur die Zeilen ausgegeben, welche der Benutzer in der Skriptdatei angegeben hat. Andernfalls werden alle eingelesenen Zeilen wieder ausgegeben.
e *skript*	Kommando (Skript)
f *skriptdatei*	Die Skriptdatei ist eine ASCII-Datei, keine HIT-Datei!
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`).

Achtung:

- Es werden alle Zeilentypen (Textzeilen, Zeilenlineale und Steuerzeilen) bearbeitet, deshalb sollte man Manipulationen wie `s/1/111/` unterlassen (diese Manipulation würde in einem Zeilenlineal mehrere linke Ränder erzeugen!).

- Bei der Angabe der Option `-n` werden nur solche Zeilen ausgegeben, welche im Skript bzw. in der Skriptdatei angegeben werden. Es kann dabei auch eine Datei entstehen, die kein einziges Zeilenlineal mehr enthält (sie ist aber normalerweise nicht durch ein Kennzeichen (Flag) im Dokumentkopf (Header) als formatfrei gekennzeichnet).

10.10 Dateityp ermitteln

Dieses Programm arbeitet wie das UNIX-Dienstprogramm `file`. Zusätzlich wird unterschieden, ob es sich bei den Dateien um HIT-Dateien handelt.

Syntax

hfile *datei*

`hfile` gibt bei HIT-Dateien einen der folgenden Dateitypen aus:

- HIT-Datei
- HIT-Datei ohne Zeilenlineale
- formatierte HIT-Datei
- formatierte HIT-Datei ohne Zeilenlineale
- in Proportionalschrift aufbereitete HIT-Datei
- HIT-Formular
- HIT-Formular ohne Zeilenlineale
- HIT-Umgebungstabelle
- HIT-Makrotabelle
- HIT-Sicherungsdokument bzw. Paging-Datei

10.11 Mehrfache Zeilen einmal ausgeben

Dieses Programm arbeitet auf einer sortierten HIT-Datei (siehe `hsort`). Im Standardfall werden alle einfachen Zeilen sowie von allen wiederholten Zeilen jeweils eine ausgegeben. Wie bei `hsort` werden nur die Textzeilen berücksichtigt.

Syntax

huniq [-d] [-u] [-a *datei2*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

d	Es werden nur die mehrfachen Zeilen ausgegeben.
u	Es werden nur die einfach vorkommenden Zeilen ausgegeben.
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`).

10.12 Zählen von Wörtern und Zeilen

Dieses Programm ermittelt die Zahl der Wörter, Textzeilen, Steuerzeilen, Lineale und die Länge des längsten Linealbereichs.

Syntax

hwc [-w] [-t] [-l] [-m] [-z] [-a *datei*] [*datei1*]

Folgende Aufrufoptionen sind möglich:

w	Anzahl der Wörter ausgeben (Standard).
t	Anzahl der Textzeilen ausgeben (Standard).
l	Anzahl der Zeilenlineale ausgeben.
m	Anzahl der Steuerzeilen ausgegeben.
z	Länge des längsten Linealbereichs ausgeben.
datei1	Name der Eingabedatei (Standardwert: `stdin`).
a *datei2*	Name der Ausgabedatei (Standardwert: `stdout`). Die Ausgabe der gewünschten Information erfolgt als ASCII-, nicht als HIT-Datei!

11 Die Systemarchitektur

Dieser Abschnitt wendet sich an all diejenigen, die die interne Struktur des HIT-Systems kennenlernen wollen. Sie werden damit in die Lage versetzt, evtl. auftretende Systemfehler zu analysieren und zu beheben.

Die Beschreibung basiert auf der HIT-Version für Anlagen mit X/OPEN-Umgebung. Auf die HIT-Version unter dem Betriebssystem SINIX V2.0/2.1 wird hier nicht eingegangen; für sie gilt die Beschreibung analog.

Zum Verständnis sind außer der Beherrschung des HIT gute SINIX-Kenntnisse dringend erforderlich.

Für die Verwendung von HIT-COL benötigen Sie zusätzliche Kenntnisse des COLLAGE-Systems.

11.1 Das HIT-System und seine Komponenten

Entsprechend der UNIX-Philosophie der "tools" ist das gesamte HIT-System modular aufgebaut, d.h. die Funktionen werden jeweils einem Modul zugeordnet.

Da das HIT-System sehr umfangreich ist, wird im Dateiverzeichnis `/usr/bin` ein eigenes Dateiverzeichnis `HIT` eingerichtet, in dem sich alle Module befinden.

Um mit HIT arbeiten zu können, wird die Shell-Variable $PATH in der Datei `/etc/profile` entsprechend geändert. Benutzer, die sich einen eigenen Suchpfad definiert haben, müssen diese Shell-Variable auswerten (z.B. `PATH=$PATH:/usr/gast/bin`).

11.1.1 Struktur der Programme unter /usr/bin/HIT

clou	*	Bausteinverarbeitungs-Programm CLOU-Anweisungen (beginnend mit dem Nummernzeichen #) in Textbausteinen werden beim Einfügen vom CLOU interpretiert.
cloudaemon		Spoolsystem für Ausdruck der mit `clouhg` erstellten Dokumente Es wird dafür gesorgt, daß diese auch ausgedruckt werden, wenn die Anlage zwischenzeitlich abgeschaltet wurde.
clouhg	*	Verarbeitung von Schreibaufträgen (Textverarbeitung im Hintergrund) Die aufgerufenen Bausteine erscheinen nicht mehr als Text am Bildschirm, sondern werden im Hintergrund zusammengesetzt. Die Weiterverarbeitung des Dokuments (Formatieren, Druck) erfolgt ebenfalls im Hintergrund.
col_import		liest den COLLAGE-Zwischenspeicher.
fedor	*	Formatiertabellen-Menü Menü-Oberfläche für die Erstellung von Formatiertabellen.
filter	*	Druckaufbereitung Die in einer HIT-Datei enthaltenen HIT-spezifischen Steuerzeichen (Seitenvorschub, Unterstreichen, Fettdruck usw.) werden in druckerspezifische Steuerzeichen umgewandelt.
fit	*	Benutzeroberfläche für das HIT-System (HIT-Menü) Verwaltung der Dokumente, Aufruf von Funktionen.
fitlpr	*	Unterprogramm des `fit` Bereitet die Ausgabe des SINIX-Kommandos `lpr -q` auf und ermöglicht das Löschen abgesetzter eigener Druckaufträge. Darüber hinaus können gesperrte Drucker freigegeben werden.

fontcc * Fontcap-Compiler
Die mit dem Editor erstellten Dicktentabellen werden in eine ablauffähige (binäre) Form gebracht.

formdoc * Dokumentformatierer
Das erfaßte Rohdokument wird infolge der Anweisungen aus der Formatiertabelle in Seiten eingeteilt, Variablen werden ersetzt, Kapitelnumerierung durchgeführt, Inhaltsverzeichnis erstellt usw.

hfind * Suchmenü
Aufgerufen von `fit`, können Suchfunktionen nach Dokumenten und Dokument-Inhalten ausgeführt werden.

hform * Formular-Editor
Dient zum Ausfüllen der mit dem Modul `hit` erstellten Formulare.

hidx * Stichwort-Generator
Erstellt - gesteuert von `formdoc` - das Stichwort-Verzeichnis für ein Dokument.

hindex * Kapitel-Übersicht
Angesprochen über die Funktionstaste SHIFT + F13 **Kapitel-Übersicht** werden die Überschriften des gerade bearbeiteten Dokuments zur Auswahl und anschließenden Positionierung angeboten.

hit * Editor
Hier werden alle Funktionen der Textbearbeitung zur Verfügung gestellt.

hitsb * Serienbriefgenerator
Aus einem Musterbrief und den in einer Adreßdatei enthaltenen Daten werden Dokumente mit variablem Inhalt erzeugt.

hless * Dieses Programm dient der Anzeige von HIT-Dateien auf dem Bildschirm.

hstat Ausgabeprogramm für anzuzeigende Bausteine
Wird vom CLOU benützt, wenn für Bausteinaufrufe Suchmuster verwendet wurden.

mador * Makro-Editor
Es können statische und dynamische Makrotabellen bearbeitet werden.

printcc * Printcap-Compiler
Die mit dem Editor erstellten Druckertabellen werden in eine ablauffähige (binäre) Form gebracht.

procon * Format-Konvertierer
Dateien bestimmter Datenformate werden in Dateien anderer Datenformate konvertiert. Folgende Konvertierungen stehen zur Verfügung:

HIT4 ↔ ISO-6937
HIT4 ↔ Basic-Teletex
HIT4 ↔ ISO-646
HIT4 ↔ TELEX
HIT4 ↔ frühere HIT-Formate
HIT4 ↔ ISO-8859

proform * Absatz-Formatierer/Druckaufbereitung
Vom Editor `hit` aufgerufen, erledigt `proform` die Absatz-Formatierung. Zur Druckaufbereitung in Proportionalschrift werden Absätze unter Verwendung von Dicktentabellen (`fontcap`) formatiert.

redraw stellt Zeichnungen, die über die Steuerzeile `Zeichnung` in ein HIT-Dokument eingefügt wurden, während der Dokumentbearbeitung am Bildschirm dar. `redraw` erwartet Dateien im COLLAGE-Metafile-Format, deren Name auf `.drw` endet. (SIDRAW-Format)

tru-D		Trennprogramm Silbentrennung nach Duden. Für den gewünschten Textbereich werden alle gefundenen Trennstellen markiert. Evtl. falsch getrennte Wörter können für zukünftige richtige Trennung in eine Ausnahmedatei aufgenommen werden.
udor	*	Umgebungstabellen-Menü Oberfläche zur Erstellung von Umgebungstabellen.

HIT-Werkzeuge

hcalc	Rechenprogramm Aufgerufen über **Markieren und Bearbeiten / ext-bearbeiten (rechnen)** können die Grundrechenarten auf in HIT erfaßte Zahlen angewandt werden.
hchange	Ändern von Zeilenlinealen Im gesamten bearbeiteten Dokument können die linken bzw. rechten Randmarken verschoben werden.
hcfilt	erzeugt das Format des COLLAGE-Zwischenspeichers.
hclip	schreibt/liest den externen Zwischenspeicher.
hcon	Zusammenführen von HIT-Dokumenten HIT-Dokumente können aneinandergehängt werden. Darüber hinaus können aus formatierten Dokumenten und Formulare normal bearbeitbare Dokumente gemacht werden.
hdiff	Vergleich von HIT-Dokumenten Es werden alle Unterschiede zwischen zwei HIT-Dokumenten ausgegeben.
hfile	Dokument-Typ ermitteln Es kann festgestellt werden, ob es sich bei einem HIT-Dokument um ein Formular, ein Formatiertes Dokument, einen Formatfreien Baustein usw. handelt.

hgrep	Suchprogramm für HIT-Texte Analog dem SINIX-Kommando `grep` kann in HIT-Dokumenten nach Zeichen(folgen) gesucht werden. Wird aufgerufen vom Modul `hfind`.
hindent	Verschieben des Textes Der gesamte Text eines HIT-Dokuments kann verschoben werden, um den linken Rand breiter oder schmäler zu machen.
hnum	Numerieren von Zeilen Alle Zeilen eines HIT-Dokuments werden mit Zeilennummern versehen.
hsed	Austausch von Zeichenfolgen In einem HIT-Dokument können bestimmte Zeichenfolgen gegen andere ausgetauscht werden.
hsort	Sortierprogramm Analog dem SINIX-Kommando `sort` können HIT-Texte sortiert werden. Wird aufgerufen von **Markieren und Bearbeiten / ext-bearbeiten (sortieren)**.
huniq	Mehrfache Zeilen suchen Zeilen, die in einem HIT-Dokument mehrfach vorkommen, werden nur einmal ausgegeben.
huntrenn	Trennstellen löschen In einem HIT-Dokument werden alle vorhandenen Trennstellen gelöscht.
hwc	Wörter und Zeilen zählen Die Zahl der Wörter und Zeilen in einem HIT-Dokument wird ausgegeben.

Um die Übersetzbarkeit der Bedienoberfläche in andere Landessprachen zu vereinfachen, wurden sämtliche Meldungstexte sowie die für den Benutzer sichtbare sprachliche Oberfläche aus den eigentlichen Programmen ausgelagert. Programme, die Meldungstexte benötigen, sind mit einem * gekennzeichnet.

Um darüber hinaus auch noch eine größtmögliche Hardware-Unabhängigkeit zu erreichen, werden für die Programme `hit` und `fit` die Bildschirmausgabe über eine HIT-eigene Bildschirmsteuerungs-Tabelle ("termcap") und die Tastatureingabe über eine Tastaturbelegungstabelle ("keycap") gesteuert. Für die Ablauffähigkeit dieser Module sind über die Meldungstext-Dateien hinaus auch noch diese Tabellen erforderlich.

Alle ausgelagerten Programmteile befinden sich unterhalb des Dateiverzeichnisses `/usr/lib/hit`.

Programme unter `/usr/bin`

Zusätzlich existieren unter `/usr/bin` drei Aufrufprozeduren, die alle benötigten Parameter zur Verfügung stellen (z.B. $PATH):

`HIT-COL`	ruft HIT-MENÜ auf
`hit-col`	ruft den HIT-Editor auf
`HLESS`	ruft `hless` auf (zum Anzeigen von HIT-Dokumenten)

11.1.2 Die Meldungstext-Dateien

Die Meldungstext-Dateien der einzelnen Module liegen in binärer Form vor. Bei jeder Ausgabe einer Meldung wird diese aus der Meldungstext-Datei gelesen. Im Dateiverzeichnis `/usr/lib/hit/messages` existiert für jedes einzelne Modul ein Unter-Dateiverzeichnis, in welchem sich die Meldungstexte befinden.

Der Standard-Name für die jeweiligen Meldungstexte ist `messages`. Der absolute Pfadname für die Meldungstexte lautet somit
`/usr/lib/hit/messages/hit/messages`,
`/usr/lib/hit/messages/formdoc/messages` usw.
Die Auslieferung der Meldungstext-Dateien erfolgt mit ihrem Nationalitätskennzeichen als Namen; ein Link auf den Namen `messages` wird bei der Installation durchgeführt.

Zusätzlich zu den Meldungstext-Dateien für die einzelnen Module existiert noch eine zentrale Meldungstext-Datei, die sich im Dateiverzeichnis `/usr/lib/hit/messages` befindet: `/usr/lib/hit/messages/messages`. Auf diese Datei wird von allen Modulen zugegriffen.

Zum Aufbau der Meldungstext-Dateien siehe unten.

Es ist möglich, zusätzlich zu den augenblicklich verwendeten Meldungstexten weitere Meldungstexte zu installieren, z.B. für Englisch, Französich, Italienisch usw. Diese Meldungstexte (die entsprechend ihrem Nationalitätskennzeichen benannt sind) können auf zweierlei Art und Weise angesprochen werden:

- Durch Aufruf des Moduls mit dem Schalter `-M` und der gewünschten Meldungstext-Datei.

  Beispiel:

  ```
  hit -M GB
  ```

  Auf die Angabe eines absoluten oder relativen Pfadnamens kann verzichtet werden, wenn sich die Meldungstexte in dem dem Modul entsprechenden Dateiverzeichnis (hier: `/usr/lib/hit/messages/hit`) befinden.

- Über die Shell-Variable `HITLANGUAGE`. Diese wird ausgewertet, wenn das betreffende Modul nicht mit dem Schalter `-M` aufgerufen wurde. `HITLANGUAGE` wird von allen Modulen ausgewertet. Damit ist es möglich, verschiedene Sprachoberflächen auf einer Anlage zu halten und benutzerkennungsspezifisch damit zu arbeiten.

  Beispiel

  Zusätzlich zum deutschen HIT sind die englischen und französischen Meldungstexte installiert. Durch Setzen der Shell-Variablen `HITLANGUAGE=GB` in seiner `.profile` kann ein Benutzer mit der englischen Sprachoberfläche arbeiten, während ein anderer Benutzer durch Eintrag der Shell-Variablen `HITLANGUAGE=F` in seiner `.profile` mit der französischen Sprachoberfläche arbeiten kann.

  Wenn Sie mit HIT V4.1 arbeiten, müssen Sie im `.profile` zusätzlich folgenden Eintrag vornehmen: `./usr/bin/HIT/HIT_env`. Bei HIT-COL V4.1 ist dieser Eintrag nicht erforderlich!

Achtung

Bei der Angabe einer Meldungstext-Datei für ein oder mehrere Module muß eine System-Meldungstextdatei (im Dateiverzeichnis `/usr/lib/hit/messages`) mit dem gleichen Namen existieren!

Falls ein Modul weder mit dem Schalter `-M` aufgerufen noch die Shell-Variable `HITLANGUAGE` gesetzt wurde, wird automatisch die Standard-Einstellung `messages` für die Meldungstext-Dateien verwendet.

Aufbau der Meldungstext-Dateien (original):

(Dieser Abschnitt dient nur der Erläuterung; die genannten Dateien werden nicht mit dem System ausgeliefert)

Um die Meldungstext-Dateien bearbeiten zu können, liegen sie in ihrer Original-Form als HIT-Dateien vor. Nach dem Bearbeiten müssen diese Dateien mit einem "Message-Compiler" in eine ablauffähige (binäre) Form gebracht werden.

Der Aufbau der Meldungstext-Datei entspricht dem Aufbau der Tabellen im HIT-System: Jeder Eintrag beginnt und endet mit einem Nummernzeichen # (falls nicht durch die Variable VAR_KLAMMER anders definiert). Unmittelbar nach dem einleitenden # folgt ein programminternes Schlüsselwort, das nicht verändert werden darf. Hinter dem sich anschließenden Doppelpunkt folgt der eigentliche Meldungstext. Dieser kann Anweisungen der Programmiersprache C wie %d, %s usw. enthalten. An ihre Stelle werden bei der Ausgabe des Meldungstextes Variablen (z.B. Dateinamen) eingefügt.

Beispiel 1

(Auszug aus der Meldungstext-Datei des Editors hit)

```
#ESC_ZEICHEN:@#.

================================================================================
                    SYSTEMZEILENTEXTE
================================================================================
#SYSTEX:Zeile:#
#FSYSTEX:Seite:#
#SYSBLK:BLK#
#SYSEFG:EFG#
#SYSUS:US#
#SYSFE:FE#
#SYSBR:DG#
#SYSKU:KU#
#SYSTI:TI#
#SYSHO:HO#
#SYSFD:FD#
#SYSEZ:EZ#
#SYSNU:NU#
#SYSRE:RE#
#SYSFORM:FELD#
#RDONLY:@
       NUR LESEN      #     19 Zeichen lang!!
#FORMATED:@
       FORMATIERT     #     19 Zeichen lang!!
#MEMOUT:@
       SPEICHER AUS   #     19 Zeichen lang!!
```

```
================================================================================
                    SUCHEN UND ERSETZEN
================================================================================
#A1_SEARCH:Suchtext:#
#A1_REPLACE:Ersatztext:#
#A1_COUNT:Ersetzungen:#
#A1_DIRECT:Vorwärts:#
#A1_QUIT:Bestätigen:#
#A1_FORMAT:Formatieren:#
#A1_HIGH:Groß und Klein:#
#A1_ATTR:Attribute relevant:#


================================================================================
                    FEHLERMELDUNGEN
================================================================================


--------------------------------------------------------------------------------
Meldungen beim Einlesen von Dokumenten: Einlesen des Textes.
--------------------------------------------------------------------------------
#DOKREAD:Dokument '%s' kann nicht gelesen werden!#
#BADHEAD:Header von Dokument '%s' nicht lesbar!#
#BADLINE:Unbekannter Zeilentyp in Dokument '%s'#
#BADPROCON:Fehler beim Konvertieren nach HIT-Format!#
#DOKTOOLONG:Dokument ist zu lang - konnte nicht ganz eingelesen werden!#
#NOCRRD:%s kann weder angelegt noch gelesen werden!#
#LINE_TOO_LONG:%d Zeilen in '%s' zu lang: wurden umgebrochen!#
#F_IN:Dokument wurde schon eingelesen - noch keine Veränderungen!#
#F_OPEN:%S '%s' kann nicht geöffnet werden!#
#NO_FILE:%S '%s' existiert nicht!#
#F_EXEC:%S '%s' ist ein ausführbares Programm!#
#F_DIR:%S '%s' ist ein Dateiverzeichnis!#
#F_BIN:%S '%s' enthält binäre Daten!#
#F_EMPTY:%S '%s' ist leer!#
```

```
================================================================================
                      ABFRAGEN UND HINWEISE
================================================================================


--------------------------------------------------------------------------------
Beim Bausteine Einfügen.
--------------------------------------------------------------------------------
#NBAU_NAME:Bitte Namen des Bausteins eingeben:#
#FORM_BAU:Formatierter Baustein %s wird eingefügt!#
#NORM_BAU:Unformatierter Baustein %s wird eingefügt!#
#BAU_REFORM:Eingefügter Baustein %s wird formatiert!#
#END_BAUSTEIN:Abbruch von Baustein Einfügen?#
#BAU_END:Baustein '%s' erfolgreich eingefügt!#
#DEFBINS:Bitte Variable eingeben!#
#CONT_BAUSTEIN:$
Um den Rest des Bausteines einzufügen, drücken Sie 'BAUSTEIN EINFÜGEN'!#
```

Beispiel 2

(Auszug aus der zentralen Meldungstext-Datei)

```
Hit - Systemtabelle

Die Systemtabelle enthält: Pfade Programmnamen Markupid's
(Steuerzeilen), Umgebungsvariablen (Namen der Variablen, die zum
Einlesen einer Umgebung immer gebraucht werden: ESC_ZEICHEN,
VAR_KLAMMER, FALSE, TRUE)

--------------------------------------------------------------------------------
Markupid's
--------------------------------------------------------------------------------

#M_FN:Fußnote#          Fußnote.
#M_FUSS:Fuß#            Fuß.
#M_KOPF:Kopf#           Kopf.
#M_ZLA:Zeilenabstand#   Zeilenabstand.
#M_PAGE:Seitennummer#   Seitennummer.
#M_KAP:Kapitelnummer#   Kapitelnummer.
#M_GRAPH:Graphik#       Graphik.
#M_UMBR:Umbruch#        Umbruchgrenze.
#M_SW:Schachtwechsel#   Schachtwechsel.

#M_ONUMBR:ein#          bedingter Seitenumbruch Anfang
#M_OFFUMBR:aus#         bedingter Seitenumbruch Ende
#M12_FONT:Schriftart#
#M13_CPI: Zeichenbreite#
#M14_MERGE: Druckformular#

--------------------------------------------------------------------------------
Umgebungsvariablen
--------------------------------------------------------------------------------

#A00_ESC:ESC_ZEICHEN#
#A00_VAR:VAR_KLAMMER#
#A00_TRUE:JA#
#A00_FALSE:NEIN#
```

```
================================================================================
Pfadnamen, Programmnamen, Shell-Variablen etc.
================================================================================

--------------------------------------------------------------------------------
Pfadnamen (PD_: Directories allg., PM_: Message-Directories)
--------------------------------------------------------------------------------

#PD_HITTMP:/usr/lib/hit/tmp#
#PD_HITRECOVERY:/usr/lib/hit/recovery#
#PD_DEF_TMP:/usr/tmp#

#PD_HELP_PATH:/usr/lib/hit/help-D#
#PD_KEY_PATH:/usr/lib/hit/keycap#
#PD_PKAP_PATH:/usr/lib/hit/printcap#
#PD_TERM_PATH:/usr/lib/hit/termcap#
#PD_UMG_PATH:/usr/lib/hit/umgebung-D#

#PD_SPOOL:/usr/lib/hit/spool#        Für CLOU im Hintergrund
#PD_PRINTER:/usr/lib/hit/printer#

#PM_PATH:/usr/lib/hit/messages#
#PM_HIT_PATH:hit#
#PM_CLOU_PATH:clou#
#PM_FILTER_PATH:filter#
#PM_FIT_PATH:fit#
#PM_FD_PATH:formdoc#
#PM_PROCON_PATH:procon#
#PM_FITSB_PATH:fitsb#
#PM_FITLPR_PATH:fitlpr#
#PM_HITSB_PATH:hitsb#
```

```
--------------------------------------------------------------------------------
Programmnamen (ggf. auch Aufrufe dieser Programme, spaeter)
(PP_: Programme, PC_: Aufrufe, spaeter)
--------------------------------------------------------------------------------

#PP_CLOU:clou#
#PP_CAT:hcat#
#PP_FORMDOC:formdoc#
#PP_PROCON:procon#
#PP_TRU:tru-D#
#PP_HIT:hit#
#PP_HITSB:hitsb#
#PP_FITSB:fitsb#
#PP_FITLPR:fitlpr#
#PP_CLSTAT:stat#
#PP_CLDAEMON:cloudaemon#

--------------------------------------------------------------------------------
Default Dateinamen (PDEF_: Defaults)
--------------------------------------------------------------------------------

#PDEF_MSGTAB:messages#
#PDEF_UTAB:hitumg#
#PDEF_TERMCAP:termcap#
#PDEF_KEYCAP:keycap#
#PDEF_FTAB:formtab#
#PDEF_FU1:prerror#
#PDEF_FU2:#

--------------------------------------------------------------------------------
Shell-Variablen (PSH_: ...)
--------------------------------------------------------------------------------

#PSH_TERM:HITTERM#              steuert Termcap-Suche (statt TERM oder TERMCAP)
#PSH_KEY:HITKEY#                steuert Keycap-Suche (statt KEY)
#PSH_SHELL:SHELL#               steuert Zugang zur Shell (gesetzt → sh erlaubt)
#PSH_HITUMGEBUNG:HITUMGEBUNG#   steuert Zugriff zur Hitumgebung
```

11.1.3 Die Systemtabellen

Um eine größtmögliche Unabhängigkeit von der Hardware zu erreichen, sind hardwarespezifische Steuerzeichen in Systemtabellen hinterlegt. Die Bildschirmausgabe erfolgt über eine HIT-eigene *Termcap*, die Tastatureingabe über eine sog. *Keycap* und die Druckerausgabe über eine sog. *Printcap*.

Ebenso wie die Meldungstext-Dateien liegen die Systemtabellen in binärer Form vor und werden beim Aufruf der Programme hinzugeladen.

11.1.3.1 Die HIT-Termcap

Die HIT-Termcap befindet sich im Dateiverzeichnis `/usr/lib/hit/termcap`. Die Standardeinstellung ist `termcap`. Benutzt wird die Termcap von den Modulen `hit` und `fit`. Falls nicht der Standardname *termcap* verwendet werden soll oder sich die Termcap in einem anderen Dateiverzeichnis befindet, so existieren wie bei den Meldungstext-Dateien zwei Möglichkeiten der Veränderung:

- Aufruf des Moduls mit dem Schalter `-T` und dem Namen der gewünschten Termcap. Falls die Termcap nicht mit absolutem oder relativem Pfadnamen angegeben wurde, wird standardmäßig auf die Termcap im Dateiverzeichnis `/usr/lib/hit/termcap` zugegriffen.
- Setzen der Shell-Variable `HITTERM`. Diese wird nur ausgewertet, wenn der Schalter `-T` nicht verwandt wurde.

Aufbau der Hit-Termcap (original)

(Dieser Abschnitt dient nur der Erläuterung; die genannten Dateien werden nicht mit dem System ausgeliefert)

Ebenso wie die Meldungstexte liegt auch die Termcap im edietierbaren HIT-Format vor und muß, um in eine ablauffähige Form gebracht zu werden, nach der Bearbeitung mit einem "Termcap-Compiler" compiliert werden.

Der Aufbau der HIT-Termcap ist analog dem der Meldungstext-Dateien. Die programm-internen Strings können in zwei Kategorien eingeteilt werden:

- Ansteuerung des Bildschirms (Cursorbewegungen, Zeichen/Zeile löschen usw.). Eingetragen sind jeweils die Escape-Sequenzen, die am Bildschirm die definierte Funktion auslösen.

- Ausgabe der Zeichen auf den Bildschirm. Da nicht alle vom Hit verwendeten Zeichen in einem Zeichensatz verfügbar sind, wird der meist verwendete Zeichensatz (ASCII int.) in den Bereitstellungsbereich G0, der Zeichensatz EURO in den Bereitstellungsbereich G1 geladen. Zwischen den Bereitstellungsbereichen G0 und G1 wird dann je nach Bedarf dynamisch hin- und hergeschaltet. Falls ein Zeichen in diesen Bereitstellungbereichen nicht verfügbar ist, so wird der entsprechende Zeichensatz in den Bereitstellungsbereich G1 nachgeladen.

  Diese Vorgehensweise bedingt, daß für jedes vom Editor verarbeitbare Zeichen der Zeichensatz, in dem es zu finden ist, sowie die Adresse in diesem Zeichensatz angegeben werden muß.

  Die Namen der einzelnen Zeichen setzen sich zusammen aus einem c_(für character), gefolgt vom programminternen Namen, der für Termcap, Keycap und Printcap gleich ist.

Beispiel

(Auszug aus der termcap)

```
#ATTRS_OFF:^[[Om#

#CURSOR_UP:^[[A#
#CURSOR_DOWN:^[[B#
#CURSOR_RIGHT:^[[C#
#CURSOR_LEFT:^[[D#

#CURSOR_MOTION:^[[%d;%dH#

#DELETE_CHAR:^[[P#
#INSERT_CHAR:^[[1#
#DELETE_LINE:^[[M#
#INSERT_LINE:^[[L#
#N_DELETE_CHAR:^[[%dP#
#N_INSERT_CHAR:^[[%d2#

#CHANGE_SCROLL:^[[%d;%dr#
#SCROLL_DOWN:^[[T#
#SCROLL_UP:^[[S#

#CLEAR_LINE:^[[2K#
#CLEAR_SCREEN:^[[2J#

#HIDE_CURSOR:^[[6p#
#SHOW_CURSOR:^[[7p#

#END_ALT_CHARSET:^O15z#
#STRT_ALTCHARSET:^O14z#

v.....................................................................v
init-terminal: NATIONAL A nach GO,
         EURO nach G1.

#INIT_TERMINAL:^[(B^[)u#
^.....................................................................^
```

```
v.....................................................................v
end-terminal:   DEUTSCH nach G0,
         KLAMMER nach G1,
         G0 aktivieren.

#END_TERMINAL:^[(K^[)w^O#
^.....................................................................^

#Z1:^[)u#   Zeichensatz EURO
#Z2:^[)t#   MATH
#Z3:^[)w#   Klammer
#Z4:^[)v#   IBM
#Z5:^[)C#   Mosaic
#Z6:^[)K#   Deutsch explizit
#Z7:^[)3#   International

#ATTR_US:4#
#ATTR_FETT:7#
#ATTR_BLINK:5#
#ATTR_HAHE:2#

#SWITCH_ATTR:^[[%s%e^O12z;%g^O3zm#

#ATTR_1:ATTR_BLINK#
#ATTR_2:ATTR_FETT#
#ATTR_3:ATTR_US#
#ATTR_4:ATTR_HAHE#
#ATTR_8:ATTR_FETT#

#BEEP_CHAR:^G#

#LINES:24#
#COLUMNS:80#
#HOR_SCROLL_FAKT:8#

#SCHMIER_ZEICHEN:Z3:^O5eh#


#C_ABSMARK:Z3:^O3eh#
#C_SPACE:Z0:^O20h#
#C_AUSRUFEZCH:Z0:^O21h#
#C_":Z0:^O22h#
#C_ASCHASH:Z0:^O23h#
#C_ASCDOLL:Z0:^O24h#
```

```
#C_%:Z0:^025h#
#C_AMPERSAND:Z0:^026h#
#C_ASC':Z0:^027h#
#C_(:Z0:^028h#
#C_):Z0:^029h#
#C_*:Z0:^02ah#
#C_+:Z0:^02bh#
#C_,:Z0:^02ch#
#C_-:Z0:^02dh#
#C_.:Z0:^02eh#
#C_/:Z0:^02fh#
#C_0:Z0:^030h#
#C_1:Z0:^031h#
#C_2:Z0:^032h#
#C_3:Z0:^033h#

... (alle ASCII-Zeichen)

#C_METABLANK:Z4:^036h
#C_RUFZCHNINVERS:Z2:^030h#
#C_c/:Z2:^031h#
#C_L-:Z1:^06eh#
#C_DOLLAR:Z0:^024h#
#C_Y=:Z2:^032h#
#C_HASH:Z0:^023h#
#C_PARAGR:Z1:^06ch#
#C_ox:Z7:^024h#
#C_< :Z2:^033h#
#C_oo:Z1:^073h#
#C_+-:Z2:^035h#
#C_22:Z4:^062h#
#C_33:Z4:^063h#
#C_xx:Z2:^036h#
#C_my:Z1:^072h#
#C_PA:Z4:^034h#

... (alle T.61-Zeichen)
```

```
#SONDER_00:Z0:^03Eh#  /* Linker Rand */
#SONDER_01:Z0:^03Ch#  /* Rechter Rand */
#SONDER_02:Z4:^036h#  /* Tabulatorzeichn */
#SONDER_03:Z2:^04Fh#  /* Tabellentabulator */
#SONDER_04:Z0:^023h#  /* Dezimaltabulator */
#SONDER_05:Z0:^02Ch#  /* Comma */
#SONDER_06:Z0:^02Eh#  /* Period */
#SONDER_07:Z4:^02Eh#  /* Klingelzeichen */
#SONDER_08:Z4:^02Eh#  /* Klingelzeichen */

        /* Fuellzeichen in Linealen */

#SONDER_09:Z0:^02Dh#  /* Fuellzeichen fuer normales Lineal */
#SONDER_0A:Z4:^03Fh#  /* Fuellzeichen fuer Seitenumbruchlineal */

        /* Sonstige SONDERZEICHEN */

#SONDER_10:Z3:^03Dh#  /* Sanduhr (Geduldssymbol) */
#SONDER_11:Z3:^051h#  /* UHR - Zeichen */
#SONDER_12:Z3:^06Dh#  /* Verstaerker Symbol */
#SONDER_13:Z3:^041h#  /* Textbereichs Begrenzungszeichen (frame - ) */
#SONDER_14:Z4:^039h#  /* Attributsign in systemline for TIEF */
#SONDER_15:Z4:^038h#  /* Attributsign in systemline for HOCH */
#SONDER_16:Z4:^032h#  /* Attributsign in systemline for HOCHTIEF */
#SONDER_18:Z3:^059h#  /* frameline ^ Marke */
#SONDER_19:Z4:^065h#  /* frameline 5 */
#SONDER_1A:Z4:^061h#  /* frameline 1 */
```

11.1.3.2 Die Tastaturbelegungs-Tabelle (keycap)

Jede Taste auf der Tastatur erzeugt bei ihrer Betätigung einen Code (siehe SINIX-Kommando `xcho`). Dieser wird über eine Tastaturbelegungs-Tabelle (keycap) der betreffenden Hit-Funktion zugewiesen.

Die Tastaturbelegungs-Tabelle befindet sich im Dateiverzeichnis `/usr/lib/hit/keycap`, die Standardeinstellung ist `keycap`. Benutzt wird die Keycap von den Modulen `hit` und `fit`. Falls nicht der Standardname *keycap* verwendet werden soll oder sich die Keycap in einem anderen Dateiverzeichnis befindet, so existieren wie bei den Meldungstext-Dateien und der Termcap zwei Möglichkeiten der Veränderung:

- Aufruf des Moduls mit dem Schalter `-K` und dem Namen der gewünschten Keycap. Falls die Keycap nicht mit absolutem oder relativem Pfadnamen angegeben wurde, wird standardmäßig auf die Keycap im Dateiverzeichnis `/usr/lib/hit/keycap` zugegriffen.
- Setzen der Shell-Variable `HITKEY`. Diese wird ausgewertet, wenn der Schalter `-K` nicht verwandt wurde.

Außerdem existiert noch eine Keycap `keycap.g`. Sie enthält die Codes, um beim Editieren mit den numerischen Tasten Semigrafik am Bildschirm zu erzeugen.
Angesprochen wird sie durch Drücken der Tasten SHIFT + F20
Linien-Graphik.

Aufbau der Keycap (original)

(Dieser Abschnitt dient nur der Erläuterung; die genannten Dateien werden nicht mit dem System ausgeliefert)

Ebenso wie die Meldungstexte liegt auch die Keycap im editierbaren HIT-Format vor und muß, um in eine ablauffähige Form gebracht zu werden, nach der Bearbeitung mit einem "keycap-Compiler" übersetzt werden.

Der Aufbau der Keycap ist ähnlich dem der Termcap bzw. Printcap: Auf den HIT-internen String folgt die entsprechende Taste. Hier sind, wie bei der Printcap, verschiedene Schreibweisen möglich. Der Eintrag kann hexadezimal, oktal, dezimal oder in ASCII-Schreibweise erfolgen. Für hexadezimal, oktal und dezimal muß das entsprechende Kürzel h, o oder d angehängt werden.

Beispiel

```
#T_A:^041h#
#T_A:^101o#
#T_A:^065d#
#T_A:A#
```

Zur Erzeugung von Zeichen, die nicht auf der Tastatur vorhanden sind, gibt es zwei Möglichkeiten:

- Kombination mit Kombitaste. Hierzu muß mindestens eine Taste als sog. "Kombi"-Taste eingetragen sein. Im Normalfall ist dies die Taste [CE] des numerischen Tastenblocks. Die Namen der Tasten, mit denen das neue Zeichen erzeugt werden soll, werden über das Zeichen & verknüpft.

  Beispiel

  ```
  #T_Y=:T_Y&T_=#
  ```

- Kombination mit "non-spacing"-Zeichen. Soll ein Zeichen so definiert werden, daß sich die Schreibmarke nicht weiterbewegt, so muß es mit der fiktiven Taste T_NS verknüpft werden.

Beispiel

```
#T_ASCDACH:T_NS&^05eh#
```

Beispiel für die Hit-Tastaturbelegungs-Tabelle (Auszug)

```
#KEY:^07fh#       Tastaturcode  7f oder ff

#T_USR2F:^[#
#T_INVALID:^[+:^[%:^[&:^[,:^[2:^[4:^[5:^[6:^[7:^[8:^[1:^[V:^[W:^[X:^[Y:^[Z:#

#T_SPACE: #
#T_AUSRUFEZCH:!!#
#T_":"#
#T_ASCHASH:^023h#
#T_ASCDOLL:^024h#
#T_%:%#
#T_AMPERSAND:&#
#T_ASC':T_NS&'#
#T_(:(#
#T_):)#
#T_*:*#
#T_+:+#
#T_,:,#
#T_-:-#
#T_.:.#
#T_/:/#
#T_0:0#
#T_1:1#
#T_2:2#
#T_3:3#
#T_4:4#
#T_5:5#
#T_6:6#
#T_7:7#
#T_8:8#
#T_9:9#
#T_DP:!:#       Doppelpunkt muß entwertet werden
```

```
#T_;:;#
#T_<:<#
#T_=:=#
#T_>:>#
#T_?:?#
#T_KLAMMERAFFE:T_PARAGR&T#_PARAGR#
#T_A:A#
#T_B:B#

... (alle ASCII-Zeichen)

#T_y:y#
#T_z:z#
#T_GESCHKLAMAUF:T_a_UMLAUT&T_a_UMLAUT#
#T_PIPE:T_o_UMLAUT&T_o_UMLAUT#
#T_GESCHKLAMZU:T_u_UMLAUT&T_u_UMLAUT#
#T_ASCTILDE:T_ss&T_ss#

*******************************************************************************

#T_METABLANK:^05fh#
#T_RUFZCHNINVERS:T_AUSRUFEZCH&T_AUSRUFEZCH#
#T_c/:T_c&T_/#
#T_L-:T -&T_L#
#T_DOLLAR:T_ASCDOLL&T_ASCDOLL#
#T_Y=:T_Y&T_=#

... (alle T.61-Zeichen)

#F_KOMBI:^01bh^05eh:^01bh^05dh#    CE als Kombinationstaste
#F_END:^D#            Ende der Dokumentbearbeitung

/*************************  Funktionstasten 00-45  *************************/

Fkt_10:^[;#          Attribut loeschen
Fkt_11:^[O#          neues Zeilenlineal einfuegen
Fkt_12:^[P#          Zeilenlineal bearbeiten
Fkt_13:^[O#          Externfunktion
Fkt_14:^[g#          Ausdrucken
Fkt_15:^['#          Suchen wiederholen
Fkt_16:^[<#          Bereich einfuegen
Fkt_17:^[G#          Blocksatz
Fkt_18:^[-#          sichern
Fkt_19:^[ #          Form attrsw
```

```
Fkt_1A:^[[1#          Fuege Char ein
Fkt_1B:^[o#           Fuege Wort ein
Fkt_1C:^[[L#          Fuege Zeile ein
Fkt_1D:^[[P#          Loesche Char
Fkt_1E:^[p#           Loesche wort
Fkt_1F:^[[M#          Loesche Zeile
Fkt_20:^[[H#          Verstaerker
Fkt_21:^[[B#          Cursor down  (Pfeil nach unten)
Fkt_22:^[[D#          Cursor left  (Pfeil nach links)
Fkt_23:^[[C#          Cursor right (Pfeil nach rechts)
Fkt_24:^[[A#          Cursor up    (Pfeil nach oben)
Fkt_25:^[[T#          Cursor Absatz nach unten
```

11.1.3.3 Die Druckertabelle (printcap)

Die Ansteuerung der angeschlossenen Drucker erfolgt - wie beim Bildschirm und der Tastatur - ebenfalls über Tabellen, die in binärer Form vorliegen. Sie befinden sich im Dateiverzeichnis `/usr/lib/hit/printcap`.

Über den Aufbau der `printcap` und die Möglichkeiten ihrer Veränderung informiert das Kap. 7.2.

11.1.4 Dateiverzeichnisse, die zum Ablauf benötigt werden

Zusätzlich zu den Meldungstexten, der Termcap und Keycap müssen für die Ablauffähigkeit des HIT-Systems noch eine Reihe von Dateiverzeichnissen vorhanden sein. Es handelt sich um:

- `/usr/lib/hit/tmp`
  Hier legen die Module während der Dateibearbeitung ihre temporären Dateien ab.

- `/usr/lib/hit/recovery`
  In dieses Dateiverzeichnis werden die temporären Dateien transferiert, die sich beim Hochfahren des Systems noch im Dateiverzeichnis `/usr/lib/hit/tmp` befinden (siehe Kapitel 11.4).

- `/usr/lib/hit/spool`
  In diesem Dateiverzeichnis stehen die mit `clouhg` erstellten Schreibaufträge bis zu ihrem vollständigen Ausdruck.

- `/usr/lib/hit/public`
  Auf dieses Dateiverzeichnis greift die Funktion `in/aus Verteiler` zu.

- `/usr/lib/hit/printer`
  Dieses Dateiverzeichnis enthält die Druckeraufrufe in Form von Shell-Prozeduren.

- `/usr/lib/hit/scripts-D`
  In Unter-Dateiverzeichnissen stehen Prozeduren für die Module `fit` (FIT), `fitlpr` (FITLPR) und `fedor` (FEDOR) sowie für installierte Anwendungen (APP). Darüber hinaus existieren die Datei `application` (ausgewertet beim Erstellen der Steuerzeile `Anwendung`) sowie `rc` (wird beim Systemstart aufgerufen).

  Die HIT-MENÜ-Funktionen `Inhaltsverzeichnis Archiv/Ordner` sowie `Texthandbuch` werden über CLOU-Bausteine (im Dateiverzeichnis `FIT`, Suffix `.tbs`) gesteuert. Eine Formatanpassung kann durch Modifikation dieser Bausteine erfolgen.

- `/usr/lib/hit/umgebung-D`
  Dieses Dateiverzeichnis enthält genau zwei Dateien, nämlich `hitumg` (die Standard-Umgebungstabelle) und `formtab` (die Standard-Formatiertabelle).

- `/usr/lib/hit/help-D`
  In entsprechenden Unter-Dateiverzeichnissen befinden sich die Hilfe-Texte für die Module `fit`, `hfind`, `hform`, `hindex` und `hit`. Bei `messages` handelt es sich um komprimierte HIT-Texte; `memout` bei den Hilfe-Texten für hit ist ein normaler HIT-Text, der bei Speicherknappheit zur Anzeige gebracht wird.

11.2 Die Dateistruktur des Benutzers

Im Gegensatz zum SINIX-Standard-Menüsystem, das das HOME-Dateiverzeichnis eines Benutzers als (einziges) Archiv kennt und darin das Anlegen von Ordnern erlaubt, kann ein HIT-Benutzer in seinem HOME-Dateiverzeichnis beliebige Archive und darin beliebige Ordner anlegen. HIT macht sich damit die baumartige Datei-Struktur des Betriebssystems SINIX zunutze.

Beispiel

SINIX-Standard-Menüsystem

`/usr/mgast`	HOME-Dateiverzeichnis, Archiv
`/usr/mgast/Briefe`	Ordner
`/usr/mgast/Briefe/Mahnung`	Dokument

HIT

`/usr/mgast`	HOME-Dateiverzeichnis
`/usr/mgast/Dokumente`	Archiv
`/usr/mgast/Dokumente/Kurs`	Ordner
`/usr/mgast/Dokumente/Kurs/Anleitung`	Dokument

Aus obigem Beispiel ist auch gut ersichtlich, daß HIT-Dokumente, um im SINIX-Standard-Menüsystem angesprochen werden zu können, erst in dieses (also eine Hierarchiestufe höher) transferiert werden müssen.

Beim Aufruf von HIT-MENÜ wird geprüft, ob im HOME-Dateiverzeichnis ein Dateiverzeichnis `.Papierkorb` existiert und (falls es nicht vorhanden ist) angelegt. `.Papierkorb` enthält ein Dateiverzeichnis `.Ordner`, dieses wieder ein Dateiverzeichnis `.Dokumente`. Die Funktion `in/aus-Papierkorb` legt gelöschte Archive/Ordner/Dokumente unter dieser Struktur ab bzw. greift beim Zurückholen darauf zu.

11.3 Vorgänge beim Aufruf des Textsystems HIT

`/usr/bin/HIT/HIT_env` prüft, ob der Benutzer bereits mit HIT gearbeitet hat. Trifft dies nicht zu, so werden Standard-Archive eingerichtet und Beispieldokumente aus der Benutzerkennung `admin` kopiert.

11.3.1 Start von HIT-MENÜ

HIT-MENÜ (`fit`) legt die vom Benutzer gemachten Einstellungen für einen späteren Aufruf in einer sog. Statusdatei mit dem Namen *.F$USER* im HOME-Dateiverzeichnis ab. Diese wird bei jedem Aufruf gelesen.

11.3.2 Start der Textbearbeitung (hit)

Nach dem Drücken der START-Taste in HIT-MENÜ legt `fit` eine "Kommunikationsdatei" mit dem Namen `FIT`*PID* im Dateiverzeichnis `/usr/lib/hit/tmp` an. In dieser Datei stehen neben dem Dateinamen alle Aufrufparameter, die `hit` übergeben werden. Der Aufruf des Editors selbst erfolgt mit dem Schalter -h und dem Namen der Kommunikationsdatei, aus der der Editor sodann alle nicht direkt übergebenen Aufrufparameter liest.

Nach dem Aufruf des Editors sind die Programme `hit` und `fit` geladen, die Bildschirmausgabe des jeweils nicht aktiven Programms wird eingestellt. Hiermit ist ein schneller Wechsel zwischen Editor und HIT-Menü möglich, die Zwischenspeicher bleiben während der gesamten HIT-Sitzung erhalten.

Der Editor legt im Dateiverzeichnis `/usr/lib/hit/tmp` für das zu bearbeitende Dokument eine temporäre Datei an. Diese besitzt ein eigenes Format und ermöglicht den schnellen Zugriff auf alle Textpassagen.

In der Regel erfolgt eine Aktualisierung der temporären Datei ca. einmal in der Minute. Bei einem Anlagenausfall bleibt die temporäre Datei erhalten, sodaß sich der Datenverlust auf den Zeitraum bis zum letzten Zurückschreiben beschränkt (zum automatischen Rekonstruieren siehe Kapitel 11.4).

Im aktuellen Dateiverzeichnis legt der Editor eine Locking-Datei für das zu bearbeitende Dokument an. Diese hat den Namen *.#Dateiname* und enthält den Namen der zu diesem Dokument gehörenden temporären Datei. Damit wird verhindert, daß mehrere Benutzer gleichzeitig auf ein und dasselbe Dokument lesend und schreibend zugreifen. Außerdem wird der Inhalt der Locking-Datei beim automatischen Rekonstruieren ausgewertet.

Wenn der Editor mit einem Dokumentnamen aufgerufen wird, zu dem bereits eine Locking-Datei existiert, wird in der Systemzeile anstelle der Attribute `NUR LESEN` ausgegeben. D.h., daß in diesem Dokument keine Veränderungen vorgenommen werden können.

11.3.3 Beenden der Textbearbeitung

Beim Drücken von ENDE und Sichern des Dokuments laufen folgende Aktionen ab:

- Aktualisieren der temporären Datei
- Abspeichern des Dokuments im externen HIT-Format im aktuellen Dateiverzeichnis
- Löschen der temporären Datei
- Löschen der Locking-Datei
- Zurückschreiben des Dokumentnamens in die Kommunikationsdatei zur Auswertung durch `fit`.
- Einstellen der Bildschirmausgabe (der Prozeß bleibt erhalten)
- Aktivieren des `fit`

Falls eine Datei beim Verlassen des Editors nicht gesichert wurde, in der Umgebungstabelle aber `Sicherungsdatei` gesetzt ist, so wird die temporäre Datei unter dem bei `Sicherungsdatei` angegebenen Namen ins aktuelle Dateiverzeichnis transferiert. Damit bleibt die Datei erhalten und kann, falls gewünscht, wie ein normales Dokument weiter bearbeitet werden. HIT erkennt dabei selbsttätig, daß es sich um eine temporäre Datei handelt.

Alle laufenden Prozesse werden erst nach Betätigen von ENDE in HIT-MENÜ beendet.

11.4 Vorgänge bei bzw. nach einem Systemabsturz

Bei einem Ausfall der Anlage wird die Bearbeitung der Dokumente nicht korrekt beendet. Damit bleiben die momentanen temporären Dateien erhalten und stehen für die automatische Rekonstruktion zur Verfügung.

Beim Hochfahren der Anlage läuft u.a. die Prozedur `/etc/rc.sys5` ab. Diese erhält bei der HIT-Installation den Eintrag `/usr/lib/hit/scripts-D/rc&`. Es handelt sich dabei um folgende Shellprozedur:

```
# Prozedurname: rc
# Aufruf durch /etc/rc
# Alle fuer den korrekten Ablauf des HIT erforderlichen directories werden,
# falls nicht vorhanden, eingerichtet.
# Dateien im Verteiler (public) und recovery-Dateien, die aelter als
# 7 Tage sind,. werden weggeworfen.
# Durch Systemabsturz unter "tmp" verbliebene Paging-Dateien werden nach
# "recovery" gestellt, um automatisches Rekonstruieren zu ermoeglichen.
# cloudaemon wird gestartet

for dir in tmp recovery spool public
do
    if [ ! -d /usr/lib/hit/"$dir" ]
    then /bin/mkdir /usr/lib/hit/"$dir"
        /bin/chmod 777 /usr/lib/hit/"$dir"
    fi
done

/bin/find /usr/lib/hit/public -type f -atime +7 -exec /bin/rm -f {} \;
/bin/find /usr/lib/hit/recovery -type f -atime +7 -exec /bin/rm -f {} \;
/bin/mv /usr/lib/hit/tmp/HP* /usr/lib/hit/recovery 2>/dev/null
/bin/rm -f /usr/lib/hit/tmp/* /usr/lib/hit/tmp/.#*
/bin/rm -f /usr/lib/hit/spool/CLOUDAEMON

/usr/bin/HIT/cloudaemon&
```

Wird nun die Textbearbeitung mit dem nicht gesicherten Dokument gestartet, so erkennt der Editor an der ebenfalls noch vorhandenen Locking-Datei, daß das Dokument entweder bereits bearbeitet wird oder ein Systemabsturz vorausging. Der Absturz wird daran erkannt, daß sich die zugehörige temporäre Datei nicht im Dateiverzeichnis `/usr/lib/hit/tmp`, sondern unter `/usr/lib/hit/recovery` befindet. Der `hit` geht von selbst in den Rekonstruiermodus über, d.h. die noch vorhandene temporäre Datei wird weiter verwendet.

Wenn nach einem Absturz die Anlage nicht neu hochgefahren wird, können die zum Zeitpunkt des Absturzes gerade bearbeiteten Dokumente nur noch gelesen werden (Locking-Datei vorhanden, Paging-Datei nicht unter `/usr/lib/hit/recovery`). In diesem Fall kann über den Aufruf der Prozedur `hrec` die Paging-Datei nach `/usr/lib/hit/recovery` gestellt werden; das Rekonstruieren ist dann wieder möglich.

Normalerweise speichert HIT die temporären Dateien unter `/usr/lib/hit/tmp` ab. Sie haben die Möglichkeit, diese Einstellung zu verändern. Das ist v.a. dann sinnvoll, wenn die einzelnen Benutzer auf verschiedenen Plattenbereichen arbeiten.

Belegen Sie für jeden Benutzer die Shell-Variable `HITTMP` mit dem Namen des Dateiverzeichnisses, in dem die temporären Dateien abgelegt werden sollen.

```
HITTMP=Dateiverzeichnis
export HITTMP
```

Das Dateiverzeichnis muß mit seinem absoluten Pfadnamen angegeben werden.
Durch diese Maßnahme kann die Performance von HIT erheblich verbessert werden.
Bitte beachten Sie, daß die Prozedur `/usr/bin/HIT/rc` entsprechend erweitert bzw. verändert werden muß, damit bei einem Systemabsturz die temporären Dateien für die automatische Rekonstruktion zur Verfügung stehen.

Beispiel

```
HITTMP = /usr1/hit/tmp
export HITTMP
```

Dann erweitern Sie `/usr/bin/HIT/rc` folgendermaßen:

```
/bin/mv /usr1/hit/tmp/HP* /usr/lib/hit/recovery 2>/dev/null
/bin/rm-f /usr1/hit/tmp/* /usr1/hit/tmp/.#*
```

11.5 Installation des HIT-Systems

Ein korrekter Ablauf des Textsystems HIT ist mit der automatischen Installation gewährleistet. Bei Verzicht auf die Installations-Prozedur müssen alle Kommandos von Hand nachvollzogen werden.

11.5.1 Struktur der Liefereinheiten

Das Textsystem HIT besteht aus drei getrennten Liefereinheiten, die für sich alleine nicht ablauffähig sind. Die einzelnen Einheiten sind der Grundausbau, der Sprachzusatz und die entsprechende Key-Diskette. Der Grundausbau enthält die Kernprogramme unter `/usr/bin/HIT`. Der Sprachzusatz enthält alle Teile, die irgendwie sprachabhängig sind (z.B. die Meldungstexte, das Silbentrennprogramm, die Menüs des Systemverwalters, die Installations-Prozedur).

11.5.2 Installations-Prozedur

Alle Installationsabfragen und Meldungen erscheinen in englischer Sprache. Die Installation von HIT ist auf Anlagen, die über einen X/OPEN-Modus verfügen, nur in diesem möglich. Bei der Abfrage, in welchem Universum HIT installiert werden soll, ist somit `att` einzugeben (die Abfrage nach dem Universum unterbleibt, wenn von `/etc/superinstall` die Datei `install.conf` ausgewertet wird).

Die Installation wird abgebrochen, wenn gerade mit HIT gearbeitet wird. Existiert ein Benutzer `mgast`, wird die Beispielsitzung in dessen HOME-Dateiverzeichnis kopiert. Ansonsten wird nach dem entsprechenden Benutzer abgefragt.

Nach dem Einlesen der Meldungstexte und der Bestimmung der Betriebssystem-Parameter wird die Versionsnummer einer evtl. vorhandenen HIT-Version ermittelt. Ehe nun eine vorhandene HIT-Version gelöscht wird, werden die Druckeraufrufe, die Original-Druckertabellen sowie die Ausnahmedatei in einem Dateiverzeichnis `/usr/lib/hit.sav` gesichert. Der Systemverwalter kann damit nach Beendigung der Installation seine spezielle Anlagen-Konfiguration wieder herstellen.

Danach erfolgt das Einlesen der gesamten Version. Dazu werden nacheinander alle Datenträger angefordert und dabei die Reihenfolge überprüft.

Um die HIT-Version ablauffähig zu machen, werden anschließend folgende Anpassungen vorgenommen:

- Aufruf einer Shell-Prozedur (`/usr/bin/HIT/HIT_env`), die die HIT-spezifischen Umgebungsvariablen setzt. Der Aufruf erfolgt aus `/etc/.profile` und `/usr/att/etc/profile` heraus.
- Ergänzung der Datei `/etc/rc.sys5` um den Eintrag `/usr/bin/HIT/rc&` (für automatisches Rekonstruieren).
- Einrichten der Verweise für die Meldungstext-Dateien, die Tastenbelegungs- und Bildschirmsteuerungs-Tabellen.
- Einrichten der Dateiverzeichnisse `tmp`, `recovery`, `spool` und `public` unter `/usr/lib/hit`.
- Anpassung der Druckeraufrufe an die Anlagen-Konfiguration (Druckertypen, Druckergruppen).
- Einrichten einer Datei `.profile` für die Benutzerkennung, die bei der Installation die Beispielsitzung erhalten hat.
- Löschen alter Einträge in `/usr/admin/.products` und Eintrag der neuen Versionsbezeichnung.

A Anhang

A.1 HIT-Zeichen

Die folgende Übersicht bezieht sich auf die Tastatur des PC-MX2.

- Solchermaßen gekennzeichnete Zeichen können vom Bildschirm nicht abgebildet werden; Ersatzdarstellung bei HIT: 

Zeichen	Name HIT-int.	sedez. Code
	SPACE	020
!	AUSRUFEZCH	021
"	"	022
#	ASCHASH	023
$	ASCDOLL	024
%	%	025
&	AMPERSAND	026
'	ASC'	027
(	(	028
)	)	029
*	*	02a
+	+	02b
,	,	02c
-	-	02d
.	.	02e
/	/	02f

Zeichen	Name HIT-int.	sedez. Code
0	0	030
1	1	031
2	2	032
3	3	033
4	4	034
5	5	035
6	6	036
7	7	037
8	8	038
9	9	039
:	DP	03a
;	;	03b
<	<	03c
=	=	03d
>	>	03e
?	?	03f

Zeichen	Name HIT-int.	sedez. Code
@	KLAMMERAFFE	040
A	A	041
B	B	042
C	C	043
D	D	044
E	E	045
F	F	046
G	G	047
H	H	048
I	I	049
J	J	04a
K	K	04b
L	L	04c
M	M	04d
N	N	04e
O	O	04e
P	P	050
Q	Q	051
R	R	052
S	S	053
T	T	054
U	U	055
V	V	056
W	W	057

Zeichen	Name HIT-int.	sedez. Code
X	X	058
Y	Y	059
Z	Z	05a
[	ECKKLAMAUF	05b
\	BACKSLASH	05c
]	ECKKLAMZU	05d
^	ASCDACH	05e
_	ASCUNDER	05f
`	ASC`	060
a	a	061
b	b	062
c	c	063
d	d	064
e	e	065
f	f	066
g	g	067
h	h	068
i	i	069
j	j	06a
k	k	06b
l	l	06c
m	m	06d
n	n	06e
o	o	06f

Zeichen	Name HIT-int.	sedez. Code
p	p	070
q	q	071
r	r	072
s	s	073
t	t	074
u	u	075
v	v	076
w	w	077
x	x	078
y	y	079
z	z	07a
{	GESCHKLAMAUF	07b
|	PIPE	07c
}	GESCHKLAMZU	07d
~	ASCTILDE	07e
▬	METABLANK	0a0
¡	RUFZCHNINVERS	0a1
¢	c/	0a2
£	L-	0a3
$	DOLLAR	0a4
¥	Y=	0a5
#	HASH	0a6
§	PARAGR	0a7
¤	ox	0a8

Zeichen	Name HIT-int.	sedez. Code
‘	xA9	0a9
“	xAA	0aa
«	<<	0ab
←	xAC	0ac
↑	xAD	0ad
→	xAE	0ae
↓	xAF	0af
°	oo	0b0
±	+-	0b1
²	22	0b2
³	33	0b3
×	xx	0b4
µ	my	0b5
¶	PA	0b6
·	..	0b7
÷	DP-	0b8
’	xB9	0b9
”	xBA	0ba
»	>>	0bb
¼	14	0bc
½	12	0bd
¾	34	0be
¿	??	0bf
`	`	0c1

Zeichen	Name HIT-int.	sedez. Code
´	´	0c2
^	DACH	0c3
~	TILDE	0c4
¯	MACRON	0c5
˘	BREVE	0c6
˙	I_PUNKT	0c7
¨	UMLAUTPUNKTE	0c8
°	RING	0ca
¸	CEDILLE	0cb
˝	''	0cd
˛	OKOGNEK	0ce
ˇ	CARON	0cf
–	USR20	0d0
¹	USR21	0d1
®	USR22	0d2
©	USR23	0d3
™	USR24	0d4
♪	USR25	0d5
⅛	USR2C	0dc
⅜	USR2D	0dd
⅝	USR2E	0de
⅞	USR2F	0df
I	USR00	080
L	USR01	081

Zeichen	Name HIT-int.	sedez. Code
⊥	USR02	082
⌋	USR03	083
⊢	USR04	084
+	USR05	085
⊣	USR06	086
⌈	USR07	087
T	USR08	088
¬	USR09	089
_	USR0A	08a
Ω	O_	0e0
Æ	AE	0e1
Đ	D–	0e2
ª	a_	0e3
Ħ	H–	0e4
IJ	IJ	0e6
Ŀ	L.	0e7
Ł	L/	0e8
Ø	O/	0e9
Œ	OE	0ea
º	o_	0eb
Þ	IP	0ec
Ŧ	T–	0ed
Ŋ	N,	0ee
ħ	n`	0ef

Zeichen	Name HIT-int.	sedez. Code
ĸ	KK	0f0
æ	ae	0f1
đ	d–	0f2
ð	d/	0f3
ħ	h–	0f4
ı	i–	0f5
ij	ij	0f6
ŀ	l.	0f7
ł	l/	0f8
ø	o/	0f9
œ	oe	0fa
ß	ss	0fb
þ	ip	0fc
ŧ	t–	0fd
ŋ	n,	0fe
À	A`	100
Á	A´	101
Â	A_DACH	102
Ã	A_TILDE	103
Ā	A_MACRON	104
Ă	A_BREVE	105
Ä	A_UMLAUT	106
Å	A_RING	107
Ą	A_OKOGNEK	108

Zeichen	Name HIT-int.	sedez. Code
Ć	C´	109
Ĉ	C_DACH	10a
Ċ	C.	10b
Ç	C_CEDILLE	10c
Č	C_CARON	10d
Ď	D_CARON	10e
È	E`	10f
É	E´	110
Ê	E_DACH	111
Ē	E_MACRON	112
Ė	E.	113
Ë	E_UMLAUT	114
Ę	E_OKOGNEK	115
Ě	E_CARON	116
Ĝ	G_DACH	118
Ğ	G_BREVE	119
Ġ	G.	11a
Ģ	G_CEDILLE	11b
Ĥ	H_DACH	11c
Ì	I`	11d
Í	I´	11e
Î	I_DACH	11f
Ĩ	I_TILDE	120
Ī	I_MACRON	121

Zeichen	Name HIT-int.	sedez. Code
İ	I.	122
Ï	I_UMLAUT	123
Į	I_OKOGNEK	124
Ĵ	J_DACH	125
Ķ	K_CEDILLE	126
Ĺ	L´	127
Ļ	L_CEDILLE	128
Ľ	L_CARON	129
Ń	N´	12a
Ñ	N_TILDE	12b
Ņ	N_CEDILLE	12c
Ň	N_CARON	12d
Ò	O`	12e
Ó	O´	12f
Ô	O_DACH	130
Õ	O_TILDE	131
Ō	O_MACRON	132
Ö	O_UMLAUT	133
Ő	O_"	134
Ŕ	R´	135
Ŗ	R_CEDILLE	136
Ř	R_CARON	137
Ś	S´	138
Ŝ	S_DACH	139

Zeichen	Name HIT-int.	sedez. Code
Ş	S_CEDILLE	13a
Š	S_CARON	13b
Ţ	T_CEDILLE	13c
Ť	T_CARON	13d
Ù	U`	13e
Ú	U´	13f
Û	U_DACH	140
Ũ	U_TILDE	141
Ū	U_MACRON	142
Ŭ	U_BREVE	143
Ü	U_UMLAUT	144
Ů	U_RING	145
Ű	U_"	146
Ų	U_OKOGNEK	147
Ŵ	W_DACH	148
Ý	Y´	149
Ŷ	Y_DACH	14a
Ÿ	Y_UMLAUT	14b
Ź	Z´	14c
Ż	Z.	14d
Ž	Z_CARON	14e
à	a`	150
á	a´	151
â	a_DACH	152

Zeichen	Name HIT-int.	sedez. Code
ã	a_TILDE	153
ā ▮	a_MACRON	154
ă	a_BREVE	155
ä	a_UMLAUT	156
å	a_RING	157
ą	a_OKOGNEK	158
ć	c´	159
ĉ ▮	c_DACH	15a
ċ ▮	c.	15b
ç	c_CEDILLE	15c
č	c_CARON	15d
ď	d_CARON	15e
è	e`	15f
é	e´	160
ê	e_DACH	161
ē ▮	e_MACRON	162
ė ▮	e.	163
ë	e_UMLAUT	164
ę	e_OKOGNEK	165
ě	e_CARON	166
ģ ▮	g´	167
ĝ ▮	g_DACH	168
ğ	g_BREVE	169
ġ ▮	g.	16a

Zeichen	Name HIT-int.	sedez. Code
ĥ ▮	h_DACH	16c
ì	i`	16d
í	i´	16e
î	i_DACH	16f
ĩ ▮	i_TILDE	170
ī ▮	i_MACRON	171
ï	i_UMLAUT	173
į ▮	i_OKOGNEK	174
ĵ ▮	j_DACH	175
ķ ▮	k_CEDILLE	176
ĺ	l´	177
ļ ▮	l_CEDILLE	178
ľ	l_CARON	179
ń	n´	17a
ñ	n_TILDE	17b
ņ ▮	n_CEDILLE	17c
ň	n_CARON	17d
ò	o`	17e
ó	o´	17f
ô	o_DACH	180
õ	o_TILDE	181
ō ▮	o_MACRON	182
ö	o_UMLAUT	183
ő	o_"	184

Zeichen	Name HIT-int.	sedez. Code
ŕ	r´	185
ŗ ▮	r_CEDILLE	186
ř	r_CARON	187
ś	s´	188
ŝ ▮	s_DACH	189
ş	s_CEDILLE	18a
š	s_CARON	18b
ţ	t_CEDILLE	18c
ť	t_CARON	18d
ù	u`	18e
ú	u´	18f
û	u_DACH	190
ũ ▮	u_TILDE	191

Zeichen	Name HIT-int.	sedez. Code
ū ▮	u_MACRON	192
ŭ ▮	u_BREVE	193
ü	u_UMLAUT	194
ů	u_RING	195
ű	u_"	196
ų ▮	u_OKOGNEK	197
ŵ ▮	w_DACH	198
ý	y´	199
ŷ ▮	y_DACH	19a
ÿ	y_UMLAUT	19b
ź	z´	19c
ż	z.	19d
ž	z_CARON	19e

Zeichen	Name HIT-int.	sedez. Code
¬	USR26	0d6
¦	USR27	0d7
•	USR10	090
◆	USR11	091
♂	USR12	092
♣	USR13	093
♀	USR14	094
♫	USR15	095
☺	USR16	096

Zeichen	Name HIT-int.	sedez. Code
☼	USR17	097
♥	USR18	098
↔	USR19	099
►	USR1A	09a
0B	USR1B	09b
0C	USR1C	09c
0D	USR1D	09d
0E	USR1E	09e
0F	USR1F	09f

Literatur

/1/ C.J. Date
An Introduction to Database Systems
Addison Wesley Publishing Company 1981

/2/ Betriebssystem SINIX
Buch1

/3/ INFORMIX Datenbanksystem
für **Siemens PC**

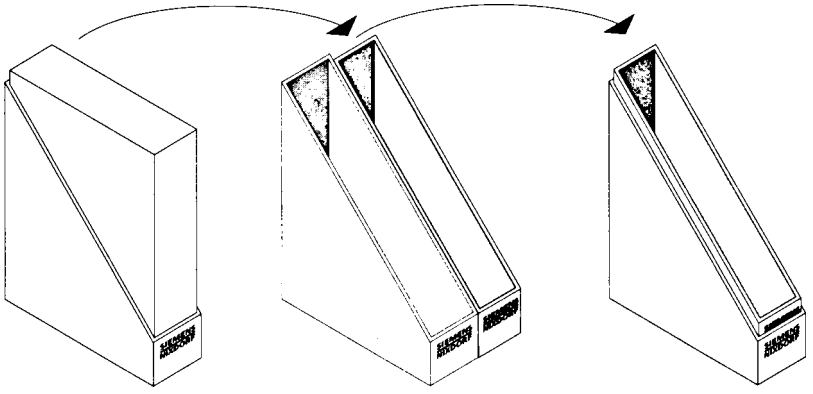
/4/ CES
C-Eentwicklungssystem

/5/ Betriebssystem SINIX
Buch2 Menüs

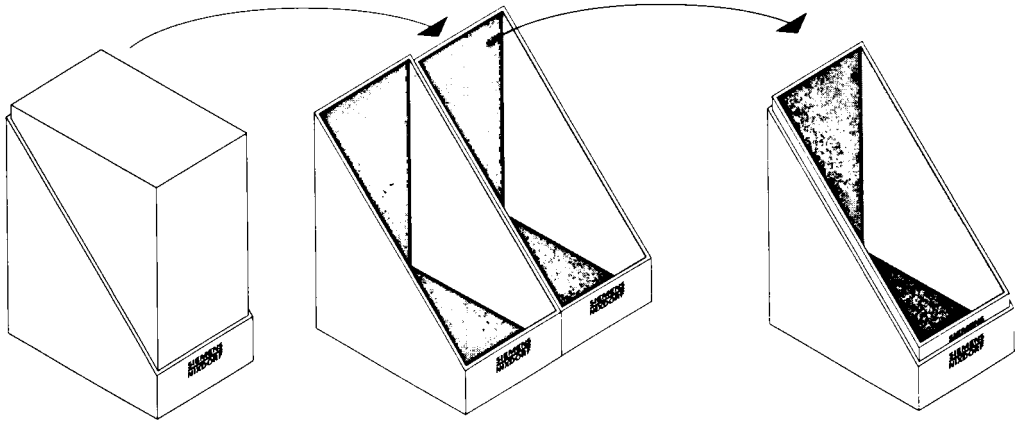
Stichwörter

Sammelboxen

Für Handbücher des vorliegenden Formates bieten wir zweiteilige Sammelboxen in zweierlei Größen an. Der Bestellvorgang entspricht dem für Handbücher.



Breite: ca. 5 cm
Bestellnummer: U3775-J-Z18-1



Breite: ca. 10 cm
Bestellnummer: U3776-J-Z18-1

924775

Herausgegeben von / Published by
Siemens Nixdorf Informationssysteme AG
Postfach 21 60, W-4790 Paderborn
Postfach 83 09 51, W-8000 München 83

Bestell-Nr. / Order No. ***U3550-J-Z97-3***
Printed in the Federal Republic of Germany
6400 AG 5923. (8000)