allegro news Nr.56

Ausgabe 99/4, 19.1.2000
Diese Ausgabe hat 16 Seiten

Universitätsbibliothek der Technischen Universität Braunschweig, Pockelsstr. 13, D-38106 Braunschweig. Tel. (0531) 391-5011, FAX -5836

www.allegro-c.de/allegro/

Ihr wartet jetzt auf ein Gedicht?
Das gibt's von nun an aber nicht.
Und warum?
Es wär' dumm,
denn wenn wir ständig neue machen
sind's nachher nur noch Dutzendsachen.

Schlussstriche

Lange wurde das herannahende Jahrtausend mit Vorschusslorbeeren überschüttet (klang nicht früher mal "im Jahr 2000" wie der Inbegriff von besserer Zukunft schlechthin?), zuletzt aber wurde ihm mit skurrilen Beklemmungen entgegengezittert, düster dräuend über allem das Damoklesschwert mit dem sinistren Monogramm Y2K. Natürlich fing es so banal an wie jedes andere, nur mit noch mehr Pomp und Rummel. Immerhin erwies sich die Panikmache als eine sich selbst vereitelnde Prophezeiung: die Desaster blieben aus, weil immense Summen und Arbeitskraft in ihre Verhütung geflossen waren. Wenn jemals am falschen Platz gespart wurde, dann am Speicherplatz, bei den bewussten zwei Ziffern. Das war teuer ...
Die oft aufflackernde Streitfrage, ob denn nun das neue Millennium wirklich schon da sei oder ob es erst beim nächsten Jahreswechsel anhebe, ist schon deshalb gänzlich müßig, weil mit *jedem* neuen Jahr eines beginnt - ein Jahrtausend ist ja schlicht jede Zeitspanne von 1000 Jahren. Der Beginn unserer Zeitrechnung scheint zudem unaufklärbar umnebelt, der Anfangspunkt des Jahres ist willkürlich, für den Rest des Universums ist es eine unbedeutende Größe, und im übrigen ist die Zahl 2000 nur im Dezimalsystem so schön rund. Dieses aber ist bloß eine Konvention und hat mit dem Phänomen "Zeit" (was immer das wirklich ist) nicht das Geringste zu tun. Keineswegs trennt uns nun plötzlich ein dicker Schlussstrich von einem vergangenen, bewältigten Geschichtsbuchkapitel. Man ist denn auch wohl allenthalben zur grauen Tagesordnung übergegangen.
Die Tagesordnung der ***allegro***-Entwicklung durchquert allerdings doch ein Schlussstrich: er trennt die alten DOS- und UNIX-Programme von den neuen objektorientierten und Client/Server-Verfahren. Die ersteren sind im Alltagseinsatz zwar noch zahlenmäßig die überwiegenden, das akute Interesse gilt aber den letzteren. Erstmals vorgestellt wurde das neue, objektorientierte Programmierkonzept zwar schon in der Nummer 39 (Sept. 1995). Verkörpert im ***avanti***-Server hat es in der Folge eine große Vielfalt von WWW-Anwendungen ermöglicht (`http://www.allegro-c.de/allegro/ac-dbs.htm`). Erst im abgelaufenen Jahr 1999 hat aber die Wende auch für viele Einzelplatz- und LAN-Anwender stattgefunden oder steht unmittelbar bevor. Mit dieser Nummer können wir feststellen: es gibt kaum noch Gründe, mit PRESTO und APAC statt mit *a99* und *alcarta* zu arbeiten. Es sei denn, die Hardware stammt noch aus finsteren Tiefen eines vergangenen Millenniums. Das ist allerdings so schlimm wieder nicht: der Schlussstrich unter DOS ist gewissermaßen eine gestrichelte Linie, also durchlässig, denn alle Daten bleiben unverändert, beide Programmsysteme können gleichzeitig auf dieselbe Datenbank zugreifen. Es gibt daher beim Umstieg kein hartes Entweder-Oder, keinen "point of no return" und kein "rien ne va plus".
Die Tagesordnung war in den letzten Monaten dominiert von Arbeiten an ***a99*** und ***alcarta***. Besonders das in Nr. 54 erstmals vorgestellte FLEX-System (eine Art Makro-Sprache) wurde laufend erweitert und verbessert. Jetzt existieren bereits FLEX-Arrangements, mit denen ganze Erwerbungs- und Ausleihverfahren betrieben werden (s.S. 11). So wie ***avanti*** mit seiner Befehlssprache, weitgehend mit der FLEX-Sprache identisch, dem kompetenten Anwender das Werkzeug für die Entwicklung von individueller Client-Software an die Hand gibt, so können solche Anwender nun auch mit ***a99/alcarta*** neue Funktionen bis hin zu ganzen Geschäftsgängen maßgeschneidert selbst entwickeln und implementieren oder auch CD-ROM-Datenbanken mit neuen, attraktiven Eigenschaften ausstatten - eine unter DOS undenkbare FLEXibilität ist erreicht. Wegen dieser herausragenden Bedeutung wird die FLEX-Sprache in dieser Nummer erneut vollständig und auf neuestem Stand dokumentiert, wobei neue Befehle als solche gekennzeichnet sind.
Wenn man allerdings erlebt, wie die gesamte FLEX-Dokumentation und vieles andere jetzt im *a99*-Fenster abgerufen und bei Bedarf auch sofort ausgedruckt oder in eigene Texte übernommen werden kann, oder dass sich damit interaktive Lektionen und selbstlaufende Demos gestalten lassen, dann drängt sich der Gedanke auf, dass jetzt die *allegro news* selber einem Schlussstrich ganz nahe gekommen sind! Zwar sind die *news* zur Zeit noch für nicht *ganz* wenige Anwender die einzige Informationsquelle für Neuerungen, außerdem sind sie nebenbei auch so etwas wie eine Chronik der Entwicklung, aber der Aufwand ihrer Erstellung und Produktion ist denn doch kritisch zu betrachten. Als Alternative steht im Raum, im Jahr zwei CDs zu produzieren und jeweils den Inhalt des Web-Servers und das Archiv der E-Mail-Liste (mit den sog. "Verlautbarungen") sowie sonstiger Dokumentationen und auch Datenbanken darauf zu verbreiten. Hören wir Protest oder Zustimmung?
Aber *zwei* Schlussstriche sind definitiv bereits gezogen: erstens unter die alte Rechtschreibung (schon in der vorigen Nummer, nachdem selbst der SPIEGEL umgefallen war), zweitens unter die Gedichte, die seit 1992 (ab Nr. 25) die "news" begleiteten. Oben steht das letzte und begründet selber diese Tatsache. Zum 50. Geburtstag des Verfassers und zum gleichzeitig stattfindenden Expertentreffen 1999 hat *Allegrologin* Anette Wegelt unter dem Titel "allegro spiritoso" eine Sammlung von 50 solchen Gedichten herausgegeben, also ein paar mehr als die in den "news" erschienenen. Damit soll's nun gut sein.
Einen Schlussstrich zieht, last not least, dieser Tage auch unser hoch geschätzter Freund und Kollege **Dr. Peter Pfeiffer** in Wolfenbüttel, ehemals ***allegro***-Pionier an der Staatsbibliothek Unter den Linden, dem wir unter anderem die UNIX-Portierung verdanken: er begibt sich in seinen wohlverdienten Ruhestand. Viele werden mit uns froh sein, dass er nur seine berufliche, nicht aber seine ***allegro***-Laufbahn damit an den Nagel hängt! Eher im Gegenteil - dafür hat er dann erst richtig Zeit ...

a99/alcarta : Die FLEX-Sprache

Erneut soll auf den folgenden Seiten eine zusammenfassende, vollständige Darstellung der FLEX-Befehle auf dem aktuellen Stand gegeben werden. Die erste Beschreibung des Konzepts und der Befehle erschien in Nummer 54 und wurde in 55 ergänzt. Danach kamen aber noch, immer wieder angeregt durch engagierte *Allegrologen*, zahlreiche Neuerungen und Verbesserungen, über die bisher nur in der E-Mail-Liste berichtet wurde. Hier sind sie mit **NEU** am rechten Rand markiert. Es wurden außerdem etliche Details an den Randbedingungen einzelner Befehle verbessert, um Irritationen abzufangen.
Die nachfolgende Beschreibung ist auch im Programm abrufbar: man gibt im Schreibfeld ein: **`h flex`**, dann wird der Hilfetext flexger.rtf geladen (bzw. flexeng.rtf, wenn man die Sprache Englisch eingestellt hat). Mit dem Button [Drucker] kann man die Dokumentation, wie auch jeden anderen Hilfetext, sofort zu Papier bringen. (Das eben konfrontiert uns mit der Frage, ob man nicht schon bald auf gedruckte Dokumentationen weitgehend wird verzichten können, siehe S. 1. Auch andere Dateien lassen sich so sichtbar machen: geben Sie mal **`h a99.ini`**).
Die FLEX-Befehle bilden eine Art Makrosprache für das *allegro*-System unter Windows. Fast alle Vorgänge, die man mit der Hand auslösen kann, sind auch über einen FLEX-Befehl ausführbar.
Wichtig: es gibt eine **interne Variable** (im Folgenden kurz **iV**). Das ist ein temporärer Zwischenspeicher, der nur so lange lebt, bis er durch einen weiteren Befehl überschrieben wird. Das geschieht z.B. durch die Befehle "ask", "date" und "variable".
Die **iV** hat und braucht keinen Namen, d.h. man sieht sie in den Befehlen nicht, doch das vereinfacht und verkürzt viele Befehle. Man verwendet die interne Variable in der Regel sofort z.B. in einem "insert"-Befehl (z.B. `insert #98` , um den Inhalt der iV in die Kategorie #98 zu kopieren). Will man die iV länger aufbewahren: mit `insert #uxy` in eine Anwendervariable #uxy kopieren. (Und von dort bei Bedarf wieder zurück in die iV mit dem Befehl `variable #uxy`.)

So erstellt man FLEXe

A. Interaktiv
Der einfachste Weg, einen FLEX zu erstellen und auszuführen ist dieser:

1. **Eingeben:** Befehlsfolge im Schreibfeld eintippen, \ als Trennzeichen, am Anfang eine Ziffer 0, 1, ... 9
   Beispiel: `3 message Index 1 wird gezeigt\index |1shakesp`
   [Enter] drücken. Der FLEX wird in Variable #uX3 gespeichert
2. **Ausführen:** Alt+Ziffer (Alt+3 für das Beispiel, um #uX3 zu aktivieren)
3. **Ändern:** Alt+r um den Reservespeicher zu sehen, Zeile #uX3 auswählen
   dann [Enter]; Befehlsfolge ändern, nochmals [Enter] zum Zurückspeichern. Mit Alt+3 wieder ausführen.

Auf diese Weise kann man sich bis zu 10 FLEX-Makros einrichten, die zusammen mit den Phrasen abgespeichert werden und somit für nachfolgende Sitzungen erhalten bleiben.
Nur die FLEXe in den Variablen **`#uX`**`i` kann man mit der Alt-Taste auslösen, sie haben also eine Sonderstellung.
Ein FLEX kann auch in einer Datei vom Typ .FLX stehen (ein Befehl pro Zeile). In der #u-Variable steht dann X *dateiname.*
Z.B. gibt es für die Erwerbungsfunktionen (Modell ORDA, s.S. 11) mehrere solche FLEXe: BESTELL.FLX etc.

B. Einbettung in Parameter oder Hilfetexte
Andere Wege, FLEXe bereitzustellen, sind das Einbetten in Hilfedateien oder in die Anzeigeparameter. Das jedoch sind Aufgaben für *AllegrologInnen.* Es gibt etliche vorbereitete Parameterdateien (D-WRTF.APR) und Hilfedateien wie z.B. input.rtf oder catger.rtf, die man als Vorlage heranziehen kann. Man gibt im Schreibfeld ein: `h input.rtf` statt nur `h input`, dann sieht man auch die eingebetteten FLEXe (Zeilen, die mit '?' anfangen). Bei diesen FLEXen muss es immer zu einem `#uYi` ein `#uZi` geben, und der Inhalt des `#uYi` muss einer Zeichenkette im Anzeigefeld entsprechen. Wird dieses angeklickt (doppelt), sucht das Programm das gleichlautende #uYi und führt das zugehörige #uZi aus.

C. Phrasenspeicher **NEU**
Zum Dritten kann man FLEXe im Phrasenspeicher unterbringen! Wird dann die Phrase angefordert, wird statt dessen der FLEX ausgeführt. Beispiel: wenn in PHRASE.A99 diese 4 Zeilen stehen:

```
6   (steht für Strg+f)
x var #uSW(1,0)\ins #uSW\var "="\ask +Suchwort?=#uSW\help\ins #uSW
7   (steht für Strg+g)
x var ">" #uSW(1,0)\ins #uSW\help
```

Dann wird mit Strg+f die Phrase 6 (Suchen im Anzeigefeld) und mit Strg+g die Phrase 7 (Weitersuchen) ausgeführt. Und zwar braucht dabei der Cursor nicht im Schreibfeld zu sein, das klappt auch, wenn er woanders ist. (S. Befehl **`help`**)
Achtung: die Werte 8, 10 und 13 dürfen nicht belegt werden, das sind die Funktionen "Backspace", "Enter" und "Return". Dagegen ist 9 der Wert TAB, d.h. wenn man Phrase 9 belegt, kann man diesen FLEX auch mit der TAB-Taste auslösen.
Phrasen können statt mit der Hand oder durch Eintragung in die Datei PHRASE.A99 auch mit dem Befehl **`phrase`** belegt werden (s.u.)

D. AutoFLEXe **NEU**
Eine Sonderstellung haben die Variablen **`#uX:`** und **`#uX;`** : wenn man sie aus den Anzeigeparametern heraus mit einem FLEX belegt, wird dieser sofort nach Fertigstellung der Anzeige ausgeführt. Im Falle von **`#uX:`** nur dann, wenn gerade kein Index- und kein Ergebnisfenster geöffnet ist, im Falle **`#uX;`** aber ohne Einschränkung sofort.

Liste der FLEX-Befehle

#nnn*Text*

Die Kategorie #nnn*Text* wird in den aktuellen Satz eingefügt. Wenn *Text* leer ist, wird #nnn gelöscht.
#nnn kann auch eine Nutzervariable #uxy sein.
Wenn der Text auf kompliziertere Weise zusammengesetzt werden soll, muss man zuerst mit Hilfe des Befehls **variable ...** den Text in der internen Variablen erstellen, dann mit **insert #nnn** abspeichern.

$a+#nnn*Text*

Das Teilfeld ▼a mit Inhalt *Text* wird an die Kategorie #nnn angehängt. Wenn #nnn noch nicht da ist, entsteht sie.
Statt '$' kann das Teilfeld-Dreieck verwendet werden. z.B.
▼u+#90 Meier Wirkung: ▼uMeier an #90 anhängen

$a-#nnn*Text*

Wenn Teilfeld ▼a existiert, wird es ersetzt, sonst wird ▼a*Text* an #nnn angehängt.
Wenn *Text* leer ist, wird das Teilfeld ▼a aus #nnn beseitigt.

***#nnn_ABC_XYZ_** Ersetze in Kategorie #nnn des aktuellen Satzes die Zeichenkette "ABC" durch "XYZ"

_ABC_XYZ_ Ersetze die Zeichenkette "ABC" im gesamten Satz durch "XYZ"

? |i*abc*

Register i aufblättern an der Stelle *abc,* z.B. ? |1goethe
Der FLEX wird dann erst fortgesetzt, wenn man den Index verlässt. Daher kann man mehrere solche Befehle aufeinander folgen lassen und andere Befehle dazwischen schalten.

:*label*

Sprungmarke. Mit **jump** *label* kann hierher gesprungen werden. Wichtig beim Befehl **if**

Von den nachfolgenden Befehlen genügen jeweils die ersten 3 Buchstaben, also z.B. "dow" statt "download", "var" statt "variable", "ins" statt "insert", aber auch "down" oder "disp" etc. würden funktionieren.
(Bei ***avanti*** muss man die Befehlswörter vollständig angeben. Wenn man FLEXe schreibt, die auch als ***avanti***-Aufträge funktionieren sollen, muss man das beachten.)

activate

Nur sinnvoll in einem ExFLEX (vgl. "news" 54, S.10), um das Programm in den Vordergrund zu bringen
Siehe Beispiel in der Beschreibung zur Fremddaten-Methodik, S.13

ansi Die interne Variable ist im ASCII-Code, mache daraus ANSI (Windows) **NEU**
ascii Die interne Variable ist im ANSI-Code, mache daraus ASCII (DOS). Kann nach **read** nötig sein. **NEU**

ask |i*prompt***=***Vorgabe*
ask +|i*prompt***=***Vorgabe*

Aufforderung zu einer Eingabe. Die eingegebene Zeichenkette wird in der **internen Variablen** gespeichert.
Wenn ein + gesetzt ist, wird die Eingabe an die iV hinten angehängt.
Für die Eingabe erscheint ein kleines Dialogfenster mit einer Eingabezeile.
i = Nummer des Index, der aufgeblättert wird, wenn man [Index] drückt
prompt = Aufforderungstext
(Wenn man |**i** weglässt, kommt ein rein zufälliger Indexabschnitt)
Leereingabe führt zum Abbruch der Befehlskette.
Der Teil "=*Vorgabe*" kann entfallen; der Text *Vorgabe* erscheint im Eingabefenster.
Vorgabe kann auch eine Kategorie oder eine #u-Variable sein:
ask |1Verf.?=#40 legt den Inhalt von #40 als Vorgabe vor.
Die Eingabe geht aber nicht direkt zurück in die Kategorie, sondern in die iV und muss mit insert #40 zurückübertragen werden, falls gewünscht.
Statt der gesamten Angabe |**i***prompt***=***Vorgabe* kann auch eine Kat.nummer oder #u-Variable stehen, dann muss deren Inhalt diesem Text entsprechen. So können Fragetexte und Vorgaben satzspezifisch individualisiert werden.
Mit nachfolgendem **if "" ...** kann man abchecken, ob etwas eingegeben wurde!
Ob die Aufforderung mit [Esc] abgebrochen wurde, kann auch mit **if cancel ...** festgestellt werden. Die iV ist dann ebenfalls leer.

call *Programmaufruf*

Hinter "call " kann man einen kompletten Startbefehl für ein externes Programm hinschreiben, wie beim Flipbefehl ~.
Es kann sich auch um einen DOS-Befehl handeln, z.B. `call x d-wrtf.apr`
oder `call rename e.adt f.adt`
Wenn das externe Programm eine Datei E.ADT produziert, kann diese anschließend mit "`read`" (s.u.) eingelesen werden: sie wird dann in den aktuellen Satz eingemischt. So kann man von außen Daten einschleusen.
Die Abarbeitung des FLEX geht erst weiter, wenn man das externe Programm verlässt.

choose *Suchbefehl* **NEU**

Es wird die Ergebnismenge gebildet *und* angezeigt. Anders als bei **Find** wird der FLEX nicht beendet.
Die Kurzzeile des ausgewählten Satzes wird in die **iV** kopiert - mehr passiert nicht. Mit dem Befehl **load** aber kann man gleich danach den ausgewählten Satz laden lassen, wenn damit etwas passieren soll.
Wird die Ergebnismenge mit [Esc] verlassen, ist die iV leer. Außerdem kann man den Abbruch auch mit **if cancel ...** checken.
Fehlt der *Suchbefehl*, wird dafür der Inhalt der iV genommen.

copy (wie Button [Neusatz], Funktion "Kopie", siehe auch Befehl **new**)

Der aktuelle Datensatz, so wie er gerade ist, wird als neue Kopie behandelt. Nachfolgend gibt man z.B. `#nnn-` oder `insert-`Befehle und/oder Ersetzungen, dann evtl. **put** zum Speichern des neuen Satzes.

date *b*

(*b*=Breite des Datums, 8-17 Byte; 8 ist nur das Datum, 17 ist Datum/vollständige Uhrzeit)
Datum (und Uhrzeit) werden in die interne Variable kopiert.
Diese speichert man mit `insert ...` in eine Kategorie oder ein Teilfeld oder eine #uxy

delete *ABC*

Datei *ABC* löschen. Das ist nützlich z.B. für die Datei E.ADT, wenn man ein Fremddaten-Kopierverfahren organisieren will.

display *xABC*
display +*xABC*
deposit *xABC*

Abschnitt `#-x` in den Anzeigeparametern ausführen. Die Zeichenkette *ABC* ist dann in der Variablen #u1 enthalten.
An den betr. Sprungmarken können z.B. Manipulationen am Datensatz stattfinden. Die gesamte Mächtigkeit der Exportsprache kann somit in die Tätigkeit eines FLEX einbezogen werden.
Ist '+' gesetzt, wird das Ergebnis an die vorhandene Anzeige *angehängt,* sonst würde sie überschrieben.
Der Befehl `deposit` bewirkt, dass die Anzeige sich nicht ändert, d.h. die Verarbeitung nur intern durchgeführt wird. Das ist nützlich, wenn man komplizierte Manipulationen machen will, die mit der FLEX-Sprache allein nicht gehen. Man verwendet dann #u1, um in den Anzeigeparametern zu entscheiden, was zu tun ist. Das Ergebnis der Verarbeitung ist dann in der Internen Variablen deponiert - daher der Name "deposit". Also: mit den zwei Befehlen
`deposit`
`write`
kann man die Anzeigeform in die Ausgabedatei überführen! Das wird man selten machen. Das Normale wird sein, dass man einen bestimmten Abschnitt ausführen lässt, um z.B. #u-Variablen zu besetzen oder eine Zusammensetzung aus mehreren Kategorien oder -bestandteilen in die interne Variable zu bringen. Man gibt also normalerweise `deposit yTEXT.` Damit wird der Abschnitt `#-y` ausgeführt, und es liegt dort `#u1 TEXT` vor.
Wenn eine Ausgabe entsteht, hat man sie anschließend in der iV und kann sie mit "write" ausgeben lassen oder mit "insert" in eine Kategorie kopieren, oder aber ignorieren, wenn es nur auf die internen Manipulationen ankommt.

display p *name* **NEU**

Lade andere Anzeigeparameter, dann sofort Anzeige d. aktuellen Satzes.
z.B. **display p d-alfa** Lade D-ALFA.APR

download

Aktuellen Satz exportieren (wie über Menü Export). Soll nur ein bestimmter Abschnitt in den Exportparametern ausgeführt werden: vorher eine Variable besetzen, z.B. #uFL (mit Befehl `var xxx\ins #uFL)`, und in den Parametern am Anfang einen Sprung einbauen: `#uFL +X e0`

download set

Aktuelle Ergebnismenge exportieren (wie über Menü Export)

end Der FLEX wird beendet. Sinnvoll oft nach **yesno** und **noyes** , z.B **NEU**
if no end wenn [Nein] geantwortet wurde, dann Ende

erase
Der aktuelle Satz wird gelöscht. Dazu braucht nicht in der .INI-Datei `access=3` gesetzt zu sein, d.h. man kann das Löschen in erwünschten Fällen per Parametrierung ermöglichen. Sogar unter ***alcarta***!)Eine Gefahr des Missbrauchs besteht dort nicht, weil es kein Schreibfeld zum Eingeben eigener FLEXe gibt.)
Ist der Satz ein Offline-Satz, wird er ungültig gemacht; war er gerade ganz neu, verschwindet er aus der Anzeige.

extern (wie Alt+t)
Der aktuelle Satz wird in E.ADT ausgegeben und der externe Editor aufgerufen.
Nach Rückkehr kann der Satz wieder eingelesen werden.

file *dateiname* (Wenn Name mit '+' beginnt: anhängen, sonst überschreiben) **NEU**
Anzeigefenster in Datei ausgeben. Wenn Dateityp .rtf, dann als RTF-Datei, sonst als ASCII-Textdatei.
Zusammen mit **help** kann man also ASCII- und RTF-Dateien im Anzeigefenster bearbeiten (bis 64K).

find
find *suchbefehl*
Find **NEU**
Find *suchbefehl*
Bilde eine Ergebnismenge. Diese wird bei 'f' nicht angezeigt, sondern sie wird anschließend als aktuelle Ergebnismenge benutzt, wenn man mit den Befehlen `next, prev`, oder `download set` arbeitet.
Wenn *suchbefehl* fehlt, wird der Inhalt der iV benutzt. Wenn man diese vorher geeignet besetzt, kann man den Suchbefehl in Abhängigkeit vom aktuellen Satz gestalten. Der erste Satz der Ergebismenge wird sofort geladen, wird also zum aktuellen Satz.
Wenn *suchbefehl* mit dem Zeichen + beginnt, wird die nachfolgende Zeichenkette im Volltext der aktuellen Ergebnismenge gesucht, wie bei der Eingabe per Hand im Suchbefehlsfeld.
Bei F statt f wird die Erg.Menge sofort angezeigt und der FLEX *beendet*.
Mit **if empty ...** kann man die Situation abfangen, dass nichts gefunden wurde.
Eine noch etwas andere Variante ist **choose ...** (der FLEX wird dabei *nicht* beendet)

Find offline Sonderfall! Die Offline-Datei wird gezeigt **NEU**

flex *xyz* **NEU**
Senden einer Mitteilung an ein anderes ***a99/alcarta***.
Es wird eine "ExFLEX"-Botschaft an das System gesendet. Wenn ein anderes *a99* oder *alcarta* läuft, kann es diese auffangen, die FLEX-Datei *xyz.flx* lesen und ausführen. Wenn *xyz* fehlt, wird der Inhalt der iV genommen.
Beispiel: s. S.13 im Verfahren für das Umschalten zwischen zwei Datenbanken.

form *i*
Formular i wird aufgeblättert (i = 1...) (in der Reihenfolge der .FRM-Datei).
Man kann mehrere solcher Befehle hintereinander schalten. Statt der Nummer i kann auch die Überschrift des Formulars angegeben werden, z.B. **form Buch**. wenn es ein Formular mit der Überschrift [Buch] gibt. **NEU**
Es muss nicht die gesamte Überschrift sein, ein eindeutiger Anfangsteil genügt.
Mit **if cancel ...** kann man anschließend checken, ob das Formular mit [Esc] verlassen wurde.

help *name*
help **+***name* **NEU**
Hier ist *name* der Name einer RTF-Datei (ohne .rtf), welche dann als Hilfetext in das Anzeigefenster geholt wird. Entspricht dem Flip h. Statt RTF darf es auch eine gewöhnliche ASCII-Datei sein. Das Programm setzt als Kopf HELPHEAD.RTF davor, damit sie angezeigt werden kann - sie erscheint dann natürlich in ganz schlichter Form.
Ist der Name einschl. .RTF angegeben, kann man auch die eingebetteten Flips sehen. Das sind Zeilen, die mit ? beginnen, und zwar stehen sie in der RTF-Datei am Ende, immer in dieser Form:
`\par ?Fliptext=x ...`
Irgendwo im normalen Text der Hilfedatei muss dann *Fliptext* stehen, und zwar links und rechts davon entweder eckige Klammern oder je ein Code 160, den man mit Alt+0160 eingibt. Meistens lässt man den *Fliptext* blau und unterstrichen erscheinen, damit intuitiv klar wird, dass es ein Flip ist.
Wenn ein '+' vor dem Namen steht, wird die Datei an den Text angehängt, der schon in der Anzeige steht, anstatt ihn zu überschreiben.
Sonderfall: Steht ein ! statt *name,* kommt die zur Datenbank gehörige Hilfeseite, z.B. catger.rtf.
Fehlt *name* völlig, wird der Inhalt der iV statt dessen genommen. So kann eine vom Satzinhalt abhängige Hilfedatei angefordert werden, oder aber ein Suchbefehl für das Anzeigefeld:
Mehr über das Hilfesystem: siehe die Datei HELP.RTF, siehe auch S. 11: ORDA (Menü in Hilfedatei)

help =*suchwort* **NEU**
help >*suchwort* **NEU**

Dies sind Suchbefehle für den Inhalt des Anzeigefeldes!
Im Anzeigefeld wird das erste Vorkommen von *suchwort* gesucht, wenn '=' davor steht, bzw. das nächste (in Vorwärtsrichtung), wenn '>' davor steht. Nützlich ist das vor allem, wenn lange Hilfetexte im Anzeigefeld stehen, es funktioniert aber bei jeder Art von Anzeige, also auch bei langen Datensätzen mit vielen Nachladungen.
Mit zwei FLEXen kann man ein bequemes Suchen organisieren:

```
#uX8x x var #uSW(1,0)\ins #uSW\var "="\ask +Suchwort?=#uSW\help\ins #uSW
#uX9x var ">" #usw(1,0)\ins #usw\help
```

Mit Alt+8 wird dann nach dem Suchwort gefragt und nach dem ersten Vorkommen gesucht, mit Alt+9 wird die Suche fortgesetzt.
Alternativ: man gibt im Schreibfeld diese Phrasen ein:

```
p ^f x var #uSW(1,0)\ins #uSW\var "="\ask +Suchwort?=#uSW\help\ins #uSW
p ^g x var ">" #usw(1,0)\ins #usw\help
```

Dann erfolgt die Suche bzw. Weitersuche mit Strg+f bzw. Strg+g.

if *#nnn command* **NEU**
if *#nnn$a command* **NEU**

Wenn #nnn bzw. Teilfeld a in #nnn existiert (besetzt ist), wird *command* ausgeführt,
z.B. kann es ein **jump ...** sein.

Bei den folgenden Bedingungsprüfungen genügt jeweils der erste Buchstabe, also z.B. **if c**
Bei vielen anderen Befehlen ist jeweils angegeben, ob man anschließend mit **if** ... eine Bedingung checken kann.

if yes *command* [gleichwertig: **if ok**...] **NEU**

Wurde die letzte **yesno** oder **noyes**-Frage mit "Ja" beawortet ...

if no *command* **NEU**

bzw. mit "Nein", dann wird *command* ausgeführt

if cancel *command* **NEU**

Wurde die letzte **yesno** oder **noyes**-Frage mit [Abbruch] oder [Esc] beantwortet,
wurde das vorangehende **ask** oder **select** mit [Abbruch] oder [Esc] verlassen,
war das Dateiende erreicht (d.h. konnte mit **read** nichts mehr gelesen werden?)

if main *command* **NEU**

Ist der aktuelle Satz ein Hauptsatz einer verknüpften Familie? ...

if fam *command* **NEU**

Ist der Satz ein Mitglied einer verkn. Familie (Haupt- oder Untersatz)? ...

if sub *command* **NEU**

Ist der Satz ein verknüpfter Untersatz? ...

if hiera *command* **NEU**

Ist es ein hierarchischer Satz? ...

if diff *command* **NEU**

Wenn der aktuelle Satz vorher verändert wurde (Hintergrund ist gelb!) wird *command* ausgeführt, z.B. **put**.

if empty *command* **NEU**

Wenn die aktuelle Erg.Menge leer ist, wird *command* ausgeführt.

if "xyz"*command* **NEU**

Wenn die iV mit xyz anfängt, wird *command* ausgeführt. Soll der Inhalt einer Kategorie oder eines Teilfelds abgeprüft werden, bringt man diesen Inhalt zunächst mit **var ...** in die iV und wendet darauf dann Befehle der Form **if "xyz"** an.
Sonderfall: Mit **if "" ...** kann man prüfen, ob die iV leer ist (z.B. nach **ask**)

index |**i***abc* [siehe auch oben **?...**] **NEU**

Register i aufblättern an der Stelle *abc,* z.B. `ind |1goethe`

input *n*

Damit kann die Nummer der Datei verändert werden, in welche die neuen Datensätze zu speichern sind. Sichtbar wird das nicht.
Allerdings muss die Berechtigung (`access=` in der INI-Datei) mindestens 2 sein.

insert #nnn
Inhalt der internen Variablen in die Kat. #nnn kopieren.
Wenn man `insert #uxyABC` schreibt, wird noch ABC vor die interne Variable gesetzt.
Der Befehl nimmt also den nachfolgenden Text, hängt die iV hinten an, und interpretiert das Ganze als Kategorie.

insert $a+#nnn
Inhalt der internen Variablen als Teilfeld $a an #nnn anhängen

insert $a-#nnn
Teilfeld a ersetzen, wenn es vorh. ist, sonst anhängen. Wenn die interne Variable leer ist: Teilfeld $a löschen (z.B. **var ""\ins #nnn**)
Sonderfall: **insert $$-#nnn** Setzt die iV an den Anfang des Kategorietextes, ohne Teifeldzeichen.

jump *label* **NEU**
Sprung zur Sprungmarke *:label* (s. oben)

Jump xTEXT
wie **display**, aber Ausgabe per Exportparameter statt Anzeigeparameter.
Sicherer funktioniert die Ausgabe per **download**, siehe dort.

load **NEU**
Nur nach vorangegangenem **choose ...** (siehe dort)
Der ausgewählte Satz wird geladen. Nochmaliges **load** lädt wieder den Ausgangssatz.

menu *Text* **NEU**
Dient zur Flexibilisierung des Hauptmenüs.
Der anwendungsspezifische Menüpunkt zwischen "Option" und "?" wird auf den Wert *Text* gesetzt. Dieser Menüpunkt löst den FLEX aus, der in #uXz steht. Dies kann vorher oder im selben FLEX beliebig vorbereitet werden. Wenn *Text* fehlt, wird der Inhalt der iV genommen. Man kann also vorher mit dem **var**-Befehl die iV vorbereiten. z.B.
var #uxy
menu
Als Default für den anwendungsspezifischen Menüpunkt dient der Text in Zeile 243 der Datei UIFEGER.

message *Text*
Text wird als Meldung in einer Box angezeigt und muss mit [OK] bestätigt werden, sonst geht's nicht weiter.
Wenn der Nutzer etwas entscheiden soll, nimmt man statt dessen den Befehl **yesno**
Wenn er etwas eingeben soll: Befehl **ask**

new (wie Button [Neusatz], siehe auch Befehl **copy**)
Es wird ein neuer, leerer Datensatz angelegt. Der aktuelle Satz wird vorher in den Offline- und in den Hintergrundspeicher kopiert. Nachfolgend: manuelle Eingabe oder #- und `insert`-Befehle, bzw. `transfer`, um Kategorien aus dem Hintergrundspeicher zu übernehmen!

next [Gegenteil: **prev**]
Der nächste Satz der Ergebnismenge, in der vorher eingestellten Sortierfolge, wird geladen und angezeigt.
Mit **if yes/no ...** kann man checken, ob es einen nächsten Satz gab. Wichtig bei Schleifen! **NEU**

open *name* **NEU**
Die Datei *name* wird zum Lesen geöffnet. Anschließend kann mit **read...** daraus gelesen werden. Am Ende des FLEX wird die Datei automatisch geschlossen.
Fehlt *name,* wird die iV genommen.
Mit **if yes/no ...** kann man checken, ob das Öffnen gelungen ist oder nicht. Also kann man mit der Kombination **open** *name***if no jump ...** auch die Existenz einer Datei prüfen.
Kommt innerhalb eines FLEX ein weiteres **open**, wird die erste Datei geschlossen.

order *M P*
Ergebnismenge ordnen (sortieren)
M = Modus: a=aufsteigend, d=absteigend, n=Nach Satznummern
P = Position des Sortierfeldes in der Kurzliste: Das erste Zeichen ist Position 0

phrase *i text* **NEU**

Hiermit kann man Phrasen belegen. Der Wert i kann eine Zahl >0 sein oder ein Buchstabe mit ^ davor, z.B. ^z (gleichwertig mit 26). Wenn *text* ein FLEX-Befehl ist (beginnt mit "x " oder "X "), wird hinterher beim Auslösen der Phrase dieser FLEX ausgeführt, also z.B. mit Strg+z. (Achtung, nicht benutzen: 8,9,10,12 bzw. ^h, ^i, ^j, ^m)
Wenn *text* fehlt, wird Phrase *i* gelöscht.

prev [Gegenteil: **next**]

Der vorige Satz der Erg.Menge wird geladen und angezeigt
Abbruch der Befehlskette erfolgt, wenn es keinen vorigen gibt (sondern der erste geladen ist).
Mit **if yes/no ...** kann man checken, ob es einen vorigen Satz gab. Wichtig bei Schleifen! **NEU**

print

Anzeigefenster ausdrucken (wie Print-Button)

Put Speichern mit Rückfrage
put Speichern ohne Rückfrage

Der aktuelle Satz wird gespeichert (wie bei [Speichern]-Button)

Put new
put new

Der aktuelle Satz wird als neuer Satz gespeichert. (Dasselbe würde `copy\put` bewirken.)
War es ein Online-Satz und wurden keine Veränderungen gemacht, entsteht damit eine Dublette.

qrix ... **NEU**

Identisch mit **index** , siehe dort (nur wegen Kompatibilität mit ***avanti***).

read

Datei E.ADT wird eingelesen. Diese kann eine Reihe von Kategorien enthalten, die dann alle in den aktuellen Satz eingefügt werden. (Erfassungshilfe)

read extern **NEU**

Datei EXTERN.DAT wird als externe Ergebnismenge eingelesen (wie über das Menü "Datei").

read #xyu **NEU**
read iv

Aus der mit **open ...** geöffneten Datei wird eine Zeile gelesen und in die Kategorie #xyz bzw. in die iV kopiert.
Mit **ascii** kann man die iv umwandeln. Wenn also die Datei ANSI-Daten hat, aber die Datenbank ist ASCII: **read iv\ascii\ins #uxy** statt nur **read #uxy**.
Mit **if cancel ...** kann man checken, ob die Datei zu Ende war, mit **if no ...** ob sie gar nicht geöffnet war, mit **if yes ...** ob es geklappt hat.

repeat

Die gesamte Befehlskette wird so lange wiederholt, bis ein Befehl nicht ausführbar ist oder **end** kommt.
Sinnvoll in Verbindung mit `next / prev`, um automatische Schleifen zur Abarbeitung von Ergebnismengen zu bilden. **repeat** kann nur als letzter Befehl in einer Kette stehen (wenn noch was folgt, wird es ignoriert)
Während des Ablaufs einer Schleife kann man mit 'x' unterbrechen und dann wahlweise weiterlaufen lassen oder abbrechen. Das geht bei einer mit **jump ...** gebildeten Schleife nicht:
Schleifen kann man auch mit if... und jump... bilden, man muss aber aufpassen, dass irgendwann eine Ende-Bedingung erreicht wird, sonst hängt das Programm.

select *prompt=antw1|antw2|antw3|...* **NEU**
select +*prompt=antw1|antw2|antw3|...* **NEU**

Auswahlliste anbieten mit den angegebenen Antworten. Die ausgewählte Antwort wird dann in die interne Variable kopiert. Wenn + vor dem *prompt* steht, wird die Antwort an die iV angehängt.
Jede Antwort kann von der Form A=B sein, dann wird nur A in die iV kopiert. Sinnvoll ist das, wenn statt Klartext nur ein Code einzugeben ist.
Mit **if "" ...** kann man testen, ob etwas ausgewählt wurde.
Mit **if cancel ...** kann man checken, ob die Auswahl mit [Esc] verlassen wurde.
Varianten:

select *prompt=***#usp**. **NEU**

In #usp muss dann die Liste der Antworten stehen

select #usp. **NEU**

In #usp müssen dann *prompt und* Antwortliste stehen.

set pX **NEU**

Primärschlüssel für **update** soll an der Sprungmarke #-X bestimmt werden. (default ist @)

set u*xyz* **NEU**

Modus für **update**. x und y wie bei DOS-UPDATE,
z=0 bedeutet: Keine Speicherung, sondern Zwischenlagerung im Offline-Speicher. Die Speicherung kann anschließend über das Menü "Datei" erfolgen - oder auch nicht (Update "zur Probe"!). Mit Alt+w kann man bei jedem Satz besichtigen, wie er vorher ausgesehen hat und wie nach dem Update.

show *xxx* **NEU**

Steuerung des Listenfensters. Für *xxx* kann stehen:

`reserve`	Hintergrundspeicher
`record`	Datensatz-Kategorien
`ergeb`	Liste der Ergebnismengen
`cfg`	Konfiguration
`list`	Kurzliste der aktuellen Erg.Menge anzeigen
`keys`	Registereinträge zum aktuellen Satz anzeigen (im Anzeigefeld)
`iv`	Inhalt der internen Variablen (erscheint im Schreibfeld). Damit kann man im Schreibfeld einen zusätzlichen Hinweis erscheinen lassen, etwa zu einer Hilfeseite.

Jeweils 3 Buchstaben des Befehls genügen, z.B. `sho erg`.

sleep *n*

Das Programm soll *n* Millisekunden untätig verharren, bevor es weiter arbeitet. (Für 3 Sekunden muss man also schreiben: `sleep 3000`)
Nutzbar ist das z.B., wenn man eine Folge von Hilfeseiten abrollen lassen will. Das kann durch eingestreute `yesno`-Befehle noch flexibilisiert werden:

```
help name1\sleep 4000\yesno Weiter?\help name2\sleep 4000...
```

transfer #nnn

Kategorie #nnn aus dem Hintergrundspeicher in den aktuellen Satz kopieren.
Wenn eine Variable #uxy in eine Kategorie #nnn kopiert werden soll: `var #uxy\ins #nnn`

undo

Entspricht dem Button [Wechseln] : Umschalten zwischen Bearbeitungs- und Originalzustand.

upload **NEU**

update **NEU**

Diese Funktionen entsprechen den gleichnamigen ***avanti***-Funktionen. Vor **update** muss mit **set u...** der Modus gesetzt werden.
Das DOS-Programm UPDATE wird dadurch ersetzt. Der Ablauf kann allerdings interaktiv überwacht und kontrolliert werden. Und nicht nur Grunddateien (Typ .ALG) kann man einmischen, sondern auch (wie bei ***avanti***) die einfacher zu erstellenden Externdateien (Typ .ADT).

variable *cstring* (Was ist ein *cstring* ? Siehe unten)

Der *cstring* wird in die interne Variable kopiert. Der *cstring* kann genauso strukturiert werden wie bei dem Befehl **write** (siehe unten)

variable +*cstring*

Inhalt an die iV hinten anhängen. Hiermit kann man beliebige Inhalte zu einer Zeichenkette verknüpfen, z.B.

`var #40 ": " #20`	Inhalt von #40 und #20 in iV kopieren mit ": " dazwischen
`var +" (" #76 ")"`	(Inhalt von #76) an iV hinten anhängen
`ins #upt`	iV in #upt speichern (Die iV als solche bleibt erhalten!)

write

Inhalt der internen Variablen in die Export-Ausgabedatei schreiben

write *cstring* [Das ist genau der ***avanti***-Befehl]

So kann man am schnellsten Satzinhalte und beliebigen Text ausgeben. Der *cstring* kommt zusätzlich in die iV!
Beispiel:

```
write "Titel: " #20 n "Verfasser: " #40 n "Ort: " #74 " (" #76 ")"
```

produziert eine Ausgabe in dieser Form:

```
Titel: Hamlet
Verfasser: Shakespeare, William
Ort: London (1982)
```

Das 'n' ist der Befehl für eine "neue Zeile".
Schlichte Exporte kann man hiermit, genau wie bei ***avanti***, ohne Exportsprache machen.

xport p *name*
Parameterdatei *name.c*PR für den Export laden

xport f *filename*
xport f +*filename*
Nachfolgende Exporte sollen in die Datei *filename* gehen.
Das '+' bewirkt, dass an die Datei angehängt wird (wenn sie schon existiert), sonst wird sie überschrieben.

yesno *Frage*
noyes *Frage*
Die *Frage* wird in einer Ja/Nein/Abbruch-Box gezeigt. Ohne Antwort geht es nicht weiter.
Die Antwort kann mit einem nachfolgenden **if**-Befehl ausgewertet werden, z.B. **NEU**
if no end
Die Antwort bleibt bestehen, bis z.B. das nächste yesno oder noyes oder ask etc. kommt, d.h. es können sich mehrere if-Befehle darauf beziehen.
Bei "noyes" ist der Button [Nein] statt [Ja] der default-Button! Dann muss der User bewusst auf [Ja] gehen - [Enter] bedeutet sonst Nein.

zzz
Kein Befehl. Jedes Nicht-Befehlswort wird ignoriert, kann also als Kommentar benutzt werden. Eine Zeile, die mit Leerzeichen beginnt, wird ebenfalls ignoriert - wie in den Parameterdateien.

Was ist ein *cstring*? (siehe Befehle **variable** und **write**)

Das ist eine Kette von Datenelementen, beliebig zusammengesetzt, wobei die Elemente aus vier Typen bestehen, getrennt durch Leerzeichen (mehrere wirken wie eins, TAB geht nicht):

1. `"xyz"` Zeichenketten, statt "..." geht auch '...'
2. d d d ASCII-Codes als Dezimalzahlen (z.B. 27 69 für Esc E)
3. #nnn Kategorietexte (Es kann auch eine #u- oder Sondervariable sein, z.B. #dt für das aktuelle Datum)
#nnn$a Teilfelder (nur der Inhalt des Teilfeldes wird ausgegeben)

Es gibt für **3.** noch zwei Möglichkeiten der Verfeinerung:

A. An beide Formen kann man einen Ausdruck `(i,j)` anhängen, mit i>=0 und j>=0. **NEU**
Dann wird vom Kategorie- bzw. Teilfeldtext nur der Teil ab Position i genommen (Zählung beginnt bei 0) und zwar j Zeichen (wenn j=0, dann der gesamte Rest - (0,0) wäre also alles; (0,1) wäre nur das erste Zeichen).
Ist i größer als die Länge der Kategorie, kommt nichts heraus.
Ist j größer als die Länge des Restes, wird mit Leerzeichen aufgefüllt! (so kann man Ausgabefelder mit fester Länge erzeugen)

B. Die zweite Möglichkeit: man hängt einen Ausdruck der Form `(b"xyz")` oder `(e"xyz")` an; **NEU**
damit wird aus dem Kategorietext der Teil hinter "xyz" bzw. vor "xyz" herausgelöst, "xyz" selbst wird in beiden Fällen nicht mit ausgegeben (wie bei den Export-Manipulationsbefehlen b und e).

4. Außerdem kann man eine Anzahl *besonderer Variablen* in den Ausgabetext einbauen:
(einfach nur einen der folgenden Buchstaben, außerhalb von "..."; die ersten drei gibt es nicht bei *avanti*)

D	Datenbank-Pfad (INI-Befehl DbDir)	**NEU**
N	Name der Datenbank (INI-Befehl DbName)	**NEU**
f	letzter find-Befehl (Name der letzten Ergebnismenge)	**NEU**
i	interne Nummer des Datensatzes	
l	Größe ("length") der Ergebnismenge	
n	neue Zeile	
p	Primärschlüssel des aktuellen Satzes	
r	relative Nummer des Satzes in der Ergebnismenge	
s	Kurzzeile des Satzes (aus der .STL-Datei)	

Beispiel: **write "Heute ist der " #dt(b", ") n "Sie benutzten die Datenbank " N**
schreibt in die Exportdatei die Worte "Heute ist der " und dahinter das aktuelle Datum ohne den Wochentag, dann eine neue Zeile (#dt hat z.B. die Form "Mon, 17. Jan 2000") und dann den Text "Sie benutzten die Datenbank ...".

FLEX-Beispiele

Beispiel 1 **: Teil-Automatisierung einer Bearbeitung**
Die Aufgabe ist folgende: Man hat eine Menge Datensätze zu bearbeiten, die man jeweils über die Inventarnummer aufruft, dann eine bestimmte Kategorie hinzufügt oder ändert und den Satz dann wieder abspeichert - manchmal aber auch nicht.
Annahme: es gibt ein Register INV im Index 9 mit der Inventarnummer.
Dazu eignet sich ein FLEX nach diesem Muster:

Inventarnummer abfragen (Vorgabe: Inhalt von #uib), Index 9 soll Druck auf [Index] erscheinen
:anfang
ask |9Inventarnummer=#uib
Ist die Eingabe leer? Dann Abfrage wiederholen
if "" jump anfang
Wurde [Esc] gedrückt? Dann Ende
if cancel end
Eingegebene Nummer wieder in #uib kopieren:
ins #uib
Text der internen Variablen zusammensetzen aus "inv " und #uib
var "inv " #uib
und als find-Befehl ausführen (d.h. "find inv ...")
find
Kategorie `#nnn` mit Inhalt *`Text`* einfügen in den geladenen Satz
#nnn *Text*
Fragen, ob gespeichert werden soll
yesno Speichern?
und ausführen, wenn Antwort "OK"
if yes put
Statt der beiden letzten Befehle kann man auch **Put** setzen, dann kommt ebenfalls die Aufforderung zur Bestätigung
Fragen, ob weiter gemacht werden soll: Wenn ja, zurück zum Anfang
yesno Weiter?
if yes jump anfang

Tip: Man schreibt diesen gesamten Text (d.h. das Fettgedruckte reicht) in eine Datei EDIT.FLX und gibt dann im Schreibfeld ein: `#uX1X edit` . Von nun an kann mit Alt+1 dieser FLEX ausgeführt werden.

Beispiel 2 **: Eingabe-Unterstützung**
Neue Datensätze sollen immer schon mit einer Anzahl fester Kategorien vorbesetzt werden. Statt dies über die Formulartechnik zu lösen, kann man auch einen FLEX anlegen, indem man im Schreibfeld eingibt:

```
#uX3x new\#nn1 Text1\#nn2 Text2\#nn3 Text3\trans #nn4\form 2
```

Dadurch wird zuerst ein neuer (leerer) Satz angelegt, dieser mit drei vorbereiteten Kategorien belegt, Kategorie #nn4 wird aus dem Hintergrundspeicher übernommen, dann Formular 2 aktiviert für die weitere Eingabe. Mit Alt+3 aktiviert man diesen FLEX. Man sieht leicht, wie man dieses Beispiel ausbauen kann.
Man kann mehrere solche FLEXe für unterschiedliche Satztypen anlegen.

Größeres Anwendungsbeispiel: ORDA

Mit Hilfe der FLEX-Technik wurde ein Paket von Funktionen geschaffen, das schon weitgehend das konventionelle Erwerbungsprogramm ***ORDER*** ablösen kann: es heißt ORDA (ORDER-Alternative).
Dieses Paket kann sich jeder leicht installieren und es dann ausgiebig testen, denn es enthält auch eine Demo-Datenbank mit allen notwendigen Satztypen und einer ausreichenden Menge von Beispielen. Das Paket liegt unter DEMO2.LZH auf dem FTP-Server; man kopiert es auf ein Verzeichnis (normalerweise C:\ALLEGRO\DEMO2) und packt es mit dem Befehl `lharc x demo2` aus. Danach startet man es von c:\allegro mit dem Befehl **`a99 demo2\orda`**.
Besonders interessant für alle, die etwas Ähnliches machen wollen: die Dateien ORDA.RTF und die FLEXe in den Dateien des Typs .FLX, die alle ausführlich kommentiert sind, z.B. BESTELL.FLX und KONT.FLX. Die Hilfeseite ORDA.RTF fungiert als Hauptmenü des Verfahrens, denn sie besteht praktisch nur aus eingebetteten FLEXen.
Wer mit den Windows-Programmen oder überhaupt mit ***allegro*** noch nichts zu tun hatte, kann sich auch ein komplettes Demo-Paket vom Web-Server herunterladen: siehe **`www.alcarta.com/download.htm`**
So sieht der Bildschirm aus, wenn man auf "Erwerbung" geklickt hat:

Was man im Anzeigefeld sieht, das ist in Wirklichkeit eine Hilfeseite namens ORDA.RTF. Sie funktioniert hier als Menü, denn hinter den blau unterstrichenen Menüpunkten steckt jeweils ein FLEX. Wie funktioniert das?
Der Menüpunkt "Neue Bestellung" z.B. sieht in der Datei ORDA.RTF so aus:

```
\~ \cf2\b\ul Neue Bestellung\plain\~
```

Die Codes \~ stellen das Begrenzungszeichen dar (Code 160, sieht aus wie ein Leerzeichen).
Der Text zwischen den Begrenzungszeichen, "Neue Bestellung", wird in der zugehörigen FLEX-Zeile mit einem FLEX verknüpft, der bei einem Doppelklick auf "Neue Bestellung" dann ausgelöst werden soll:

```
\par ?Neue Bestellung=X bestell.flx
```

Die Datei BESTELL.FLX enthält den eigentlichen FLEX. Sie liegt zweckmäßig auf dem Datenverzeichnis.

In ähnlicher Machart ist auch schon ein **Ausleihpaket** im Test, wodurch das DOS-Programm ALFA abgelöst werden kann.

Fremddaten mit *a99* : Endlich nur noch 1 Knopfdruck

Zu den beliebtesten Eigenschaften von PRESTO gehört es, dass man mit Alt+a auf eine zweite (und dritte) Datenbank umschalten kann und dass in dieser sofort dieselbe Registerstelle aufgeschlagen wird. Mit *a99/alcarta* ist dies etwas schwieriger zu realisieren, jedoch erreicht man letztlich mit ganz kurzen FLEXen sogar einen noch etwas größeren Komfort.
Aufgabe: von der eigenen Datenbank A umschalten auf die Fremddatenbank B. Dort soll dieselbe Registerstelle erscheinen. Ein in B gefundener Satz soll per Knopfdruck von B nach A wandern.

Lösung: Mit Alt+a wie bei PRESTO ist die Sache nicht zu lösen (denn damit wird das Menü "Anzeige" ausgelöst). Statt dessen wollen wir die Umschaltung A→B mit Alt+1, die Rückschaltung B→A mit Alt+9 machen.
Man startet beide Datenbanken getrennt mit je einem Aufruf von *a99*, für B kann es auch alcarta sein. Wir nehmen an, beide Aufrufe erfolgen vom selben Verzeichnis.
In `B.INI` hat zu stehen:

```
ExportParameter=E-W
OutputFile=E.ADT
DbAux=Z      (nur nötig, wenn die zweite Datenbank denselben Namen besitzt; wegen der Hilfsdateien)
```

NEU: Hat die Fremddatenbank ein anderes Format, muss man geeignete Exportparameter statt E-W erstellen. Damit wird auch das direkte Importieren aus anderen Formaten möglich, was mit PRESTO nicht geht.

Umschaltung A → B

Dazu braucht man folgenden FLEX (namens SWITCH.FLX)

```
  Datei alta.flx aufmachen
xport file alta.flx
  Die Befehle hineinschreiben:
write "activate" n "index |" #uxb(0,1) #uxa n
  flex abschicken an das andere a99
flex alta
```

Wenn man den letzten Suchbefehl statt Registerstelle will:

```
   xpo file alta.flx
   wri "activate" n "Find " f
   flex alta
```

Wird dieser FLEX in A aktiviert, schreibt er zuerst eine Datei `alta.flx` und sendet dann die Botschaft an B, diese auszuführen.
In ALTA.FLX steht dann (das entsteht aus der **`write`**-Befehlszeile):

```
activate
index |ixyz
```

wobei *xyz* der letzte Zugriffspunkt im aktuellen Register *i* ist. (Die Schreibweise `#uxb(0,1)` bewirkt, dass nur das erste Zeichen von `#uxb` ausgegeben wird, das ist die Registerziffer.)
Der neue Befehl **`activate`** am Anfang sorgt dafür, dass sich B dann selber aktiviert, also in den Vordergrund kommt.
Dann blättert es Register *i* an der Stelle *xyz* auf.
Es funktioniert in beiden Richtungen! Alt+TAB dagegen schaltet um, ohne dass sich was tut. Das kann ohne Schaden zwischendurch immer mal gemacht werden.

Einfachster Einbau in A: Im Schreibfeld eingeben

```
   #uX1X switch.flx
```

Dann Auslösung mit Alt+1. (Bleibt für nachfolgende Sitzungen erhalten)
Natürlich kann der Aufruf auch in ein Flip zum Anklicken oder auf den eigenen Menüpunkt gelegt werden.

Verbesserung: Die Variable **`#uxa`** enthält ja nicht die Benutzereingabe, sondern die letzte im Register tatsächlich zum Zugriff benutzte Registerstelle. Beim Umschalten will man aber meist an die Stelle, die man im eigenen Register nicht gefunden hatte. Folgendes kann man tun: in den Indexparametern zu den Abschnitten #-1 ... #-9 ergänzt man diese Zeile

```
#u1 =SB e0
```

(evtl. noch y2 oder y1 vor dem =SB.) Und dann verwendet man oben statt **`#uxa`** die Variable **`#uSB`**.

Rückschaltung B → A

Mit dem FLEX

```
x download\flex copy
```

wird zuerst der aktuelle Satz in die Datei E.ADT herausgeschrieben, dann wird A veranlasst, die Datei COPY.FLX zu lesen und auszuführen. In COPY.FLX sollte stehen:

```
    Wenn sofortige Aktivierung von A gewünscht:
  activate
new
read
delete e.adt
    Wenn sofortige Speicherung in Datenbank A gewünscht:
  put
```

So wird durch einen einzigen Mausklick ein gefundener Datensatz der Fremddatenbank in die Datei E.ADT exportiert, vom anderen *a99* dann eingelesen und als neuer Datensatz behandelt. Durch **`put`** wird er sofort gespeichert Wenn **`put`** fehlt, kommt er nur in den Bearbeitungsspeicher (mit Alt+q später aufsuchen).
Der ebenfalls neue FLEX-Befehl **`delete`** löscht die Datei E.ADT, so dass die nächste Aktion in B sofort stattfinden kann.

Jetzt hat man also auch die Wahl: entweder mehrere Übernahmen hintereinander ohne jedesmal umzuschalten (dann kein **`activate`** in COPY.FLX) oder jedesmal umschalten zwecks Bearbeitung, dann kein **`put`** in COPY.FLX.

Das Bearbeiten kann auch in B erfolgen, denn mit `download` wird der aktuelle Zustand exportiert. Hinter `download` empfiehlt sich dann, ein **`undo`** einzuschieben, damit nicht beim Verlassen von B der geänderte Satz dort gespeichert wird.

Einfachster Einbau in B: Im Schreibfeld eingeben

```
#uX9x download\flex copy
```

Dann Auslösung mit Alt+9. (Bleibt für nachfolgende Sitzungen erhalten)
Natürlich kann der Aufruf auch in ein Flip oder auf den eigenen Menüpunkt gelegt werden.

Summa summarum kann jetzt mit ***a99*** nicht nur derselbe Komfort (genau so wenig Tastendrücke) wie bei PRESTO erreicht werden, sondern es gibt auch noch eine Reihe von neuen Möglichkeiten.
SWITCH.FLX und COPY.FLX sind im Paket `a99upd.exe` auf dem FTP-Server enthalten.

Aufbohrung: Dateien können jetzt größer sein als 16 MB

Hinweis: Dieser Abschnitt betrifft nur Inhaber von Großbanken! Wer weniger als 3 oder 4 Millionen Sätze hat und sicher ist, eine solche Größe nicht so bald zu erreichen, kann sich die Zeit für die Lektüre sparen. Wichtig und beruhigend zu wissen ist nur: Die neuen Programme arbeiten mit Alt- und Kleinbanken genauso wie die bisherigen, es gibt nichts zu ändern oder zu beachten, insbes. die Leistung ist nicht verringert und die Programme sind nur ganz geringfügig größer geworden.
Es gibt deshalb jetzt, nach der Bewährungsphase, nur noch die neuen Programme, nicht etwa eine große und eine kleine Programmversion.

Bisher: eine einzelne Datenbankdatei, Typ .cLD, konnte bis zu 16.000.000 Byte groß werden. Bis zu 255 solche Dateien bilden eine Datenbank. Das macht maximal 4 Gigabyte Nutzdaten.

Jetzt: Zwar bleibt es bei maximal 255 Dateien, aber jede davon kann ein Vielfaches der Größe von 16 Megabyte erreichen. Man muss dafür nur einen neuen Befehl in die Indexparameter einsetzen.

Warum das Ganze? Mit 16.000.000 Byte kann man z.B. ca. 16.000 Datensätze in einer Datei speichern, wenn die durchschnittliche Länge ca. 1000 Byte ist, oder ca. 32.000, wenn ein Satz im Schnitt 500 Byte lang ist, usw.
(Wohlgemerkt: die *durchschnittliche* Länge ist maßgebend, nicht die *maximale*, die ist dabei unwichtig.)
Weil es nur bis zu 255 Dateien in einer Datenbank geben kann, ist bei ca. 4 Millionen Sätzen Schluss. (Bei einer mittleren Länge von 500 Byte je Satz natürlich erst bei 8 Millionen.) Weil manche Projekte sich auf diese Gegend zu bewegen, musste etwas geschehen.

Lösung: Mit dem neuen Befehl **`ii`** in den Indexparametern kann man die mögliche Grenze der .cLD-Dateien heraufsetzen, und zwar so:

`ii=2` : verdoppeln (also z.B. maximal 32.000.000 Bytes je Datei)
`ii=3` : verdreifachen ... maximal 48.000.000 ...
usw.

Was ist zu tun?

1. Man holt sich die aktuellen Versionen folgender Programme (vom Verzeichnis PUB/AC15/PROG)
PRESTO.EXE (bei Bedarf auch ALFA, MENUED, GAPAC, PRESTOI, INVENT, REF) (Datum ab 16.11.1999)
APAC.EXE
INDEX.EXE
UPDATE.EXE
SRCH.EXE (das alte tut es auch noch, bis auf Nachladungen)
ACP.EXE (***CockPit***; Korrektur bei Entlüftungs-Aufruf)
(Die Änderungen in der Klassenbibliothek sind ebenfalls erfolgt, daher musste *avanti* nur noch neu kompiliert werden.)
Die UNIX-Programme presto, index, srch und update sind für SUN und Linux ebenfalls angepasst.
Der Z39-Server ist nicht betroffen, da er nicht selber auf die Datenbank zugreift.

2. In die eigenen Indexparameter den neuen Befehl ii einbauen, z.B.

```
ii=2.
```

Faustregel: N Millionen Sätze, dann ii=N/4. Hat man also z.B. 13 Millionen Sätze, dann ii=13/4 = 3.25, also besser 4.
(ausgehend von einer mittleren Länge von 1000 Byte/Satz)
Empfehlung: keine unnötig große Zahl wählen. Runde Zahlen wie 8, 16 oder 32 haben keinen besonderen Vorteil.
Größer als 124 darf die Zahl **`ii`** nicht sein. Das absolute Maximum steigt damit auf 496 GB Nutzdaten.

3. ALD-Dateien neu aufbauen (Tips dazu siehe unten)

4. Arbeiten wie gewohnt. Aber nur noch die neuen Programme verwenden!
In PRESTO kann man jetzt mit Alt+F7 den Wert von `ii` sehen.
Eine parallel geschaltete Datenbank braucht nicht denselben Wert von `ii` zu haben!

Tips zum Neuaufbau

ACHTUNG: Es genügt nicht, nur die TBL-Datei zu erneuern! Vielmehr müssen die Datenbankdateien erneuert werden (und die .TBL damit auch). Anschließend beginnt dann jeder Satz auf einer Position, die durch `ii` teilbar ist, das ist der Punkt.
Im Schnitt werden dann ii/2 Byte je Satz mehr gebraucht, d.h. die ALD-Dateien werden nur ganz geringfügig größer.

A. Entlüften

Das geht am schnellsten. Der Index als solcher braucht ja nicht erneuert zu werden; Entlüften erneuert .ALD und .TBL.
Wenn cat der Datenbankname ist und katalog das Verzeichnis, lautet der Befehl:

```
index -fr0 -d*katalog -ecat/katalog -ka -n1
```

(Nebenbei: die internen Satznummern bleiben bei `-fr` erhalten.)

B. Völlig neu Indexieren

Beim Entlüften bleiben die Dateien als solche erhalten, es werden nicht mehrere zusammengefasst. Jede kann anschließend aber weiter anwachsen.
Auch wenn man per ***CockPit***-Menü "Organisieren" / "Index erneuern" eine Datenbank neu indexiert, bleiben die Dateinummern erhalten. Damit ist in diesem Fall nichts gewonnen. Man wird ja in der Regel mehrere alte .cLD zu einer neuen zusammenfassen wollen.
Dann geht man so vor:

0. Die Datenbank liegt auf ALT (z.B. ALT = F:\ALLEGRO\KATALOG)
1. Neues Verzeichnis NEU anlegen (z.B. NEU = G:\ALLEGRO\KATALOG)
2. Alle Parameter der Datenbank dorthin kopieren, vor allem die Indexparameter (z.B. xyz.api), darin ii einstellen!
3. Folgenden Befehl auf dem Programmverzeichnis geben:

```
index -f70 -dALT -exyz/NEU -ka -n1 -z75000000
```

Hier ist angenommen, die einzelne Datei solle nicht größer als 75 MB werden. Fehlt die Option -z, wird als Default `ii*16000000` genommen.
Man erhält die Liste aller .ALD-Dateien zum Markieren, man setzt '+' vor alle zu indexierenden Dateien (es müssen nicht gleich alle auf einmal sein, man kann in mehreren Abschnitten vorgehen, wenn es sich um sehr große Mengen handelt. Dann bei den nachfolgenden Aktionen -f71 statt -f70 und evtl. -n2 o.a.)

Hat man V14-Ersetzungen oder Schiller-Räuber-Schlüssel, muss man zweistufig indexieren,. Das geht so:

```
index -f70 -@1 -dALT -exyz/NEU -ka -n1 -m0
index -fi1 -@2 -d*NEU -exyz/NEU -ka -m0
```

C. Datenbank aus Grunddateien neu aufbauen

Wenn man vorher noch keine Datenbank hatte, sondern erst eine aufbauen will. Es liegen dann Dateien des Typs .cLG vor, meistens per Import produziert. Wenn diese Dateien abc*.alg heißen und auf DATEN liegen, startet man so:
Ohne V14-Schlüssel:

```
index -f70 -dDATEN\abc -exyz/NEU -ka -n1
```

Mit V14-Schlüssel:

```
index -f70 -@1 -dDATEN\abc -exyz/NEU -ka -n1 -m0
index -fi1 -@2 -d*NEU -exyz/NEU -ka -m0
```

Wird eine bestimmte maximale Dateigröße gewünscht, kann auch hierbei die Option `-z` eingesetzt werden (nur nötig bei `-f7`)
Soll der Aufbau in mehreren Etappen geschehen, muss bei der zweiten und weiteren Etappe `-f71` statt `-f70` gesetzt werden.

Ist die Datenbank fertig, dann ist der wichtige Wert `ii` auch in der TBL-Datei verankert! Selbst wenn die Indexparameter danach versehentlich verändert würden, funktioniert die Datenbank immer noch, denn die Programme entnehmen dann den Wert `ii` der TBL-Datei. Wenn man diesen natürlich selber verändert, geht nichts mehr. (Aber wer macht schon sowas!)

Noch ein Hinweis
Wenn man Füllzeichen definiert hat (Befehl f in der CFG), dann erhält beim Neuaufbau jeder Satz mindestens diese Anzahl Füllzeichen, dann aber zusätzlich noch so viele, dass eine Satzlänge erreicht wird, die durch `ii` teilbar ist. Im Schnitt sind das `ii/2` Füllzeichen zusätzlich. Das gilt hinterher auch für Neuaufnahmen.

Indexieren mit Nachladung

Ein ganz anderes Thema: INDEX kann jetzt auch nachladen.
Das versammelte Expertenkollegium hatte auf dem Treffen im letzten September ein Votum für ein erweitertes INDEX.EXE abgegeben: INDEX müsse endlich das Nachladen lernen. Dafür lagen überzeugende Gründe vor, also wurde ans Werk gegangen.

Die Sache war vor einer Weile schon angegangen worden, aber liegen geblieben, weil gar nicht so einfach. Ein mittlerer Kraftakt hat's aber dann gebracht, es funktioniert.
Das Nachladen geht NUR im zweiten Durchlauf, der mit Option `-@2` gestartet wird. Im ersten Durchlauf (mit `-@1`) müssen dann bekanntlich die Primärschlüssel alle gebildet werden, und NUR diese können zum Nachladen im zweiten Durchlauf dann genutzt werden - andere sind ja noch nicht da.
Auch das Produzieren der Kurzanzeige (.STL) profitiert davon, denn auch im Durchlauf mit `-fs` kann INDEX nachladen (vorausgesetzt, die Indexdatei existiert in dem Moment).
Alte Indexparameter werden unverändert und unverlangsamt funktionieren.
Das Nachladen kostet natürlich Zeit, abhängig von der Menge der Fälle.

Beispiel: man will Untersätze in der Kurzliste mit dem Titel des Hauptsatzes aufscheinen lassen, nicht mit ihrem eigenen (denn oft haben sie keinen)
Dann genügt unter der Sprungmarke `#-0` dieses:

```
...
#00 c"+" e"+" |90          NACHLADUNG des Hauptsatzes, wenn + vorkommt
#20 e" : " U f'"' y0       dessen Titel nehmen
#</                        Ruecksetzen
```

während man vorher nur die mittlere Zeile hatte.

Export

Manipulationsbefehle b, e, c (Handbuch S. 194f, 199)
Die Zeichen # < > haben Sonderfunktionen, wenn sie in einem Ausdruck wie z.B. `b"xyz"` als erstes vorkommen. Befehle wie `... b"#a" ...` oder `... e"#"` funktionieren nicht. Wenn nämlich '#' als erstes Zeichen hinter " steht, interpretiert das Programm die Zeichen dahinter als Kategorienummer und sucht dann nach dem Inhalt der betr. Kategorie - was in den beiden Beispielen nichts ergibt. Diese Sonderfunktion wurde vor allem für den M-Befehl c geschaffen, wirkt aber auch bei b und e.
Wenn man z.B. schreibt
`... c"#uxy" ...` dann wird geprüft, ob im Arbeitstext der Inhalt von #uxy vorkommt, nicht die Zeichenfolge "#uxy".
Ferner kann man schreiben:
`... c">#uxy" ...` dann wird geprüft, ob der Arbeitstext als Zeichenkette (nicht als Zahl, dafür ist Befehl `x">..."` zuständig!) größer ist als der Inhalt der Nutzervariable #uxy.
Abhilfe: Für `e"#"` oder `b"#"` kann man schreiben `e"[#]"` bzw. `b"[#]"`, ansonsten muss man das Zeichen '#' am Anfang eines Suchbegriffs vermeiden.

Stoppwort-Tabelle (Handbuch S.177)
Ein Stoppwort darf kein Leerzeichen enthalten. (Ein Wort auch nicht – dann sind's ja *zwei* Wörter.)

E-Mail-Diskussionsliste

Man schließt sich den ca. 280 Lesern der *allegro*-Liste an, indem man an die Adresse **`maiser@buch.biblio.etc.tu-bs.de`** eine Botschaft mit nur einer Zeile sendet: **`subscribe allegro`** . Man wird dann sofort eingetragen und erhält eine Mitteilung mit weiteren Informationen, insbesondere, wie man sich wieder abmeldet. Es gibt auch ein Archiv der Liste.

Frühere Ausgaben, aktuelle Programme etc.: **FTP:** `134.169.20.1` oder **WWW:** `www.allegro-c.de/allegro`