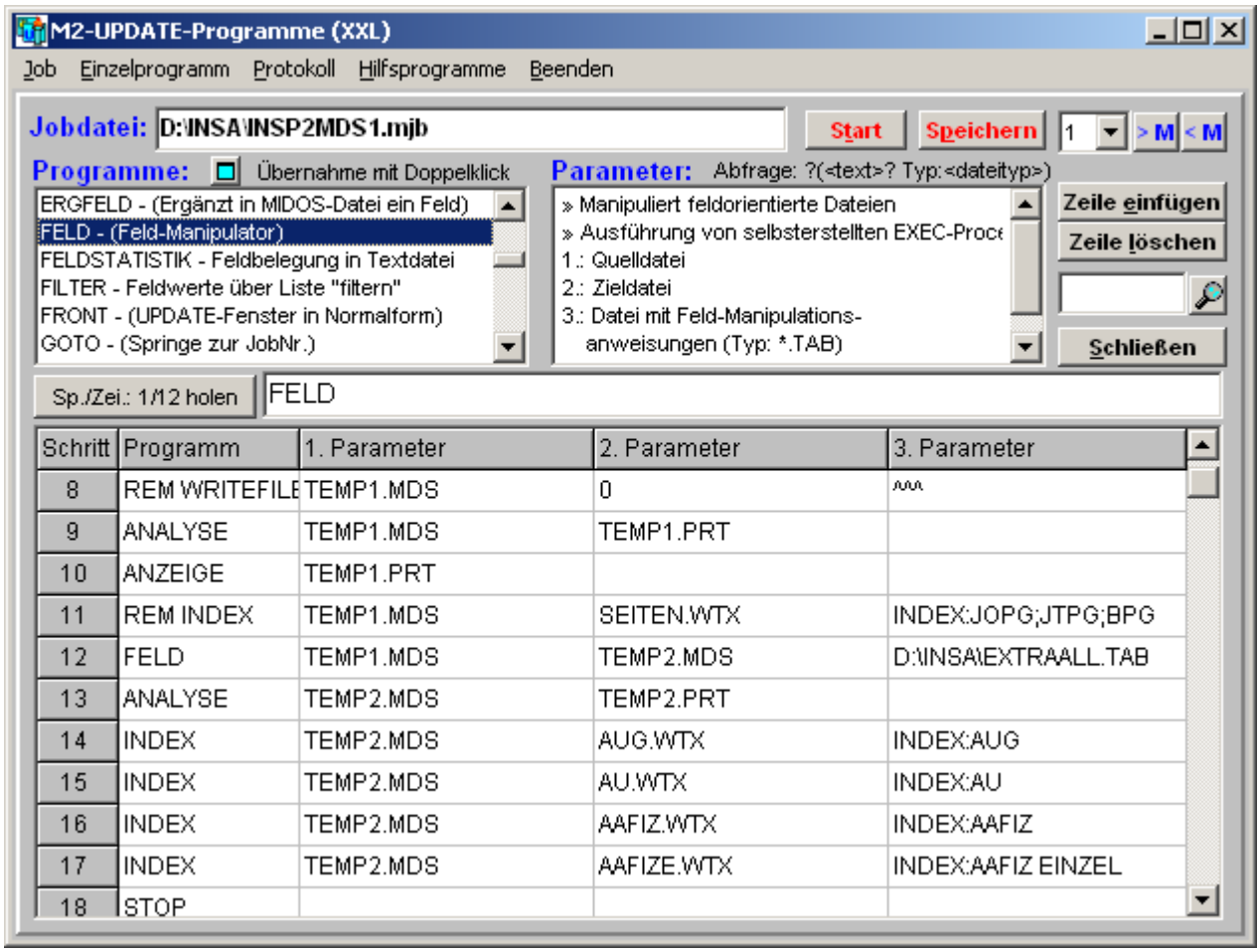
Dokumentation

Feldmanipulation in MIDOS-UPDATE (MU.EXE)

Die Script-Sprache zur Manipulation von Datenfeldern von Datenbankdateien der Datenbanksysteme MIDOS97, MIDOS2000 und MIDOS 6 für MS-WINDOS-Betriebssysteme

von Diplom-Ingenieur Thomas Stys, Hanau

Funktion in MIDOS-UPDATE (MU.EXE):



Inhalt

Werkzeug für Stapelverarbeitung/Testumgebung MWUPDATE.EXE

Mit MWUPDATE.EXE steht ein mächtiges Werkzeug zur Verfügung, mit dem unter den üblichen WINDOWS-Betriebssystemen typische Funktionen der Daten- und Informationsverarbeitung auf der Basis vieler vorgefertigter Module zusammengestellt und getestet werden können. Nach erfolgreichen Tets können die Funktionsfolgen in Jobs für die wiederholte Abarbeitung (Stapelverarbeitung, vergleichbar Batch-Prozessen unter WINDOWS) abgelegt werden.

Neben den Möglichkeiten, die es gibt, via Stapelverarbeitung Datenbanken so aufzubereiten, dass sie Online im Internet oder in internen Firmennetzen angeboten werden können, verfügt MWUPDATE.EXE über Funktionen, mit den beinahe beliebige Datenströme analysiert und in MIDOS-Datenbanken umgeform werden können.

Diese Funktionen sind

- die Zeichenkettenmanipulation
- die Feldmanipulation
- die Analysefunktion

Die folgenden Ausführungen beziehen sie auf die Feldmanipulation.

Die Feldmanipulation eine der komplexeren Funktionen, bei der es darum geht, eine Datei - vorliegend im MIDOS-Datenbankformat - in eine andere Datei umzuwandeln, wobei eine Script-Sprache für die Programmierung bestimmter steuernder Abarbeitungen benutzbar ist (ähnlich Turbo-Pascal).

Die wichtigsten Zeichenkettenoperationen wie:

- Auftrennen von Zeichenkette
- Zusammenziehen von Zeichenkette
- Sortieren von Feldwerten
- Dopplungsprüfung von Feldwerten
- Prüfen der Plausibilität
- Transliterieren
- Transkribieren
- Umcodieren
- Übersetzen (Austauschen) von Feldwerten
- Ergänzen von Feldwerten

- Länge ermitteln
- Subzeichenkettenposition bestimmen

sind - gesteuert von Bedingungen - möglich.

Im Bild 1 ist der Startbildschirm von MWUPDATE.EXE zu sehen. Die Bezeichnung M2-Update-Programme (XXL) soll zeigen, dass es möglich ist, mit diesem Modul Dateien beinahe beliebiger Größe zu verarbeiten (bis 16 Gigabyte, sofern Platz auf der Festplatte ist).

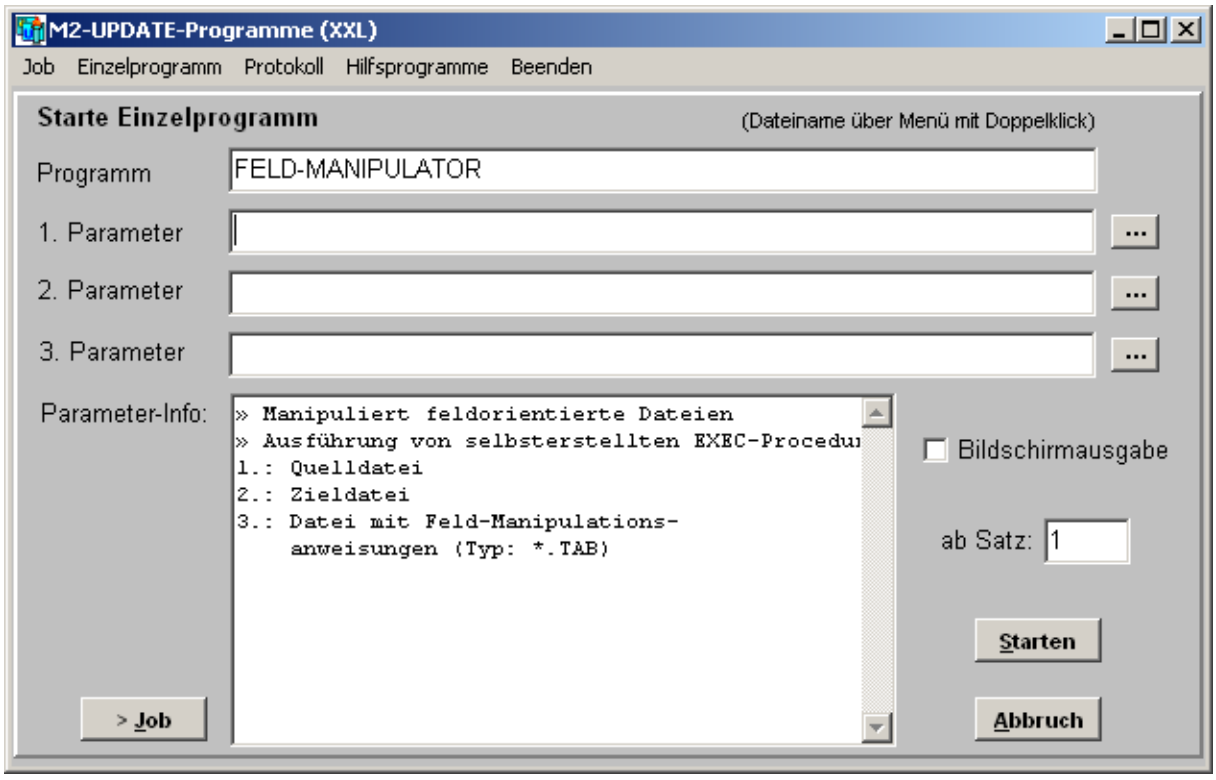


Bild 1: MWUPDATE-Grundmenü

Auf der Startseite sind die folgendnen Unterfunktionen zu sehen:

Job: Anlegen, Aufrufen, Speichern von Abarbeitungsfolgen im Jobs.

Einzelprogramm: Enthält einige häufig genutzte Funktionen zur direkten Auswahl und kann zwischen Einzelprogrammverarbeitung und Jobverarbeitung wechseln. Der Wechselzwischen Job und Einzelprogramm ist auch über den Umschalter">Job" (links unten) möglich.

Protokoll: Zeigt das Protokoll der Verarbeitung an. Je nach Funktion, kann es von Bedeutung sein, dieses Protokoll durchzusehen.

Hilfsprogramme: Enthält einige direkt auswählbare Funktionen, die das Hantieren erleichtern bzw. schnelle Analysen sowohl von Datenbanken als auch der programmtechnischen und softwareseitigen Ausstattung des Computers, auf dem UMWUPDAT betrieben wird.

Wird die Funktion "FELD-MANIPULATION" ausgewählt, ist im Parameter 1 die Quelldatei, im Parameter 2 die Zieldatei und im Parameter 3 die Datei mit den Feld-Manipulationsanweisungen (Dateityp *.TAB) anzugeben.

Wird "Bildschirmausgabe" ausgewählt, läuft die Feld-Manipulation im Testmodus auf dem Bildschirm ab. Die Durchlaufgeschwindigkeit der Bildschirmausgabe ist einstellbar.

Zwei Kommandos Showvars und ShowZF, die im weiteren erläutert werden, sind nur im Testmodus wirksam. Sie halten die Verarbeitung an und zeigen bestimmte Zustände an.

Außerdem kann die Satznummer eingestellt werden, ab der die Ausgabe auf den Bildschirm erfolgensoll.

Ein wichtiger Hinweis sei gegeben:

In den neuen Funktionskomplexen "MDS2XML" (Umsetzung von MIDOS-Datenbanken in XML-Format und "XML2MDS" (Umsetzen von XML-Datenströmen in MIDOS-Datenbanken) sind die Zeichenkettenmanipulation und die Feldmanipulation in einer erweiterten Form integriert.

Das Grundformat der Datenbankdateien

Mit dem Modul MWUPDATE.EXE des Datenbanksystem MIDOS 6 haben Sie einen leistungsfähigen, programmierbaren Transformationsbaustein für MIDOS-Datenbanken oder andere beliebige formatierte Textdateien zur Verfügung, dessen Ziel es ist, diese Dateien den eigenen Wünschen entsprechend in eine MIDOS-Datenbank zu integrieren.

Liegen der Datenbankgenerierung Fremddateien zu Grunde, die nicht im MIDOS-Datenbankformat vorliegen, sind diese zunächst mit dem MIDOS-Manipulator, Unterfunktion MAN oder REL (für relationale Strukturen) im Funktionskomplex UPDATE, in eine feldorientierte Struktur zu überführen.

Diese Struktur ist durch folgende zu erzeugende Merkmale gekennzeichnet:

<feldname>::=<bezeichner>

Länge: 1...15 (empfohlen 2 bis 8 Zeichen)

<feldnamenende>::=<spezialzeichen>

Standard: ":" (Doppelpunkt)

<feldwert>::=<zeichenkette>

Standard: erweiterter ANSI-Zeichensatz, max. Länge: 500000 Zeichen, größere Feldlängen auf Wunsch

<dokumentende>::=<spezialzeichen>

Standard: "&&&", max. Länge: 20 Zeichen

<feldende>::=<spezialzeichen>

Standard: CRLF (H'0D0A', #13#10)

Für die zuverlässige Markierung von Wiederholelementen in Mehrfachfeldern (auch "multiple Felder" genannt), ist es möglich, einen Feldaussagentrenner (auch "Wortgruppentrenner" oder "Feldwertetrenner" genannt), zu erzeugen.

<feldaussagentrenner>::=<spezialzeichen>

Standard: | (H'#124)

Allgemein gilt folgendes einfache Schema:

```
<feldname><feldnamenende><feldwert>[<austrz><feldwert>...]<feldende>
<doocumentende>
```

Der kürzest mögliche Datensatz besteht aus einem einstelligen Feldnamen und einem einstelligen Wert gefolgt vom Datensatzendezeichen:

```
A:1
&&&
```

Eine etwas weiter gefasstes Beispiel zeigt folgendes Schema:

```
ST:Feldmanipulation in M2-UPDATE (XXL). Die Script-Sprache zur
Manipulation von Datenfeldern in Datenbankdateien der
Datenbanksysteme MIDOS97, MIDOS2000 und MIDOS 6
GT:Systemunterlagen-Dokumentation
AB:Hier folgt ein kurzes Beispiel.
DE:Beispiel|Test|MIDOS 6|Transfer|Feld|Beschreibung|
Kommandosprache
DT:19980919
TD:19.09.1998
RZ:100.10
WE:DEM
OT:Hanau|Berlin
AU:Stys, T.|Kunkel, P.
&&&
```

Aus praktischen Gründen wurden in den letzten Jahren Erweiterungen an den Möglichkeiten der Feldverwendung im Datenbanksystem ergänzt, für deren Nutzung auf andere Datenübernahmeprogramme zurückgegriffen werden kann, wie z.B.

- XML2MDS, liest XML-Dateien, analysiert sie und protokolliert die Analyseergebisse und stellt Mittel bereit, um die XML-Dateien mit Manipulationen in ein gültiges MIDOS-Datenbankdateiformat zu überführen.
- HTML-Tabellen einlesen
- Relationale Dateitabellen einlesen und zusammenführen (CSV-Dateien, oder "delimited with"-Dateien (z.B. Tabulator formatierte Dateien, oder Dateien fester Satzstruktur ohne Trennzeichen (z.B. mit Leerzeichen formatierte Tabellen.
- MDS2XML, erzeugt mittels Steuerschema aus MIDOS-Dateien XML-Dateien mit geprüfter Syntax und geprüftem Zeichensatz.

Allgemeine Syntax der Feldmanipulation

Die Feldmanipulation kann auf jede Datei angewendet werden, die in einem gültigen Datenbankformat von MIDOS2.1, MIDOS3.0, MIDOS97, MIDOS2000, MIDOS 6 vorliegt.

Die mit '*' gekennzeichneten Kommandos sind nur auf das strenger definierte MIDOS 6-Format anwendbar. Entscheident dafür sind die

Parameter:

<aussagentrenner> = |

<dokumentende> = &&&

<feldende> = CRLF

Bei Unstimmigkeiten bezüglich des vorliegenden Formates konsultieren Sie den Entwickler oder bearbeiten Sie Ihre Dateien mit der Funktion "Analyse" im Menu "Jobs" oder in MWUPDATE. Prüfen Sie die Dateien an den Stellen, die Ihnen

- als Syntaxfehler oder
- als extreme Werte für die Feldlänge oder
- als verdächtige Zeichen oder
- extreme Feldaussagenlänge (z.B. 0, variabel statt fest) oder
- extreme Fekdaussagenanzahl (z.b. 0, oder mehr als 1, wenn nur 1 zulässig)

nachgewiesen werden. Dazu benutzen Sie den Dateilister, den Sie mit der F2-Taste im Job (von MWUPDATE) oder im Dateimanager mit F2-Taste (auf markierter Datei) aufrufen können. Dort gibt es die Funktion "Gehe zur Dateiposition". Die Dateiposition der Extremwerte können Sie im Protokoll der Analyse sehen.

Extreme Werte sind:

Dokumentlänge:

Kleiner 7 oder gößer als die Summe aller Feldlängen in der Datei MISCHABS

Ab 2005 kann die Dokumentlänge 1 Megabyte betragen. Auf Wunsch können Sie erweiterte Module bis 4 Megabyte Datensatzlänge beziehen.

Feldlänge:

gleich 0 oder größer als die Feldlänge in der Datei MISCHABS

oder ein Datumsfeld hat eine variable Länge

oder ein Zahlenfeld hat keine feste Länge

oder ein Feld mit Werten fester Länge ist variabel (Sprachencodes, Ländercodes mit eigentlich fester Länge sind variabel lang usw.)

Aussagenlänge:

gleich 0 oder größer als die definierte Aussagenlänge in der Datei MISCHABS

oder größer als von Ihnen erwartet

oder kleiner als von Ihnen erwartet

Aussagenanzahl:

größer als die definierte Aussagenanzahl in der Datei MISCHABS

oder kleiner oder größer als von Ihnen erwartet

Feldanzahl:

Größer als die Dokumentanzahl (dann haben ein oder mehrere Dokumente kein richtiges Ende) oder in Dokumenten kommt das gleiche Feld mehrmals vor.

Hinweis: Ein MIDOS-Datensatz enthält jeden Feldnamen nur einmal. Hat ein Feld mehrere Werte, so ist es als Mehrfachfeld zu definieren und die Werte sind mit dem Aussagentrenner zu separieren.

Die Fehlerabschätzung ist abhängig von der Art der Datenbank. Mit einiger Berufserfahrung finden Sie jedoch schnell heraus, was sein kann und was nicht plausibel ist (oder mehrdeutig und damit unbrauchbar).

Beispiele:

- Ein Personenname hat die Länge 2: nicht plausibel, wenn Nachname, Vorname vorgeschrieben ist.
- Ein Firmenname hat die Länge 603: nicht plausibel, der Wert muss geprüft werden.
- Ein Sprachencode hat die Länge 1: nicht plausibel, es müssen 2 oder 3 stellen sein.
- Ein Feld ANA hat die Häufigkeit 3222 aber es sind nur 3000 Datensätze in der Datenbank: nicht plausibel. Es gibt Dokumente in denen das Feld ANA mehrfach enthalten ist.
- Ein Feld hat die Länge 0: nicht plausibel, Felder der Länge Null können zwar entstehen bei der Feldmanipulation, Sie sollten dies aber vermeiden, denn es ist ein Unterschied, ob ein Feld nicht vorhanden ist, weil dessen Inhalt nicht zu ermitteln ist, oder ob es versehentlich leer erzeugt wurde. Leere Felder haben in einigen Prozessen der Datenbankarbeit Folgen, die bedacht werden müssen (Statistik, Ausgabe, Anzeige, Formulare, Tabellenaufbereitung)

Die Rolle der Datei MISCHABS (Eingabebeschreibung) wird in der Dokumentation zu MIDOS 6 erläutert.

Feldmanipulationstabelle

Die Kommandos der Feldmanipulation werden in einer Transformationstabelle definiert und in einer Datei abgelegt, die die Dateiendung .TAB haben sollte.

Die Größe der Datei darf z.Zt. 500 KByte nicht überschreiten. Sie muß mit dem "Editor für kleine Dateien" bearbeitbar sein, damit die Vorteile der integrierten Syntaxprüfung zur Verfügung stehen. Für die Feldmanipulation ist die Transformationstabelle nach dem Format. Größere Feldmanipulationen bearbeitet man, indem man in einer ersten Verarbeitung einen Teil der Manipulationen ausführt und den weiteren Teil der Manipulationen mit einer weiteren Tabelle bearbeitet.

Grundform:

<quellfeldname1><leerzeichen><zielfeldnameY>

<quellfeldname2><leerzeichen><zielfeldnameY>

...

<quellfeldnameN><leerzeichen><zielfeldnameZ>

zu erfassen.

<quellfeldname> steht für den Namen des Feldes in der **<quelldatei>**.

<zielfeldname> steht für den Namen des Feldes in der **<zieldatei>**.

<quellfeldname> bzw. **<zielfeldname>** können max 15 Zeichen lang sein (Der Doppelpunkt muss spätestens auf Position 16 stehen.

Besonderheiten der Feldmanipulationstabelle

1. Als <quellfeldname> kann das Schlüsselwort "**EXEC**" gefolgt von einer max. 4 stelligen Zahl (1..9999) stehen. Damit sind Prozeduren aufgerufen, die im Transformationsprogramm nach der Feldzuordnung definiert werden müssen. Die Angabe EXECn wird als EXEC-Prozeduraufruf bezeichnet. Hinter einem EXEC-Prozeduraufruf kann ein <zielfeldname> stehen. Dann wird das Ergebnis des EXEC-Prozedur diesem Feldnamen zugeordnet. Steht hinter einem EXEC-Prozeduraufruf kein Feldname, so muß in der EXEC-Prozedur ganz besonders auf die Erzeugung der richtigen Syntax des Feldes bzw. Dokumentes geachtet werden.
   Hinweis: Es sind Fälle denkbar, bei denen kein Feldname erzeugt werden soll. Sie können beinahe beliebige Strukturen erzeugen, u.a. auch HTML- oder XML- oder SGML-Dateien.

2. Der <zielfeldname> kann entfallen, dann ist <quellfeldname> gleich <zielfeldname>. Eine inhaltliche Änderung der Feldwerte erfolgt nicht (1:1-

Zuweisung). Damit soll der Datenaustausch zwischen sehr ähnlich definierten Anwendungen möglichst einfach werden.

3. Der Inhalt von <quellfeldname>, die weder in der Transfertabelle noch in einer EXEC-Prozedure genannt bzw. verarbeitet werden, werden nicht in die <zieldatei> ausgegeben.

4. Die Transformationstabelle, speziell der Teil der Feldzuweisungen, kann mit dem Wortsymbol NWENDE {nwende} abgeschlossen werden. Damit wird das als Standardwert definierte Dokumentendezeichen (&&&) automatisch nach jedem Dokument gesetzt. Es sind Fälle denkbar, in denen eine andere Endezeichenkette gewünscht wird. Dann ist NWENDE wegzulassen und das Dokumentende ist via EXEC-Prozedur zu setzen.
   Hinweis: Sie müssen in dem Fall ein eindeutiges Feld kennen, das in jedem Dokument enthalten ist und nach dessen Ausgabe Sie die Endezeichenkette setzen können. Es kann auch ein Feld mit dieser Endezeichenkette sein.

5. Ist ein in der Transformationstabelle genanntes <quellfeld> leer, wird es auch im Zielformat leer erzeugt. Soll der Transfer anders ablaufen, ist eine EXEC-Prozedur zu schreiben, die die Übergabe leerer Felder in die Ausgabe unterdrückt. (siehe Bedingte Anweisung if...)

6. Soll im Rahmen der Transformation ein MIDOS-Datenbankformat entstehen, so muß der Anwender selbst für die richtige Syntax des entstehenden Formates sorgen, also Feldnamen, Feldnamenende, Feldende und Aussagentrenner sowie Dokumentende erzeugen.

7. Hinter jeder Zeile einer EXEC-Prozedur, die mit dem Zwischenfeld ZF arbeitet, kann ein Kommentar stehen.

8. Außer dem Zwischenfeld gibt es noch ein MEMO-Feld. Zwischenfeld und MEMO-Feld. Die Felder brauchen nicht definiert werden.

Eine Transformationsdatei besteht aus den Teilen:

1. einer Transformationstabelle mit Quellfeld-/Zielfeldzuweisung

2. einem EXEC-Prozedurteil mit den EXEC-Prozeduren.

Beispiel:

```
EN EXTNR -> Quellfeld = Zielfeld mit Wechsel des Feldnamens
VF VERF
ST TITEL
EK -> Quellfeld = Zielfeld
DT Feldname bleibt gleich
EXEC1 DE2 -> Zielfeldinhalt wird in EXEC1
EXEC2 STANDORT erzeugt und in DE2 gespeichet
EXEC3 -> Zielfeld wird in EXEC3 erzeugt
```

```
NWENDE -> Dokumentende automatisch übertragen
 -> Leerzeile mit 1 Leerzeichen
EXEC1:
CA(DE,1,100,|,='*') -> erste EXEC-Prozedur
CC(^,&) -> Konstante wird dzugewiesen
 -> Leerzeile mit 1 Leerzeichen
EXEC2: ... -> nächste EXEC-Prozedur Feldname im Feldteil
zugewiesen
 -> Leerzeile mit 1 Leerzeichen
EXEC3:
CC(X1:) -> Feldname erzeugt muss werden.
CS(RF,1,25)
CC(^) -> Das Feldende muß erzeugt werden
```

Die Programmstruktur und die Scripte können außerordentlich vielfältig sein. Sie sollten sich einen Vorrat an Standardprozeduren schaffen.

Syntax der EXEC-Kommandos

EXEC-Kommandos sind im Anschluß an die Transformationstabelle nach einer Leerzeile zu definieren. Eine Leerzeile im Sinne dieser Syntax besteht aus "CRLF Leerzeichen CRLF".

Es kann nur ein Kommando pro Zeile stehen. Hinter einem ZF-Kommando kann ein Kommentar stehen.

EXEC-Prozeduren sind mit EXEC<nr> einzuleiten und mit einer Leerzeile abzuschließen.

Fehlt eine EXEC-Prozedur wird beim Start des Transfers ein Syntaxfehler gemeldet. Die Syntaxprüfung ist an die Verwendung des integrierten Editors in der Grundfunktion MWUPDATE bzw. an die Benutzung der Editors für kleine Dateien unter MIDOS gebunden.

Wichtig: In EXEC-Prozeduren ist der Anwender für die Herstellung einer gültigen MIDOS-Datenbankstruktur zuständig, sofern er MIDOS-Format erzeugen will oder muss. Das betrifft vor allem

1. die Bereitstellung eines CRLF (#13#10 oder ^) am Ende eines Feldes,
2. die Bereitstellung gültiger Dokumentendezeichen (&&&),
3. das Setzen gültiger Aussagentrennzeichen (|) und
4. die Verwendung gültiger Feldnamen (1 bis 15 Zeichen plus Doppelpunkt, Ziffern und Buchstaben, möglichst keine Sonderzeichen (außer „#“, „.“, „-„).

Hinweis: Soll der Transfer in ein anders Datenbanksystem (z.B. ACCESS, ORACLE) erfolgen, so muß die Syntax dieses Systems bekannt sein und ebenfalls eigenständig erzeugt werden. Dazu ggf. REL-Konverter benutzten.

Soll das Zielsystem ein relationales Datenbanksystem sein, so kann die Unterfunktion "Austauschformat" in MWUPDATE.EXE hilfreich sein. Die Übergabe von Dokumenten an "WORD Für WINDOWS" oder ähnliche Textverarbeitungssystem erfolgt dagegen zweckmäßiger über die Funktion "Recherche", Unterfunktion "Ausgabe im RTF-Format" oder die gleichnamige Funktion in MWUPDATE.EXE, in der ggf. eine Ausgabebeschreibung zur Gestaltung der Übergabeform zu benutzen ist.

Beschreibung der Kommandos

Im folgenden gilt für die Schreibweise:

{ } steht für andere Schreibweise

[ ] steht für wahlweise Angabe

<> markiert Variable

Reservierte Wortsymbole sind in Grossbuchstaben geschrieben, obwohl Programm sowohl in Groß- als auch Kleinbuchstaben zulässig sind.

Es existieren 3 Arten der Wirkung von Kommandos, einfache Kommandos, Zwischenfeldkommandos und MEMO-Feld-Kommandos.

Einfache Kommandos lesen aus einer Datei den Inhalt eines Feldes und geben ihn ggf. gebunden an Bedingungen dierkt in die Zieldatei aus. Statt eines feldes können auch Konstanten erzeugt und ausgegeben werden. Einfache Kommandos benutzt man für die schnelle Konvertierung (Umorganisation) von eigenen Datenbanken oder Fremddatenbanken, deren Kompatiblität einigermaßen gegeben ist.

Zwischenfeldkommandos lesen aus einer Datei den Inhalt eines Feldes in ein Zwischenfeld ein, alle Bedingungen und weiteren Kommandos können nun über das Zwischenfeld ausgeführt werden, bis es gezielt in eine Zieldatei ausgegeben wird. Zwischenfeldkommandso lassen komplexere Programmaufbauten zu, mit denen auch weitgehend inkompatible Datenströme konvertiert und transformiert werden können, nachdem es gelungen ist, die Datensätze der Quelldatei in ein MIDOS-Format umzuformen. Dazu muss ggf. vorher ein Zeichenkettenmanipulator ein gültiges MIDOS-Format erzeugen.

MEMO-Feld-Kommandos lesen das Zwischenfeld und können in das Zwischenfeld schreiben. Es wird also ein Datenpuffer bereitgestellt, der auch Datensatz übergreifend wirken kann.

Einfache Kommandos

Kommentare in der Transformationstabelle

REM <kommentar>

Kommentare werden zeilenweise mit REM eingeleitet.

REM muss auf der ersten Position der Zeile stehen.

Kopieren eines Feldes

COPYALL(<feldname>,[<bedingung>])

{COPY(...)} {CO(...)}

- kopiert das gesamte Feld vom 1. Zeichen nach dem **<feldanfangszeichen>** (im allgemeinen ":") bis vor **<feldende>** (im allgemeinen CRLF (X'0D0A', #13#10, ganz früher ";")

- **<bedingung>** s.u.

Kopieren einer Konstanten

COPYCONST(<zeichenkette>,[&])

{CC(...)}

- kopiert eine Zeichenkette **<zeichenkette>**

- Die <**bedingung &>** bewirkt, dass die Konstante nur kopiert wird, wenn bereits ein Wert (**<feld>** oder eine **<zeichenkette>**) in den Ausgabebereich kopiert wurde.

- Die maximale Konstantenlänge beträgt 255 Zeichen. Längere Zeichenketten müssen aus mehreren Teilzeichenketten zusammengesetzt werden.

- Konstanten, die Zeichen aus der Syntax der Kommandos enthalten, sind in Apostroph einzuschließen.

- Die Zeichen können auch in ihrem ASCII-Wert angegeben werden.

CC(' ') -> ein Leerzeichen

CC(#32) -> auch ein Leerzeichen

cc(#9) -> ein Tabulatorzeichen

Hinweis auf eine wichtige Besonderheit:

cc(#13#10) -> ein Zeilenvorschub mit Wagenrücklauf (CRLF)

cc(^) -> Das Zeichen ^ steht innerhalb der Kommandos für #13#10

- Unter MIDOS sind die Umlaute speziell zu behandeln, wenn die Grundeinstellung **ASCII** für den Zeichensatz der Datenbanken gewählt wurde. Ein Umlaut in einer Konstanten, einer Bedingung oder einer Translate-Zeile ist in diesem Fall mit seinem

ASCII-Zeichencode darzustellen und wird in diesem Code abgefragt und geschrieben.

Sind ASCII-Umlaute im Dokumente enthalten, müssen sie mit dem ASCII-Zeichen dezimal abgefragt werden. Das gilt für jedes ASCII-Zeichen größer als dezimal 127.

Hinweis: Da das sehr fehleranfällig ist und wenig zukunftskompatibel, wird ab 2009 allen MIDOS-Anwendern empfohlen, Datenbanken ausschließlich im ANSI-Zeichensatz zu betreiben (zu erfassen und zu archvieren).

Die Entwickler empfehlen, ältere Datenbestände möglichst kurzfristig in den ANSI-Zeichensatz zu überführen und dabei auch fehlerhafte Zeichensetzungen zu korrigieren. Mit der Funktion "ASCII zu ANSI" des MWUPDATE-Paketes können Sie diese Umformung ausführen, wenn der Datenbestand ansonsten frei von Kodierungsfehlern ist.

Andernfalls müssen Sie kontextbezogene Zeichenumsetzungen mit dem Zeichenkettenmanipulator von MWUPDATE in einem vorgelegerten Manipulationsprozess ausführen. In der ANSI-Datei muss der Codebereich #129 bis #159, der von WINDOWS mit ANSI-fremden Zeichen belegt wurde, frei bleiben. Andernfalls ist die Codekompatibilität mit später wichtigen Codesystemen, wie UTF8, UNICODE nicht zu gewährleisten.

```
if exist(NAME,#154) goto xy
```

Bedeutung: Wenn im Feld NAME ein ASCII-Ü existiert, verzweige zur Marke xy, die später im Programm definiert sein muss.

```
if exist(NAME,Ü) goto xy
```

Bedeutung: Abfrage eines ANSI-Ü, mit der gelichen Reaktion.

ASCII	Glyph	ganz früher	ANSI	Zeichen in HTML, XML, SGML	auch möglich in XML
#142	Ä	Ae	#196	Ä	Ä
#153	Ö	Oe	#214	Ö	Ö
#154	Ü	Ue	#220	Ü	Ü
#225	ß	ss	#223	ß	ß
#132	ä	ae	#228	ä	ä
#148	ö	oe	#246	ö	ö
#129	ü	ue	#252	ü	&#252

Im Zusammenhang mit Zeiichensätzen wie UTF-8, UTF-16, UTD-32 oder UNICODE u.ä. können benannte Zeichenentitäten von Bedeutung sein, die jedoch nur bei der Verwendung entsprechender WEB-Browser sinnvolle Anzeigen erlauben.

Benannte Zeichenentitäten sind im Sinne von MWUPDATE Zeichenketten, auf die keine automatische Reaktion in irgendeiner Weise ausgeführt wird. Alle

Bearbeitungen müssen händisch erfolgen, da Sinn und Zweck der Zeichenentitäten dem Programm nicht bekannt sein können.

Beachten Sie, dass es in den Codetabellen Differenzen gibt, die nur unter Beachtung der Bedeutung des Zeichens im Kontext umgeformt werden können.

CC('#142nderungsdatum')

Bedeutung: Erzeugt ASCII-Code für Ä (Sie sollten das nicht tun!)

cc(Ä)

Bedeutung: Erzeugt die benannte Zeichenentität Ä die in HTML-Anwendungen für ein Ä steht.

CC(Ä)

Bedeutung: Erzeugt ein Ä im ANSI-Code.

Kopieren einer Teilzeichenkette

COPYSUBSTRING(<feldname>,<von>,<bis>,[<bedingung>],[*])

{CS(...)}

- kopiert eine Teilzeichenkette aus dem Feld **<feldname>** von der Positions **<von>** bis zur Position **<bis>**, wenn die Angaben Zahlen sind (1...32756)

- kopiert eine Teilzeichenkette aus einem Feld von der Positions **<vonzeichenkette>** bis zur Position **<biszeichenkette>**, wenn die von/bis-Angaben Zeichenketten sind und die Position **<biszeichenkette>** ungleich Null und größer als **<vonzeichenkette>** ist.

- <bedingung> siehe unten

CS(EK,1,20)

-> von Position 1 bis Position 20 wird eine Teilzeichenkette entnommen (ein substring)

CS(EK,' Berlin ','(')

-> vom 1. Auftreten des Wortes Berlin, bis zur 1. runden Klammer. Die Funktion liefert nur dann ein richtiges Ergebnis, wenn die "(" nach dem Wort "Berlin" steht, andernfalls ist das Ergebnis leer.

Kopieren von Aussagen

COPYAUSSAGE(<feldname>,<von>,<bis>,<trennzeichen>,[<bedingung>],[*])

{CA(...)}

- kopiert aus dem Feld **<feldname>** die Aussagen **<von>** einer Aussagennummer **<bis>** zu einer Aussagennummer

- **<trennzeichen>** ist der Aussagentrenner, der in der Zieldatei als Trenner verwendet wird (Normal: |).

COPYAUSSAGE(DE,3,3,|)

Bedeutung: nur die dritte Aussage in die Zieldatei kopieren

COPYAUSSAGE(DE,2,4,|)

Bedeutung: die zweite Aussage bis zur vierten Aussage in die Zieldatei kopieren

COPYAUSSAGE(DE,1,32564,' | ')

Bedeutung: (praktisch) alle Aussage in die Zieldatei kopieren, der Trenner wird mit Leerzeichen versehen.

COPYAUSSAGE(DE,4,20,#94)

Bedeutung: Aussage 4 bis 20 in die Zieldatei kopieren und das Zeichen #94 (^) als Trenner setzen.

COPYAUSSAGE(DE,4,20,#13#10)

COPYAUSSAGE(DE,4,20,^)

Bedeutung: Aussage 4 bis 20 in die Zieldatei kopieren und das Zeichen CRLF (Zeilenwechsel) als Trenner setzen.

Bedingungen für Kopieranweisungen

Als Bedingung ist eine max. 40 Zeichen lange Zeichenkette <zk> mit Operator einzugeben. Die Bedingung steuert die Ausführung der Kopieranweisungen:

=<zk>

- nur wenn <zk> in der Kopier-Zeichenkette enthalten ist

<><zk>

- nur wenn <zk> nicht in der Kopier-Zeichenkette enthalten ist

<<zk>

- nur wenn die Kopier-Zeichenkette vor <zk> im Feld steht. 1)

><zk>

- nur wenn die Kopier-Zeichenkette hinter <zk> im Feld steht 1)

VON<zk> -

- nur wenn <zk> in der Kopier-Zeichenkette enthalten ist, kopiert wird ab 1. Zeichen nach <zk> bis Aussagenende. 2)

BIS<zk>

- nur wenn <zk> in der Kopier-Zeichenkette enthalten ist, kopiert wird ab 1. Aussagenzeichen bis ein Zeichen vor <zk> 2)

1) - nur für copysubstring

2) - nur für copyaussage

Die Parameter der Kommandos sind durch Kommata zu trennen.

Ein <zk>-Parameter darf max. 255 Zeichen lang sein und ist in Hochkomma einzuschließen. Steuerzeichen sind als ASCII-Code mit "#" oder "@" einzugeben.

```
#94 = ^ (als Zeichen)
#39 = Hochkomma ( ' )
#35 = Doppelkreuz ( # )
#13#10 = ^ = (CRLF) = Zeilenwechsel
#64 = kommerzielles "A" oder at (@)
```

Übersetzen von Aussagen

TRANSLATE(feldname,von,bis,trennzeichen)

[FILE:<dateiname>]

{TR(...)}

- transformiert Aussageninhalte des Feldes **<feldname>** gemäß einer Translate-Tabelle, die nach dieser Anweisung oder in einer externen Textdatei stehen muß.

- **<von>, <bis>** definieren, von der wievielten Feldaussage bis zu welcher Feldaussage des Quellfeldes der **TRANSLATE**-Befehl ausgeführt werden soll.

- Trennzeichen definiert eine **<zeichenkette>** die im **<zielfeld>** zwischen die übersetzten Feldaussagen gesetzt werden soll. **<zeichenkette>** kann bis zu 3 Zeichen lang sein und muß ggf. in Apostroph eingeschlossen oder durch dezimale ANSI-Werte dargestellt werden, z.B. #32#124#32 oder ' | '.

- Die Tabelle ist mit einer (echten) Leerzeile (**CRLF+Leertaste+CRLF**) zu beenden.

- Liegt die Translate-Tabelle in einer externen Datei vor, muß diese mit **FILE:<dateiname>** unmittelbar in der Folgezeile von **TRANSLATE** aufgerufen werden. **<dateiname>** umfaßt entweder die komplette Laufwerks- und Verzeichnisangabe oder nur den Dateinamen. In letzteren Fall muß sich die Datei im Verzeichnis befinden, in dem auch die Transferdatei steht. Nach der Angaben von **FILE** muß in jedem Fall eine Leerzeile stehen, bevor ein weiteres Kommando folgen kann (z.B. üblicherweise **CC(^)**, wodurch der Feldabschluss erzeugt wird).

EXEC-Prozedur

Eine EXEC-Prozedur ist ein Programm, das im allgemeinen eine genau definierte Aufgabe der Datentransformation löst. Die EXEC-Prozedur für eine Translate-Aufgabe könnte z.B. folgende Struktur haben.

```
EXEC12:
REM ----------------- Translate-Vorbereitung
REM
IF not exist(TRANSLATEQUELLE) goto ende
```

```
CC(ZIELFELDNAME:)
REM
REM ----------------- Translate-Ausführung
REM
TRANSLATE(TRANSLATEQUELLFELD,1,6,|)
FILE:TRANSLATETABELLE.TXT


CC(^)
:ende
```

Tabellenstruktur von Translatetabellen

Die linke Seite der Translate-Tabelle enthält die **<quellfeldaussage>**, die rechte Seite die **<zielfeldaussage>**.

Die **<quellfeldaussage>** und **<zielfeldaussage>** in der Tabelle können mit Komma (ASCII #44) oder ^ (ASCII #94), nur wenn die Tabelle in einer externen Datei gespeichert ist) getrennt werden.

- Enthält eine Quell- oder Zielaussage ein zu erhaltendes Komma oder ein "^", dann sind diese Zeichen durch ihren ASCII-Wert auszudrücken.

- Werte von **<quellfeldaussagen>**, die nicht in der Tabelle enthalten sind, entfallen (vergl. aber **TRANSLATEALL**).

- Die Länge einer Feldaussage darf **255** Zeichen nicht überschreiten (ab 2000).

- Eine in der Tabelle enthaltene Leerzeile beendet die Tabelle.

- Ein leerer Wert auf der rechten Tabellenseite führt zur **Löschung** (leere **<zielfeldaussage>**).

- ^ als Trenner wird nur in externen Translatedateien unterstützt.

- innerhalb eines Transferprogrammes können mehrere Quellfelder mit verschiedenen Translatetabellen und Tanslatearten in verschieden Zielfelder oder bei geschickter Programmformulierung auch in ein einziges Zielfeld überführt werden.

Translate-Datei (externe Datei)

```
AAA^AAA Allgemeines
BBB^BBB Philosophisches, Beispiel
XXX^CCC Charakteristisches|AAA Allgemein #252bliches
YYY$ZZZ^ZZZ Sonstiges
ABC^
```

Bedeutung: Die Feldaussage der linken Seite wird durch die auf der rechten Seite der Tabelle ersetzt. 3. Zeile enthält die Besonderheit: Aus einer Feldaussage des Quellfeldes werden zwei Feldaussagen im Zielfeld.

Der Feldwert ABC wird nicht übersetzt, egal ob **TRANSLATE** oder **TRANSLATEALL** benutzt werden.

***TRANSLATEALL(<feldname>,<von>,<bis>,<trennzeichen>)**

[FILE:<dateiname>]

{TRALL(...)}

- wie TRANSLATE, jedoch werden nicht in der Tabelle enthaltene **<quellfeldaussagen>** unverändert als **<zielfeldaussage>** übertragen.

- Aussagenlänge max. 255 Zeichen

- die Belegung des Speichers wird überwacht, bei zu großen Tabellen (mehr als 50000 Zeilen und größer 1 Megabyte) wird eine Fehlernachricht ausgegeben und die Verarbeitung startet nicht.

Beispiel für TRANSLATE mit interner Tabelle

```
TRANSLATE(DE,1,10,|)
PC,Rechner
Computer,Rechner
Personalcomputer,Rechner
Laube,
Drucker,Drucker (Gerät)
Druckerzeugnis,Druck-Erzeugnis


cc(^)
```

- Bedeutung: Der Begriff links vom Trennzeichen (hier Komma) wird durch den Begriff rechts vom Komma ersetzt. Ist ein Komma Bestandteil des Feldwertes, muss es mit #44 umschrieben werden (sowohl als Quellwert als auch als Zielwert.

Es handelt sich immer um Feldaussagen, die mit **TRANSLATE** übersetzt werden.

In der oben aufgeführten Tabelle hat der Feldwert "Laube" keinen Zielwert und wird daher gelöscht. Außerdem werden alle Begriffe in dem betreffenden Feld nicht übersetzt, die in der Tabelle nicht links vom Trennzeichen genannt sind.

Beispiel

```
EXEC1234
if not exist(DE) goto ende
cc(DEKORR:)
TRANSLATEALL(DE,1,50,' | ')
FILE: KONVERTIERE-DESKRIPTOR.TXT


cc(^)
```

```
exit
:ende

EXEC1235
if not exist(DE) goto ende
cc(DEKORR:)
TRANSLATEALL(KLASSE,1,10,'#94')
FILE:E:\M2000\TEMA\KLASSKONK\KONVERTIERE-KLASSIFIKATION.TXT

cc(^)
exit
:ende
```

Bedeutung: In der EXEC-Prozedur wird zuerst abgefragt, ob das Feld **DE** vorhanden ist. Bei NEIN wird zur Marke `:ende` verzweigt. Dann wird der Ausgabebereich mit dem Wert `DEKORR`: gefüllt. Anschießend arbeitet sich **TRANSLATEALL** mit der Tabelle **KONVERTIERE-DESKRIPTOR.TXT** durch die ersten 50 Feldwerte des Quellfeldes **DE**, wobei es die Übersetzung ausführt, wenn der Quellwert im Quellfeld gefunden wird oder den Wert des Quellfeldes überträgt, wenn in der Tabelle keine Übersetzung gefunden wird.
Mit **FILE** wird die Tabelle als externen Datei gekennzeichnet. Ohne Pfadangabe wird sie im aktuellen Verzeichnis erwartet, in dem auch die Manipulationstabelle steht.
Dem Tabellenaufruf folgt eine (echte) Leerzeile und der Feldabschluss mit **cc(^) (CRLF).**
Das **exit** ist syntaktisch nicht notwendig, zeigt aber in den Beispielen, wo die Verarbeitung nach dem Translate fortgesetzt würde.
Wird statt **cc(^)** beispielsweise ein Aussagentrenner gesetzt mit **cc(' | ')**, kann eine weiteres Quellfeld übersetzt und an das gerade erzeugte Feld angefügt werden.

Beispiel:

Aufgabenstellung: Bei einer Thesauruspflege wurde das FREETERM-Feld ausgewertet und einige dieser FREETERM-Werte wurden als Deskriptoren in den Thesaurus aufgenommen. Nun soll die Indexierung im Feld DESKRIPTOR so geändert werden, das die FREETERM-Werte, die nun Deskriptoren sind, aus dem Feld FREETERM in das Deskriptorfeld übertragen werden. Die Datei **FREETERM2DESKRIPTOR.TXT** *enthält die Tabelle der Struktur* ***Freeterm^Deskriptor****, nur für die betroffenen Werte.*

Anschließend werden die umgespeicherten Begriffe im Feld FREETERM gelöscht.

```
EXEC1234
REM Speichere FREETERMS gezielt als DESKRIPTOR
if not exist(DESKRIPTOR) goto ende1
cc(DEKORR:)
TRANSLATE(DE,1,50,' | ')
FILE:KONVERTIERE-DESKRIPTOR.TXT

cc(|)
```

```
:ende1
if not exist(FREETERM) goto ende2
TRANSLATE(FREETERM,1,50,' | ')
FILE: FREETERM2DESKRIPTOR.TXT

:ende2
cc(^)
EXEC1236
REM Lösche Deskriptoren aus dem FREETERM-Feld
if not exist(FREETERM) goto ende
cc(FREETERM:)
TRANSLATEALL(FREETERM,1,50,' | ')
FILE:LOESCHEDESKRIPTOR.TXT

cc(^)
:ende
```

*Die Tabelle LOESCHEDESKRIPTOR enthält für die zu löschenden [neuen] Deskriptoren eine Löschanweisung der Form **Freeterm^**.*

Durch TRANSLATEALL werden alle nicht genannten FREETERM in das Zielfeld übertragen.

Formatieren von Feldern

FORMAT(<feldname>,<links/rechts>,<anzahl>,<füllzeichen>)

{FO(...)}

- formatiert das ganze Feld <feldname> links- oder rechtsbündig

- <links>/<rechts> gibt die Formatierungsrichtung an

- **<anzahl>** gibt die Länge des formatierten Feldes an

- <füllzeichen> gibt das Zeichen an, das zum Auffüllen überzähliger Zeichen benutzt wird (wenn Feldlänge kleiner **<anzahl>**).

- Ist die Feldlänge größer als **<anzahl>** wird ohne Warnung abgeschnitten. Wo abgeschnitten wird, bestimmt der Parameter links/rechts.

- Es empfiehlt sich, Zahlen mit Vornullen links aufzufüllen und Text mit Leerzeichen rechts.

- Hinweis: Durch das Aneinanderfügen von mehreren formatierten Feldern entsteht ein Grundformat für relationale Datenbanken. Das sollte aber eher mit den speziellen Grundfunktionen von MWUPDATE erfolgen, die dafür vorgesehen sind.

Steuerung des EXEC-Prozedurablaufs

Bedingte Sprunganweisungen

- bedingte (positive) Programmverzweigung:

IF EXIST(<feldname>,[<suchwort>]) GOTO <marke>

- Wenn ein **<suchwort>** angegeben ist, wird es im Feld **<feldname>** gesucht, sonst wird nur die Existenz des angegebenen Feldes überprüft.

- Ist **EXIST=true**, dann wird mit der ersten Anweisung nach der Marke **<marke>** fortgefahren - sonst wird als nächste Anweisung die nach dieser **IF**-Anweisung stehende abgearbeitet.

- Die Marke darf nur nach dieser Anweisung stehen, sonst wird Abarbeitung der EXEC-Prozedur abgebrochen.

- Existiert die Marke nicht, wird die EXEC-Prozedur abgebrochen. Es gibt nur Vorwärtsverzweigungen.

Beispiel:

```
...
if exist(DOKUMENTART,'Journal article') goto ja
if exist(DOKUMENTART,'Book chapter') goto bc
goto sonstiges
:ja
cc(DOKTYP:J)
goto ende
:bc
cc(DOKTYP:BC)
goto ende
:sonstiges
cc(DOKTYP:X)
:ende
cc(^)
...
```

- bedingte (negative) Programmverzweigung:

IF NOT EXIST(<feldname>,[<suchwort>]) GOTO <marke>

- Es wird die Nichtexistenz des Feldes **<feldname>** bzw. des **<suchwortes>** im Feld **<feldname>** geprüft.

- Ein leeres Feld **<feldname>** wird als existierend angesehen (Feldname, aber kein Wert im Feld, Länge 0). Ggf. müssen Leerfelder abgefragt werden. Dies funktioniert nur mit Zwischenfeldfunktionen (siehe Variable VAR9).

- Ein nicht vorhandenes Feld ist kein leeres Feld.

```
if not exist(FELD) goto marke
```

Der EXIST-Wert ist false, wenn eine leerer Feldname **FELD:** im Dokument ist.

Unbedingte Sprunganweisung

```
GOTO <marke>
```

- unbedingte Verzweigung: Das Programm wird an der Stelle fortgesetzt, die mit der Marke **<marke>** gekennzeichnet ist. Wird die Marke nicht gefunden oder ist die Marke vor der GOTO-Anweisung, wird die EXEC-Prozedur abgebrochen. Wird eine Marke gleichen Namens außerhalb der Prozedur gefunden und nicht in der Prozedur, dann wird zu dieser Marke außerhalb der Prozedur gesprungen. Marken sind also nicht lokal sondern im Sinne der Scriptsprache global.

```
EXIT
```

- Beendet die Abarbeitung einer EXEC-Prozedur, ohne daß das wirkliche Ende des Programmes erreicht sein muß.

Beispiel:

```
if not exist(QUELLE,Wert) goto fehler
...Verarbeitungsteil...
exit
:fehler
cc('FEHLER:Wert WERT nicht im FELD QUELLE')
cc(^)
```

Bedeutung: Wenn ein Fehler auftrat, soll eine Nachricht ausgegeben werden, andernfalls soll nach dem Verarbeitungsteil benedet werden.

Zwischenfeld-Befehle

Wirkungsweise

Alle bisherigen Anweisungen transportieren direkt Daten von einem Eingabefeld (meist Quelldatei) zu einem Ausgabefeld (in die Ausgabedatei oder auf den Bildschirm). Die folgenden Anweisungen dagegen arbeiten alle über ein Zwischenfeld. Die Arbeitsweise ist in etwa folgende:

Ein oder mehrere Felder werden in das Zwischenfeld geladen, werden dort analysiert und bearbeitet und dann in das Ausgabefeld geschrieben.

Die Zwischenfeldbearbeitung wird mit jeweils einer speziellen Anweisung eingeleitet und beendet. Zwischen diesen Anweisungen sind nur die folgenden Anweisungen und Kommentare zugelassen.

Es gibt Unterstützung bei der Programmierung durch Anzeige inner Variablen und des Zwischenfeldes.

Beginn und Ende der Zwischenfeldbearbeitung

ZFBEGIN([<mode>])

- Initialisiert das Zwischenfeld **<ZF>.** Wenn **<mode>** größer 0 ist, dann wird der Inhalt des **<ZF>** der letzten EXEC-Prozedur übernommen.

Damit ist der EXEC-Aufruf gemeint, der in der Transformationstabelle als letzter vor dem mit mode>0 definierten EXEC-Prozedur steht.

ZFBEGIN löscht also das Zwischenfeld ohne **<mode>** oder behält seinen Inhalt wenn **<mode>>0.**

- Die Reihenfolge der EXEC-Prozedurdefinitionen ist dagegen nicht von Bedeutung.

ZFENDE

- Schreibt die Daten des Zwischenfeldes in das Ausgabefeld und löscht das Ergebnis im Zwischenfeld..

Beispiele:

```
REM ---- Transfer Tabellenteil Feldzuweisung
Prozedurdeklaration
REM          exec1 speichert das Prozedurergebnis in das Feld
EN
EXEC1 EN
REM        ABSTRACT wird unter elichem Namen in die Zieldatei
REM        geschreiben
ABSTRACT
REM        DESKRIPTOR-Feld wird zu FREETERM-Feld kopiert und
REM        umbenannt
DESKRIPTOR FREETERM
REM        weitere Prozedur, der Zielfeldname muss in der
REM        Prozedur bereitgestellt werden.
EXEC2
REM ---- Ende der Tabellendefinition mit der Zuweisung von
NWENDE
REM      Standardwert: &&&
REM      Es sind Anwendungen denkbar, bei der NWENDE nicht
REM      angegeben wird, z.B. bei zeilenweiser Ausgabe einer
REM      Liste von Feldwerten.
NWENDEe
exec1:
REM ---- Prozedurdefinitionen 1
zfbegin
...
zfcc(^)
zfende

exec2:
REM ---- Prozedurdefinitionen 1
REM      In exec2 steht der Inhalt bereit, der in exec1 bei
REM      Erreichen von zfende bearbeitet war. Mode auf 1
zfbegin(1)
...
zfcc(^)
zfende
```

Aber bei anderer Reihenfolge der Prozeduraufrufe geht es anders:

```
EXEC2 EN
```

```
REM       In exec2 steht der Inhalt von exec1 nicht zur
REM       Verfügung, weil exec1 hinter exec2 definiert ist.
AB
EXEC1
nwende

 exec1:
 zfbegin
...Verarbeitungsteil ...
zfcc(^)
 zfende

 exec2:
 zfbegin(1)
...Verarbeitungsteil ...
 zfcc(^)
 zfende
```

Anzeigen interner Variablen

***SHOWVARS**

- Das Kommando zeigt die globalen Variablen VAR1...VAR9 im aktuellen Zustand an. Nur im Testmodus, hält die Ausgabe an, weiter mit Enter.

***SHOWZF**

- Zeigt das Zwischenfeld an. Dazu muss der Testmodus eingeschaltet sein (bei der Einzelprogrammverarbeitung. Die Ausgabe hält an, wenn in irgendeiner Prozedur das Kommando SHOWZF oder SHOWVARS gefunden wird. Weiter wird mit der ENTER-Taste gegangen bis wieder eines dieser Kommandos angetroffen wird.

- Durch Bedingungen kann es sein, dass im Testmodus die Kommandos übersprungen werden. Sie müssen sich gut überlegen, wo es sinnvoll ist, die Anzeige einzubauen (z.B. an Stellen, die gegen Fehler anfällig sind.

Kopieren in das Zwischenfeld

Kopieren von Feldern

ZFCOPY(<feldname>)

{ZFCO(<feldname>)}

- Kopiert das Feld **<feldname>** in das Zwischenfeld, liegen mehrer Anweisung hintereinander, wird das Zwischenfeld kumuliert.

- Mit **ZFCOPY** wird immer an vorhandene Werte im Zwischenfeld angefügt. Gelöscht werden kann das Zwischenfeld mit **ZFCLEAR** oder **ZFDEL** oder **ZFERASE**.

Kopieren von Zeilen

ZFCOPYZEILE(<nummer>,[<trenner>])

{ZFCZ(<nummer>,[<trenner>])}

- Kopiert die Zeichenkette Nummer **<nummer>** aus dem Quellsatz, die durch den Trenner <**trenner>** abgetrennt sind. Optional sind als Trenner die Zeilenwechselzeichen **#13#10 (0D0AH)** eingestellt.

- Verwendung bei Dateien mit fester Zahl von Zeilen pro Dokument.

ZFCOPYZEILE(1,|)

Bedeutung: kopiert die 1. Zeile bis zum Zeichen | im Quelldokument in das Zwischenfeld. Die Nummer kann über einer der Variablen **VAR1** bis **VAR8** übergeben werden.

Kopieren von Aussagen

ZFCOPYAUSSAGE(<feldname>,<von>,<bis>,<trennzeichen>)

{ZFCA(...)}

- Kopiert die Feldaussage mit der Position **<von>** bis zur Feldaussage mit der Position **<bis>** des Feldes **<feldname>** in das Zwischenfeld.

- **<von>,<bis>** sind die Aussagennummern. Die Nummer kann über einer der Variablen VAR1 bis VAR8 übergeben werden.

- **<trennzeichen>** werden zwischen die Aussagen gesetzt, sobald mehr als eine Aussage im Zwischenfeld landet.

- Die Funktion erlaubt das Verarbeiten der Felder Aussage für Aussage in einer Schleife.

Kopieren von Konstanten

ZFCOPYCONST(<zeichenkette>)

{ZFCC(...)}

- Kopiert die Zeichenkette <zeichenkette> in das Zwischenfeld. Die Länge der Zeichenkette kann max. 255 Zeichen betragen. Längere Zeichenketten sind aus mehreren **ZFCOPYCONST-**Anweisungen zusammenzusetzen. ZFCC fügt immer an das Ende des Zwischenfeldes an (Position var9+1!)

Substitution von Zeichenketten

ZFSUBST(<zeichenkette1>,<zeichenkette2>,[<von>],[<bis>])

{ZFSU(...)}

- Die Funktion ersetzt im Zwischenfeld die Zeichenkette **<zeichenkette1>** durch die **<zeichenkette2>.**

- **<von>, <bis>** sind optional und geben eine Anfangs- und Endeposition an, innerhalb derer die Ersetzung wirken soll. Dabei wird mit **<von>** die Position 1 der **<zeichenkette1>** angegeben, die gelten soll, wenn die Umformung erfolgen soll und unter **<bis>** die Länge dieser **<zeichenkette1>**.

ZFSU(ANWENDUNG,Abwendung)

Bedeutung: Unbedingte Umsetzung der Zeichenkette an jeder Position im Zwischenfeld, an der **<zeichenkette1>** angetroffen wird (exakte Schreibweise!).

zfsu(01.12.98,19981201,1,8)

Bedeutung: Die **<zeichenkette1>** wird nur ersetzt, wenn sie auf Position 1 des Zwischenfeldes steht.

zfsu(Anfang,Ende,var1,var2)

Bedeutung: Die Position von „A“ wird in der Variablen **VAR1** übergeben, die Länge der **<zeichenkette1>** wird in **VAR2** übergeben.

Substitution von Codes

***ZFCODE(<zeichenkette1>,<zeichenkette2>,[<von>],[<bis>])**

- Die Funktion tauscht im Zwischenfeld die Zeichen der Zeichenkette **<zeichenkette1>** durch die Zeichen der Zeichenkette **<zeichenkette2>** aus, die an der gleichen Position in beiden Zeichenketten stehen.

- **<von>, <bis>** - sind optional. Sie geben die Anfangs- und Endeposition an, innerhalb derer die Umkodierung wirken soll.

- längere Umkodierungstabellen sind aufzuteilen, da <zeichenkette1> max. 255 Zeichen lang sein kann. Da das sehr unübersichtlich ist, ist es ratsam, kleinere Zeichenkette zum Umcodieren zu benutzen.

zfcode(A,a,1,1)

Bedeutung: Aus jedem A wird ein a, wenn es auf Position 1 des Zwischenfeldes steht.

zfcode(ABCDEFGHIJKLMNOPQRSTUVWXYZ|,abcdefghijklmnopqrstuvwxyz$,2,var9)

Bedeutung: Von Position 2 des Zwischenfeldes bis zum Zwischenfeldende (siehe var9)

zfcode(#142#132#153#148#154#129#225,#196#228#214#246#220#252#223)

Bedeutung: ASCII -> ANSI im Zwischenfeld (nur für deutsche Umlaute). Zeichendarstellung durch die Umschrift mit dezimalem Bytewert.

Numerische Variable

VAR1... VAR8=y

- Verwendung von 8 globalen Variablen z.B. zur Bestimmung der Position von Substrings in den Funktionen ZFPOS oder ZFPOS1 oder zur Ermittlung der Aussagenanzahl in MAXAUSSAGEN.

- Zählen in Schleifen

VAR9

- Enthält die aktuelle Zwischenfeldlänge, kann nur lesend verwendet werden.

Position einer Zeichenkette ermitteln

VARx=ZFPOS(<zeichenkette>,[<von>][,<bis>])

- Das Zwischenfeld wird nach <zeichenkette> durchsucht, als Ergebnis wird die Position der <zeichenkette> (linkes Zeichen) in VARx (x=1..8) gespeichert. Null als Ergebnis bedeutet, die Zeichenkette ist nicht im Zwischenfeld enthalten.

- <von>, <bis> sind optional, fehlt die Angabe, wird als Standard Zwischenfeldanfang und Zwischenfeldende angenommen.

***VARx=ZFPOS1(<zeichenkette>,[<von>][,<bis>])**

- Das Zwischenfeld wird nach den einzelnen Zeichen von Zeichenkette durchsucht. Die Position des ersten Auftreten eines dieser Zeichen wird in VARx (x= 1..8) gespeichert. Ist das Ergebnis 0, bedeutet das, daß keines der Zeichen im Zwischenfeld enthalten ist.

- <von>, <bis> sind optional, fehlt die Angabe, wird als Standard Zwischfeldanfang und Zwischenfeldende (wie var9) angenommen.

var1=zfpos1(0123456789,30,var9)

-> ermittelt die Position der ersten Ziffer im Zwischenfeld, die nach der Position 30 steht

var1=zfpos('-,.;:?!()&$%"=+#39')

-> ermittelt die Position des ersten Sonderzeichens im Zwischenfeld beachte: #39 = '

Löschen von Teilzeichenketten

ZFDELETE(<von>,<bis>)

{ZFDEL(...)}

- Löscht das Zwischenfeld vom Zeichen <von> bis zum Zeichen <bis> einschließlich dieser Zeichen; es wird also die Position des letzten zu löschenden Zeichens angegeben.

```
zfdelete(var1,var9) -> bis Zwischenfeldende
zfdelete(13,var2)
zfdelete(13,255)
```

***ZFERASE(<von>,<anzahl>)**

- Löscht im Zwischenfeld ab dem Zeichen auf Position <von> in der Länge, die durch <anzahl>-Zeichen bestimmt wird.

Einfügen von Zeichenketten

ZFINS(<zeichenkette>,<ab>)

- Fügt die Zeichenkette <zeichenkette> ab der Position <ab> in das Zwischenfeld ein

Doppelte Aussagen im Zwischenfeld löschen

ZFDOPPEL[(<austrz>)]

- Die Funktion löscht doppelte Aussagen im Zwischenfeld, wobei die amweitesten links stehenden Doppel gelöscht und das am weitesten rechts stehende erhalten bleibt. Die Funktion setzt den Aussagentrenner auf die bei <austrz> angegebene Zeichenkette.

- Die Dopplungswortlänge ist 255. Die Anzahl der Feldwerte in einem Feld ist auf 10000 eingestellt. Die Länge der einzelnen Aussagen kann größer sein, als die Dopplungsprüflänge.

Ist keine <austrz> angegeben wird der Standardaussagentrenner " | " (#32#124#32) gesetzt.

Sortieren von Feldaussagen im Zwischenfeld

ZFSORT(<richtung>)

- Die Funktion sortiert die Aussagen des Zwischenfeldes nach dem Bytewert und setzt den Standardaussagentrenner " | " (#32#124#32).

- Sortierrichtung ist aufsteigen nach Bytewert. Absteigend kann unter <richtung> angegeben werden.

- Die Sortierwortlänge ist 255. Die Anzahl sortierbarer Feldwerte in einem Feld ist auf 10000 eingestellt. Die Länge der einzelnen Aussagen kann größer sein, als die Sortierlänge.

Trimmen des Zwischenfeldes

ZFTRIMM(<zeichen>[<zeichen> ...],<zeichen>[<zeichen> ...])

- Die Funktion löscht am Anfang des Zwischenfeldes alle Zeichen, die im Funktionsaufruf links vom Komma stehen.

- Die Funktion löscht am Ende des Zwischenfeldes alle Zeichen, die im Funktionsaufruf rechts vom Komma stehen.

zftrimm(|,|)

zftrimm('|:)]','|(#123#125[')

Bedeutung: Die Zeichen **|:)]** werden am Anfang des Zwischenfeldes gelöscht, egal in welcher Reihenfolge sie dort stehen. Gleichzeitig werden die Zeichen **| (#123#125[** am Ende des Zwischenfeldes gelöscht. Auch hier ist die Reihenfolge der Zeichen am Ende egal.

#123 steht für {

#125 steht für }

Die Zeichen(ketten) links und rechts sind in Apostroph einzuschließen oder mit ihrem dezimalen Bytewert anzugeben, wenn syntaktische Symbole der Scriptsprache als Zeichen auftreten (Leerzeichen, runde Klammern, Apostroph, geschweifte Klammern.

zftrimm(#44,#61#62#63)

Codewandlung im Zwischenfeld

ZFCODE(<zeichen>[<zeichen>...],<zeichen>[<zeichen>...],<von>,<bis>)

- Die Funktion codiert positionsgerecht die Zeichen der linken Seite ind die Zeichen der rechten Seite neben dem Komma.

- Die Zeichenanzahl muss recht und links gleich groß sein, andernfalls wird ein Fehler gemeldet.

- <von> gibt eine Position im Zwischenfeld an, ab der die Codeumwandlung erfolgen soll, <bis> gibt die Endepositon für die Codewandlung an.

zfcode(ABCDEFGHIJKLMNOPQRSTUVWXYZ,abcdefghijklmnopqrstuvwxyz,2,var9)

Bedeutung: Ab der Posiiton 2 im Zwischenfeld werden die Großbuchstaben in Kleinbuchstaben umgesetzt bis VAR9 (praktisch die Endeposition im Zwischenfeld).

Programmverzweigung mit Variablenwerten

Es stehen 10 Variablen zur Verfügung.

Die Variable **VAR0** enthält den Datensatzzähler, also die fortlaufend gezählte Nummer des gelesenen Satzes (jedes Lesen der Dokumentende-Zeichenkette erhöht den Zähler, leere Dokumente sollten deshalb vermieden werden). Die Variable ist nur lesbar, nicht beschreibbar.

VAR1 bis VAR8 sind lesbare und beschreibbare numerische Variable, mit denen auch Berechnungen (Länge, Anzahl) ausgeführt werden können.

VAR9 ist eine nur lesbare Variabel, die nach jeder Funktion automatisch die Länder der gültigen Zeichenkette im Zwischenfeld angibt.

IF VARx=y GOTO - bedingte Programmverzweigung in Abhängigkeit

IF VARx>y GOTO vom Wert y der Variablen VARx (x:1..9)

IF VARx<y GOTO

IF VARx<>y GOTO - <> steht für "ungleich"

Programmschleifen

REPEAT <anweisungen> UNTIL

- Die zwischen **REPEAT** und **UNTIL** stehenden Anweisungen werden beliebig oft (bis zu 10000 mal) wiederholt. Über **IF <bedingung> GOTO MARKE** ist ein Abbruch möglich. Die Marke muß sich außerhalb der Schleife befinden.

- Der Abbruch bei 10000 ist eine Sicherheitmaßnahme gegen unbeabsichtigte ununterbrechbare Programmschleifen, die durch falsche Endebedingungen entstehen können.

Der aktuelle Wert der Variablen kann mit der Funktion **showvars** unter Test-Bedingungen - also bei angeklicktem Bildschirmdurchlauf - angezeigt werden.

Beispiel:

```
if var0>10000 goto raus -> bei 10000 Dokumenten springe zur
Marke raus

VAR1=0

REPEAT -> zählt nur die Variable bis 10, Abbruchbeispiel

VAR1=VAR1+1

IF VAR1>10 GOTO M1

UNTIL

:M1

ZFENDE
```

Die Vornamen von Autoren sollen auf ihre Initialen gekürzt werden. Der Spezialfall der Doppelt-Vornamen wird aus Platzgründen nicht gesondert behandelt.

Quellfeld:

```
AU:Müller, Manfred|Meyer, Wolfgang|Schulze, Hans-Peter von
EXEC88:
zfbegin
if not exist(AU) goto f2
zfco(AU)
zfcc(|) -> Anfügen an Feld für Endebedingungen
repeat
var1=zfpos1(', ') -> Ausgangsvariablen einstellen
var2=var1+3
var4=var1+1
var3=zfpos(|)
var5=var3-1
if var5=0 goto f1
if var1<3 goto x1
if var3<4 goto x1
if var5<3 goto m1
zfdel(var2,var5)
if var3>var9 goto x1 -> Abbruchbedingungn
:m1
zfsu(', ',XX,var1,var4) -> Umsetzen wegen Anfangsbedingung
zfsu(|,$,var3,var3)
showvars
showzf
until
:x1
zfsu(XX,'-')
zfsu(.,-)
zfsu(',',-)
zfsu(#32,-)
zfsu(')',-)
zfsu('(',-)
zfsu(#39,)
zfsu(--,-)
zfsu(-|,|)
zfins(AU:,1)
```

```
zfcc(^)
zfsu(|,$)
zfsu(-$,$)
zfsu($,';')
zfsu(';^',^)
zfsu(;,'; ')
showzf
zfende
exit
:f2
cc(AU:anonym)
cc(^)
:f1
Ergebnis: AU:Müller-M; Meyer-W; Schulze-H
```

MEMO-Feld-Operationen

Ein MEMO-Feld ist ein zusätzlicher Pufferspeicher, der über die Verarbeitung eines Dokumentes hinaus mit Daten gefüllt werden kann und auf Grund einer in einer exec-Prozedur formulierten Bedingung gezielt entladen werden kann.

Max. Länge ist 500000 Zeichen.

Damit sollte es folgendes möglich sein Text (Aussagen) in das ZF kopieren, das ZF zu bearbeiten, das ZF in das MEMO zu kopieren (kumulieren), das ZF zu löschen und dann neuen Text in das ZF zu laden, zu bearbeiten usw...

In das MEMO-Feld speichern

ZFADDMEMO

- kopiert das <ZF> vollständig in das MEMO-Feld;

- wenn MEMO-Inhalt bereits existiert wird angefügt

Zwischenfeld löschen

ZFCLEAR

- löscht das <ZF> vollständig

Aus dem MEMO-Feld in das Zwischenfeld übertragen

ZFRESTOREMEMO

- kopiert MEMO-Feld nach ZF;

- das <ZF> wird dabei vollständig überschrieben. Der MEMO-Feld-Inhalt wird nur gespeichert, wenn vor ZFENDE dieser Befehl steht. Sonst bleibt das aktuelle ZF wie es ist erhalten (evtl. ist ein ZFCLEAR erforderlich).

Löschen des MEMO-Feldes

ZFMEMOCLEAR

- löscht MEMO vollständig. Der MEMO-Inhalt wird nur mit diesem Befehl gelöscht, bleibt also auch Datensatz übergreifend erhalten, solange der Befehl nicht ausgeführt wird.

Aussagenanzahl ermitteln

VARx=MAXAUSSAGEN(<feldname>)

- ermittelt die Aussagenanzahl für das genannte Feld (0 -> Feld existiert nicht)

VARx=MAXZFAUSSAGEN(ZF)

- ermittelt die Aussagenanzahl für das Zwischenfeld

Aussagentrenner ist Standard | (#124).

Kommentare

REM kommentar

- Kommentare werden mit REM auf Position 1 der Zeile eingeleitet. Jede Zeile eines Kommentars muss mit REM eingeleitet werden.

Standardprozeduren

```
exec?:
zfbegin
REM - Lösche Memofeld
zfmemoclear
REM - Bearbeiten einzelner Aussagen von Wiederholfeldern.
REM - Nur wenn das Feld vorhanden ist verarbeiten sonst zum
Ende.
if not exist(?) goto f1
REM - Aussagenanzahl des Feldes ermitteln
var1=maxaussagen(?)
REM - Gelesene Aussagen Zähler
var2=0
REM - Schleifenbeginn
repeat
REM - Schleifendurchläufe zählen, je Durchlauf 1 Aussage
var2=var2+1
REM - Endebedingung
if var2>var1 goto raus
zfcopyaussage(?,var2,var2,|)
REM - Verarbeitungsteil für jeweile einen Feldwert
```

```
REM – Prüfen, ob noch ein sinnvoller Wert vorhanden, wenn z.B.
REM ein leeres Feld oder eine leere Aussage geladen wurde,
REM steht in var9 eine 0.
if var9<1 goto ex1
REM – Aussagentrenner anfügen, damit es ein Wiederholfeld
bleibt
zfcc(|)
REM – Wert in das MEMO-Feld laden, das wird anfügend ergänzt
zfaddmemo
REM – war der Wert nicht mehr plausibel Speichern übergehen
:ex1
REM – Löschen des Zwischenfeldes
zfclear
REM - Schleifenende
until
REM – Schleifenende bei Erreichen der Endebedingung
:raus
REM - MEMO-Feld in Zwischenfeld überschreibend zurückspeichrn
zfrestorememo
REM - überzählige Aussagentrenner wegnehmen (vorn und hinten)
zftrimm(|,|)
REM – Prüfen ob der Wert noch plausibel ist, wenn z.B. eine
REM leeres Feld geladen wurde...
if var9<1 goto f1
REM – Einfügen des Zielfeldnamens auf Position 1 des
REM Zwischenfeldes
zfins(?:,1)
REM – Einen Zeilenwechsel spendieren am Feldende.
zfcc(^)
REM – ZFENDE schreibt das Zwischenfeld in das Zieldokument
zfende
REM – Zielsprungmarke, wenn Feld fehlt. Ein leeres Feld würde
REM nicht direkt zum Sprung zu dieser marke führen.
:f1
```