Kapitel 2 – Edit Makros

Edit-Makros können genutzt werden, um den Editor über einen erweiterten Befehlssatz zu unterstützen. Grundsätzlich kann zwischen zwei Formen von Makros unterschieden werden:

- einfache Makros, die eine Aneinanderreihung diverser Edit-Kommandos enthalten
- erweiterte Makros, die sowohl Programmbefehle einer Sprache, als auch Befehle des Editors oder Dialog Manager Service enthalten.

Um umfangreichere Makros zu erstellen, können Sprachen wie Fortran, Cobol oder PL/I, oder aber CLIST oder REXX genutzt werden. In der Praxis werden im allgemeinen CLIST oder REXX zur Kodierung erweiterter Makros genutzt.

Im Weiteren werden die Erklärungen auf der Basis von REXX aufgesetzt.

Edit Makros können auch Edit-Assignment-Statements enthalten, die zur Kommunikation zwischen dem Editor und dem Marco eingesetzt werden können.

Die Makrosprache kennt für jeden Primär- oder Zeilenbefehl des Editors eine adäquate Instruktion. Daher kann das Makro generell alles, was ein Bediener im Editor kann, darüber hinaus aber auch noch einiges mehr.

Edit Makros haben Zugriff auf die Services des Dialog Managers und des Systems. Da sie letztendlich Programme sind (oder Command-Lists) sind ihre Möglichkeiten nahezu grenzenlos. Die Grenzen eines Makros liegen in erster Linie dort, wo der der Programmierer an die seinen stößt.

Anwendungsbereiche

In der Praxis werden Edit Makros zu folgenden Zwecken eingesetzt:

- wiederkehrende Befehlsgruppen zusammenfassen
- Komplexe Anwendungen vereinfachen
- Parameterübergaben
- Informationsaustausch

Die nachfolgenden Seiten zeigen für die einzelnen Anwendungen je ein kleines Beispiel.

Wiederkehrende Befehle

Das folgende Makro soll alle Zeilen in einer Adressdatei, bei denen die Postleitzahl nicht mit einer 8 beginnt, löschen:

```
/* MACDEMO1 *********************************/
ADDRESS ISREDIT "MACRO"
ADDRESS ISREDIT "RESET EXCLUDED"
ADDRESS ISREDIT "EXCLUDE ALL p'8####'"
ADDRESS ISREDIT "FLIP"
ADDRESS ISREDIT "DELETE ALL EXCLUDED"
EXIT CODE(0)
```

Die folgenden Screenshots zeigen eine Datei vor und nach Ausführung des Makros:

Vor Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO)  - LINE000 COLS 001 072
COMMAND ==>  macdemo1                  SCROLL ==> PAGE
****** ************** TOP OF DATA ********************
000001 HUGO   MUELLER   61333 NEBELHORN ADELESTR. 2
000002 AUGUST BACHMAIER 85737 ISMANING  WASSERTURM 7
000003 BRILLO HELMSTEIN 47111 FEUERBACH HALDE 47
000004 MIKE   NAGEL     85737 ISMANING  ORFSTR. 9
000005 HEINZ  DOOF      82319 STARNBERG JAKOBSTR.2B
000006 RESI   HOLZAPFEL 12993 BANNWALD  ORTSRAND 58
000007 ANTON  NIETNAGEL 55374 SCHILDA   BRUNFTGASSE 4
****** ************* BOTTOM OF DATA *****************
```

Nach Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO)  - LINE000 COLS 001 072
COMMAND ==>  macdemo1                  SCROLL ==> PAGE
****** ************** TOP OF DATA ********************
000001 AUGUST BACHMAIER 85737 ISMANING  WASSERTURM 7
000002 MIKE   NAGEL     85737 ISMANING  DORFSTR. 9
000003 HEINZ  DOOF      82319 STARNBERG JAKOBSTR.2B
****** ************* BOTTOM OF DATA *****************
```

Vereinfachen komplexer Anwendungen

Um komplexere Abläufe zu vereinfachen, kann ein Makro auch Logik enthalten.

```
/* MACDEMO2 *********************************/
ADDRESS ISREDIT
"MACRO"
DO CNT=1 TO 3
   "FIND 'ZEILE-?'"
   IF RC = 0 THEN "CHANGE '?' '"cnt"'"
   ELSE CNT=3
END
EXIT CODE(0)
```

Das Beispiel bewirkt nachfolgende Veränderung in einer Datei:

Vor Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO)  - LINE000 COLS 001 072
COMMAND ==>  macdemo1                  SCROLL ==> PAGE
****** ************** TOP OF DATA ********************
000001 ZEILE-?
000002 ZEILE-?
000003 ZEILE-?
000004 ZEILE-?
000005 ZEILE-?
****** ************* BOTTOM OF DATA *****************
```

Nach Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO) -  LINE000 COLS 001 072
COMMAND ==>                            SCROLL ==> PAGE
****** ************** TOP OF DATA ********************
000001 ZEILE-1
000002 ZEILE-2
000003 ZEILE-3
000004 ZEILE-?
000005 ZEILE-?
****** ************* BOTTOM OF DATA *****************
```

Parameterübergabe und Informationsaustausch

Das folgende Beispiel zählt das Vorkommen des Strings ZEILE und teilt dies mit.

```
/* REXX * MACDEMO3 ********************************/
ADDRESS ISREDIT "MACRO (PARM)"
ADDRESS ISREDIT "SEEK ALL" PARM
IF RC = 4 THEN DO
   ZEDSMSG="Nicht gefunden"
   ZEDLMSG="Zeichenfolge" PARM "wurde nicht gefunden"
END
ELSE DO
   ADDRESS ISREDIT "(CNT) = SEEK_COUNTS"
   ZEDLMSG="String" PARM ,"wurde" CNT "mal gefunden"
END
ADDRESS ISPEXEC "SETMSG MSG(ISRZ000)"
EXIT 0
```

Nach Aufruf des Makros wird die lange Nachricht ausgegeben.

Vor Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO) -  LINE000 COLS 001 072
COMMAND ==>  macdemo1                      SCROLL ==> PAGE
****** ************** TOP OF DATA ******************
000001 ZEILE-1
000002 ZEILE-2
000003 ZEILE-3
000004 ZEILE-?
000005 ZEILE-?
****** ************* BOTTOM OF DATA *****************
```

Nach Befehlsausführung

```
EDIT - WITTEMA.PDF.DATA(DEMO) -  LINE000 COLS 001 072
COMMAND ==>                              SCROLL ==> PAGE
String ZEILE wurde insgesamt 6-mal gefunden
000001 ZEILE-1
000002 ZEILE-2
000003 ZEILE-3
000004 ZEILE-?
000005 ZEILE-?
****** ************* BOTTOM OF DATA *****************
```

REXX Makros

REXX Edit Makros bestehen aus Befehlen, die in folgende Bereiche aufgeteilt werden können:

- Edit Makro Befehle
- REXX-Programmbefehle oder Kommentare
- Dialog Manager Befehle
- TSO-Kommandos

Alle Statements werden vom Befehlsprozessor gelesen. Dabei werden symbolische Variable durch ihre Werte ersetzt.

Edit Makro Kommandos

müssen in REXX an das Environment ISREDIT übergeben werden. Nach Ersetzen der symbolischen Variablen durch ihre Werte werden die Kommandos vom Editor übernommen und ausgeführt.

Beispiele für Edit Makro Befehle sind:

```
ADDRESS ISREDIT "EXCLUDE ALL"
ADDRESS ISREDIT "FIND 'zeichenfolge'"
ADDRESS ISREDIT "RFIND"
```

REXX - Programmbefehle

übernehmen die Steuerung des Ablaufes des Makros. Befehle, die innerhalb eines Edit Makros häufig vorkommen sind beispielsweise:

```
variable=wert
IF-THEN-ELSE-Anweisungen
DO-????-END-Anweisungen
EXIT-Anweisung
```

Dialog Manager Services

können innerhalb des Makros in Anspruch genommen werden, sofern die Adressierung an das Environment ISPEXEC erfolgt. Häufig benutzte Dialog Manager Services sind (Beschreibung im Kapitel 3:

```
ADDRESS ISPEXEC "SETMSG...
ADDRESS ISPEXEC "VPUT...
ADDRESS ISPEXEC "DISPLAY...
ADDRESS ISPEXEC "EDIT...
ADDRESS ISPEXEC "LMCOPY...
```

TSO-Kommandos

Jeder Befehl, der vom REXX-Interpreter nicht ausgeführt werden kann und nicht an die Umgebungen ISREDIT oder ISPEXEC adressiert wurde, wird zur Ausführung an das TSO weitergegeben. Um Probleme zu vermeiden, ist es sinnvoll, TSO-Kommandos als Literale zu hinterlegen.

Programm Makros

So wie Dialog Manager Dialoge in verschiedenen Programmiersprachen geschrieben werden können, ist dies auch bei Edit Makros der Fall. Sprachen, die unterstützt werden sind:

- PL/I
- Cobol
- Fortran
- Pascal
- APL/2 und
- Assembler

Da im weiteren Verlauf nur auf REXX als unterstützende Sprache eingegangen wird, verzichten wir hier auch auf die Beschreibung der Unterschiede zwischen REXX/CLIST- und Programm-Makros. Details können nachgelesen werden im IBM-Manual "Edit und Edit Makros".

Regeln für Edit Makros

Befehle, die in der Kommandozeile des Editors (Primary Commands) eingegeben werden können, können auch in Makros genutzt werden. Bei einigen Befehlen gibt es aber Unterschiede, ob sie direkt unter Edit, oder innerhalb eines Makros genutzt werden:

- Wird ein Befehl direkt unter Edit genutzt, sind dessen Ergebnisse (O.K.- oder Fehlermeldung) immer unmittelbar sichtbar. Wird der Befehl im Makro kodiert, werden keine Meldungen gezeigt und die aktuelle Darstellung des Bildschirmes (sichtbare Zeilen) kann unterschiedlich sein.
- Werden in einem Makro mehrere Befehle hintereinander ausgeführt, ist dies im Einzelnen am Terminal nicht sichtbar. Als Ergebnis wird der Zustand der Daten nach dem letzten ausgeführten Befehl gezeigt.
- Manche Befehle können im Makro mit zusätzlichen Operanden versehen werden, die direkt im Editor nicht unterstützt werden.

Darüber hinaus sollten bereits bei Erstellung eines Makros einige Dinge beachtet werden:

Namensvergabe

Es kann jeder beliebige alphanumerische Name bis acht Byte Länge genutzt werden. Stelle Eins des Namens darf nicht numerisch sein. Nutzen Sie sprechende Namen.

Variablen

Variablen, die in einem Edit-Makro genutzt werden, müssen die gleichen Anforderungen erfüllen wie in ISPF.

Wenn keine Variableninterpretation gewünscht ist (weil der Name der Variablen übergeben wird, nicht ihr Inhalt), werden Variablen zwischen runde Klammern gesetzt und auf das führende Ampersand(&) verzichtet.

Variablenverarbeitung

Der SCAN-Befehl steuert den Austausch von Variablen durch ihre Werte in den Makrobefehlen, bevor sie an den Editor übergeben werden. Ist der Scanmodus aktiv, werden Makrobefehle nach Ampersands durchsucht. Wird ein Ampersand unmittelbar von einem Zeichen ungleich Blank gefolgt, wird dies als der Name der Variablen interpretiert. Der Name muss durch Blank oder Punkt abgeschlossen werden. Der Punkt erlaubt die Verkettung von Variablen ohne Zwischenräume. Dies sollte berücksichtigt werden, wenn Programm-Makros kodiert werden, die keinen Interpreter im Hintergrund haben.

Variablenzuweisung

Über die Variablenzuweisung können Editor und Makro kommunizieren. Der Befehl besteht immer aus zwei Teilen, einem Schlüsselwort und einem Wert. Das Schlüsselwort repräsentiert Daten innerhalb des Editors, der Wert die Daten im Makro. Auf diese Weise können Daten zwischen Makro und Editor in beiden Richtungen ausgetauscht werden.

Die Zuweisung erfolgt von rechts nach links. Der Ausdruck (rechts von =) wird aufgelöst, das Ergebnis wird der Variablen (links von =) zugewiesen.

```
ADDRESS ISREDIT "keyword = (variable)"
ADDRESS ISREDIT "(variable)=keyword"
```

Mögliche Werte

Der Wert einer Variablenzuweisung im Makro (steht rechts vom Gleichheitszeichen) kann folgendermaßen aussehen:

- ein Literal. Einfache Literale sind eine beliebige Folge von Zeichen ungleich Blank. Abgegrenzte Literale werden von Apostrophen(') oder Anführungszeichen(") eingeschlossen.
- eine Dialogvariable, eingeschlossen in runde Klammern

Wird die Variable rechts vom Gleichheitszeichen genutzt, wird für die Wertzuweisung ihr gesamter Inhalt, inklusive sämtlicher Anführungszeichen, Apostrophe und Blanks genutzt.

Im Umfeld von REXX sind Function- und Shared-Pool vermischt. In einem Progamm-Makro muss in jedem Fall mit VGET/VPUT-Anweisungen gearbeitet werden.

Schlüsselwort-Angaben

können aus einem einfachen Schlüsselwort bestehen, oder aus einem Schlüsselwort, gefolgt von Zeilennummer oder Marke (Label). Die folgenden Beispiele verdeutlichen die unterschiedlichen Möglichkeiten:

```
ADDRESS ISREDIT "CURSOR=2,25"
ADDRESS ISREDIT "CURSOR=2 25"
```

Beide Formen der Zuweisung sind möglich. Wird mehr als ein Wert zugewiesen, sind die einzelnen Werte durch Komma oder Blank zu trennen. Werden numerische Werte übergeben, dürfen Apostrophe oder Anführungszeichen nicht genutzt werden (Anführungszeichen, die den Makrobefehl eingrenzen werden vom Interpreter bereits aufgelöst). Das nachfolgende Beispiel ist falsch:

```
ADDRESS ISREDIT "CURSOR='2','25'"
ADDRESS ISREDIT "CURSOR='2,25'"
```

Sollen mehrere Variable gefüllt oder ausgelesen werden, sieht die entsprechende Syntax folgendermaßen aus:

```
ADDRESS ISREDIT "(var1,var2)=wert1 wert2"
ADDRESS ISREDIT "var1 var2=(var3 var4)"
```

Zur Trennung von Variablen kann Komma oder Blank genutzt werden. Sollen führende Parameter übergangen werden, sind entsprechend führende Kommata zu kodieren:

```
ADDRESS ISREDIT "(,var2)=wert1 wert2"
ADDRESS ISREDIT "var1 var2=(,var4)"
```

Überlagerungen

Bei der Wertzuweisung kann eine komplexe Kombination mehrerer Variablen und Literale genutzt werden. Diese Form nennt man eine Überlagerung (Overlay). Wenn Overlays ausgeführt werden, sollten einige Dinge beachtet werden:

Werden zwei Angaben auf einer Seite des Gleichheitszeichens durch Plus(+) verbunden, führen die Zeichen rechts vom Plus zur Überlagerung der Angaben links vom Gleichheitszeichen, sofern diese Blank sind. Sehen wir uns hierzu zwei Beispiele an:

```
"LINE .ZCSR = LINE + '//'"
```

In der Zeile, in der der Cursor steht, werden die ersten beiden Stellen durch // ersetzt. Der Rest der Zeile bleibt unverändert.

```
"MASKLINE=MASKLINE + <40 '/*' 70 '*/'>"
```

Die Spalten 40/41 und 70/71 der aktuellen Maskenzeile werden jeweils mit Kommentarbeginn und -ende ergänzt.

In beiden Fällen wird die Überlagerung nur durchgeführt, sofern die Stellen im Zielbereich Blank oder deckungsgleich sind.

Im letzten Beispiel wurde eine Schablone angelegt, die folgendes Format aufweist:

```
<spalte1 literal1 spalte2 literal2 .... >
```

Die Kodierung erfolgt im Wechsel zwischen Spaltenangabe und Literal, das an der angegebenen Spalte positioniert wird. Die gesamte Schablone wird von den Zeichen Kleiner(<) und Größer(>) eingeschlossen. Spalte und/oder Literal können auch über Variable geliefert werden. Die Variablen müssen dabei in runde Klammern eingeschlossen werden. Daraus resultierende Kodierungen sehen folgendermaßen aus:

```
<(s-var1) (d-var1) (s-var2) (d-var2) ... >
<(s-var1,d-var1) (s-var2,d-var2) ..... >
<(s-var1) literal1 spalte2 (d-var2) .... >
```

Beim Einsatz von Variablen dürfen die Namen maximal achtstellig sein.

Anwendung der Variablenzuweisung

Das folgende Beispiel zeigt eine gängige Situation für eine Variablenzuweisung im Makro. Während einer ISPF-Sitzung kennt der Editor die Einstellungen aller Modi aus dem Profil. Das Makro benötigt diese Informationen aber ebenfalls. Beschränken wir uns für das folgende Beispiel auf den Zustand des Schalters CAPS.

```
"(AKTCAPS) = CAPS"
```

Zwischen runden Klammern steht der Name der Variablen, in der der CAPS-Modus (ON oder OFF) hinterlegt wird. Erkennt der Editor einen zwischen Klammern eingeschlossenen Variablennamen an einer Stelle, an der eigentlich ein Wert erwartet wird, erfolgt vor Verarbeitung eine Variablensubstitution. Wenn wir davon ausgehen, dass bei obiger Situation CAPS auf ON gesetzt war und dieser Status wiederhergestellt werden soll, kann eine der nachfolgenden Schreibweisen genutzt werden:

```
"CAPS = ON"
"CAPS = (AKTCAPS)"
"CAPS = &AKTCAPS"
"CAPS = "AKTCAPS
```

Im letzten Beispiel ersetzt der REXX-Interpreter die Variable durch ihren Inhalt, bevor der Befehl an den Editor zur Ausführung übergeben wird.

Im vorletzten übergibt der REXX-Interpreter das Literal unverändert an den Befehlsprozessor von ISPF. Dieser ersetzt die Variable &AKTCAPS.

Im zweiten Beispiel läuft ein impliziter VGET-Befehl ab, über den der Inhalt der Variablen AKTCAPS aus dem Shared-Pool gelesen wird.

Einige Informationen können zwischen Makro und Editor paarweise in beiden Richtungen ausgetauscht werden.

```
"(LBND,RBND) = BOUNDS"
```

Die linke und rechte Boundarygrenze wird in den Variablen LBND und RBND abgelegt.

```
"BOUNDS = &LBND,&RBND"
```

Die linke und rechte Boundarygrenze wird entsprechend der Werte der Variablen LBND und RBND gesetzt.

Im Kapitel "Edit Makro Befehle" kann nachgelesen werden, welche Befehle mit einer oder zwei Variablen genutzt werden können.

Datenmanipulation durch Variablenzuweisung

Variablenzuweisung kann auch eingesetzt werden, um Zeileninhalte zu sichern, zu verändern oder zu ergänzen. Sehen wir uns dazu die folgenden Befehle an:

```
ADDRESS ISREDIT "TEMPLINE = LINE 3"
```

Kopiert den Inhalt der dritten Zeile der Datei in die Variable TEMPLINE.

```
ADDRESS ISREDIT "LINE_AFTER 7 = LINE 3"
```

Kopiert den Inhalt der Zeile 3 hinter die Zeile 7

```
ADDRESS ISREDIT "LINE 9 = (TEMPLINE)"
```

Verändert den Inhalt der Zeile 9 und überschreibt ihn mit dem Inhalt der Variablen TEMPLINE.

```
ADDRESS ISREDIT "LINE_AFTER 4 = (NEWLINE)"
```

Sorgt für das Einfügen einer neuen Zeile hinter der Zeile 4.

Unterschiede zwischen Edit- und REXX-Zuweisungen.

Folgende Unterschiede müssen berücksichtigt werden:

- Edit-Zuweisungen müssen immer an ISREDIT adressiert werden. CLIST Zuweisungen erfolgen durch den SET-Befehl, REXX-Zuweisungen in der Schreibweise var=wert.
- In einer Edit-Zuweisung muss auf der linken oder rechten Seite des Gleichheitszeichens ein Schlüsselwort (eventuell gefolgt von einer Zeilennummer oder einem Label) stehen.
- Werden in Edit-Zuweisungen Variable eingesetzt, werden die Namen der Variablen zwischen Klammern und ohne führendes Ampersand geschrieben. In einigen Fällen können auch zwei Variablennamen in der Klammer kodiert werden.
- Rechenoperationen sind in einer Edit Zuweisung nicht erlaubt. In speziellen Fällen kann durch das Pluszeichen(+) für eine teilweise Überlagerung des Zeileninhaltes gesorgt werden.

Ausführung von Zeilenkommandos

Zeilenbefehle wie M(ove) können unmittelbar vom Makro aus nicht genutzt werden. Viele Zeilenkommandos können in Makros aber über Primärbefehle oder Variablenzuweisungen nachvollzogen werden. Auf diese Weise sind Aktionen wie Zeilenkommandos C(opy), M(ove) oder R(epeat) möglich. Um beispielsweise das Zeilenkommando M(ove) in einem Makro nachzuvollziehen, wird der Zeileninhalt in einer Variablen hinterlegt. Die Zeile kann anschließend gelöscht und der Inhalt der Variablen hinter einer anderen Zeile eingefügt werden.

Parameterübergabe an ein Makro

Parameter können simple oder zwischen Apostrophe eingegrenzte Zeichenketten sein. Die Übergabe kann entweder über die VPUT/VGET-Services des Dialog Managers erfolgen, oder, wie bei den Edit-Primärkommandos, durch die Angabe hinter dem Makronamen.

Sind in einem Makro Parameter vorgesehen und werden diese bei Aufruf des Makros nicht übergeben, führt dies zum Fehler. Wurde das Makro als Kommando von der Tastatur aus aufgerufen, gibt der Editor eine entsprechende Meldung aus. Die Prüfung, ob Parameter aber in der richtigen Anzahl und Reihenfolge angegeben wurden, ist Sache des Programmierers.

Sollen Parameter an ein Makro übergeben werden, müssen diese in der Kopfzeile des Makros, zwischen runden Klammern eingeschlossen, angegeben werden. Durch den Aufruf:

```
MACDEMO4 HUBER
```

und eine entsprechende Kodierung:

```
ADDRESS ISREDIT MACRO (SUCH)
```

wird der Parameter HUBER im Makro in der Variablen SUCH hinterlegt.

Werden mehrere Parameter beim Aufruf eines Makros angegeben, erfolgt die Zuweisung nach folgendem Schema:

Sind in der Kopfzeile des Makros drei Variable definiert und werden beim Aufruf drei Parameter angegeben, erfolgt die Zuweisung der Reihe nach von links nach rechts. Der erste Parameter wird der ersten Variablen im Makro zugewiesen, und so weiter.

Sind mehr Variable im Makro definiert, als Parameter übergeben werden, werden die überzähligen Variablen auf Nullstring gesetzt.

Werden mehr Parameter angegeben, als Variable vorgesehen wurden, erfolgt die Zuordnung der Reihe nach, wobei die überzähligen Parameter als Zeichenkette in der letzten Variablen gesammelt werden.

Wird dementsprechend nur eine Variable im Makro definiert und beim Startaufruf eine Reihe von Parametern angegeben, liegen diese gesammelt in der entsprechenden Variablen vor.

Parameter werden durch Blank oder Komma voneinander getrennt. Ein Parameter kann aber auch eine beliebige Zeichenfolge (mit Blanks und/oder Kommas) sein, sofern sie zwischen Anführungszeichen oder Apostrophe eingeschlossen ist. Werden mehrere Parameter in Form abgegrenzter Zeichenketten übergeben, sind diese wiederum durch Blank oder Komma voneinander zu trennen.

Veranschaulichen wir das Ganze mit einem Beispiel. Angenommen, im Makro ist folgende Kopfzeile kodiert

```
ADDRESS ISREDIT MACRO (VAR1,VAR2,REST)
```

und der Aufruf erfolgt durch

```
MACDEMO5 DER PFERD HAT VIER BEIN
```

ist das Ergebnis:

```
VAR1  DER
VAR2  PFERD
REST  HAT VIER BEIN
```

Erfolgt der Aufruf dagegen in der Form

```
MACDEMO5 'DER PFERD' 'HAT VIER BEIN'
```

ist das Ergebnis:

```
VAR1  DER PFERD
VAR2  HAT VIER BEIN
REST  nullstring
```

Meldungsausgabe mit Makros

Meldungen werden auf die gleiche Art erzeugt, wie in einem ISPF-Dialog.

Über SETMSG kann auf eine Meldung gezeigt werden, die mit der nächsten Panelaktion sichtbar wird. Mit dem DISPLAY-Service in Verbindung mit dem Schlüsselwort MSG(..). Dieses Verfahren empfiehlt sich, wenn das Makro seinerseits mit Panels arbeitet.

Unter ISPF sind drei Meldungen vorbereitet, die ohne eigene MSG-Library genutzt werden können. Die kurze und lange Nachricht kann zu gegebener Zeit in Form von Variablen vorbereitet werden.

```
 ISRZ000 '&ZEDSMSG' .ALARM=NO .HELP=ISR2MACR
'&ZEDLMSG'
ISRZ001 '&ZEDSMSG' .ALARM=NO .HELP=ISR2MACR
'&ZEDLMSG'
ISRZ002 '&ZEDSMSG' .ALARM=NO .HELP=ISR2MACR
'&ZEDLMSG'
```

Soll in einem Makro die Meldung "PARAMETER FALSCH" oder "EINGABE MUSS NUMERISCH SEIN" ausgegeben werden, bedingt dies im Makro:

```
ZEDSMSG="PARAMETER FALSCH"
ZEDLMSG="EINGABE MUSS NUMERISCH SEIN"
ADDRESS ISPEXEC "SETMSG MSG(ISRZ000)"
```

Labels in einem Makro

Labels werden genutzt, um eine Datenzeile zu benennen. Diese Namen bleiben immer fest mit der Zeile verbunden. Sie beginnen mit Punkt(.), gefolgt von maximal 7 Alphazeichen. Das erste darf kein Z sein. Z-Label sind dem Editor selbst vorbehalten.

Labelnamen und ihre Bedeutung, wie sie im Editor Standard sind.

Labelname	Bedeutung
.ZCSR	Datenzeile, in der der Cursor steht
.ZFIRST	Erste Datenzeile (kann .ZF abgekürzt werden)
.ZLAST	Letzte Datenzeile (mögliches Kürzel .ZL).
.ZFRANGE	Die erste Zeile eines vom Benutzer definierten Bereiches
.ZLRANGE	Die letzte Zeile eines vom Benutzer definierten Bereiches
.ZDEST	Zielzeile, definiert durch den Anwender

Die Label .ZCSR, .ZFIRST und .ZLAST sind nicht zeilengebunden. .ZCSR ist cursorabhängig. .ZFIRST zeigt immer auf die erste, .ZLAST immer auf die letzte Zeile.

Anwendung der Label

Im Makro erfolgt die Festlegung eines Labels durch den Zuweisungsbefehl. Label können dort auch statt der Zeilennummer eingesetzt werden.

Gehen wir davon aus, dass innerhalb eines bestimmten Bereiches eine Verarbeitung stattfinden soll. Die aktuelle Zeile, in der der Cursor steht, soll mit dem Label BEGINN versehen werden.

```
ADDRESS ISREDIT "LABEL .ZCSR = .BEGINN"
```

Soll vor dieser Zeile eine Kommentarzeile eingefügt werden, kann das wie folgt aussehen:

```
CM="/** START" COPIES("*",60)"/"
ADDRESS ISREDIT "LINE_BEFORE .BEGINN = &CM"
```

Oder

```
ADDRESS ISREDIT "LINE_BEFORE .BEGINN = (CM)"
```

Oder

```
ADDRESS ISREDIT "LINE_BEFORE .BEGINN =" CM
```

Ist ein Label bereits für eine andere Zeile vergeben, "zieht es um", sofern die Vergabe des Labels auf der gleichen hierarchischen Ebene stattfindet.

Wird mit dem Label .ZCSR gearbeitet, ist zu bedenken, dass die Befehle FIND, CHANGE, EXCLUDE, SEEK, TSPLIT und CURSOR die Position des Cursors verändern.

Wird einer Zeile ein neues Label gegeben, werden alte Label überschrieben, sofern sich das Ganze in der gleichen Hierarchieebene abspielt. Dies gilt auch für das Zurücksetzen eines Labels in der Form:

```
ADDRESS ISREDIT "LABEL .BEGINN = ' '"
```

Über die LINENUM-Zuweisung kann abhängig vom Returncode festgestellt werden, ob ein Label vorhanden ist, oder nicht.

```
ADDRESS ISREDIT "(TMP) = LINENUM .BEGINN"
IF RC = 8 ,
     THEN /* Label nicht vorhanden */
     ELSE /* Label vorhanden */
```

Details über die Abfrage der Returncodes können im Kapitel "Edit Makro Befehle" nachgelesen werden.

Bezugnahme auf Label

Makros können, ob sie ineinander verschachtelt sind oder nicht, auf Label höherer Hierarchiestufen zugreifen. Label werden der Hierarchiestufe zugewiesen, aus der sie gebildet wurden und können problemlos gleichnamig zu Labels anderer Hierarchiestufen sein. Endet ein Makro, werden alle von ihm gebildeten Label aufgelöst. Label und Hierarchiestufe können über einen Zuweisungsbefehl ermittelt werden:

```
ADDRESS ISREDIT "(LAB,HIER) = LABEL (zeile)"
```

In der ersten Variablen wird der Labelname, in der zweiten die Hierarchiestufe, beides bezogen auf die angegebene Zeilennummer des Labels, hinterlegt. Ist kein Label für diese Zeile gesetzt, werden beide Variablen auf Nullstring gesetzt.

Label können auch außerhalb der jeweiligen Hierarchiestufe erzeugt werden:

```
ADDRESS ISREDIT "LABEL zeile  = label (level)"
```

Wird ein Label in einer höheren Hierarchiestufe gesetzt, das dort bereits definiert war, wird es überschrieben und bleibt solange gültig, bis es aufs Neue überschrieben wird, oder die betreffende Hierarchiestufe aufgelöst wird.

Label, die ohne Angabe einer Hierarchiestufe, oder mit tieferer Hierarchiestufe als die aktuelle gesetzt werden, werden in der aktuellen Hierarchiestufe gebildet. Dies hat keine Auswirkung auf Labels höherer Hierarchiestufen.

Hierarchiestufen

Jedes Makro arbeitet auf einer eigenen Hierarchiestufe, für die jeweils ein eigenes Umfeld, vergleichbar zu den Functionpools im Dialog Manager, geschaffen wird. Der Terminalbediener, der interaktiv mit dem Editor arbeitet, befindet sich in der obersten hierarchischen Ebene (Stufe 0). ruft der Anwender ein Makro auf, läuft dieses auf Stufe 1. Ruft das Makro wiederum ein Makro, wird innerhalb der Hierarchie jeweils eine Stufe nach unten verzweigt. Aus der jeweils aktuellen Hierarchiestufe kann die Verschachtelungstiefe der Makros abgelesen werden.

Wenn ein Makro ohne Levelangabe nach einem Label fragt, oder ein Label als Zeilenzeiger benutzt, wird das Label zunächst innerhalb der aktuellen Hierarchiestufe gesucht. Ist es dort nicht vorhanden, wird die Suche innerhalb der Hierarchie nach oben stufenweise fortgesetzt, bis das Label gefunden wird.

Versucht ein Makro ein Label außerhalb der Hierarchiestufe zu setzen, wird folgendermaßen verfahren:

- Wird ein Wert für die Hierarchiestufe kleiner als 0 angegeben, wird 0 angenommen.
- Wird ein Wert größer der aktuellen Stufe angegeben, wird das Label innerhalb der aktuellen Stufe gesetzt. Es können keine Label für aufzurufende Makros vorbereitet werden.

Über folgenden Befehl ist ein Makro in der Lage, die aktuelle Hierarchiestufe selbst zu ermitteln:

```
ADDRESS ISREDIT "(variable) = MACRO_LEVEL"
```

Bezugnahme auf Datenzeilen und Spalten

Datenzeilen können über ihre relative Zeilennummer oder über Labels angesprungen werden. Spezielle Zeilen wie MASK, COLS, BNDS, TABS oder MSG sind keine Datenzeilen. Ausgeblendete Zeilen hingegen werden als Datenzeilen betrachtet.

Relative Zeilennummern sind unabhängig vom Nummerntyp. Die Hinweiszeile "TOP OF DATA" trägt immer die relative Nummer 0, die erste Datenzeile immer die Nummer 1, und so weiter.

Beim Einfügen oder Löschen von Datenzeilen verändern sich ab der betreffenden Zeile nach unten alle Zeilennummern.

Im Makro ist nur der editierbare Teil der Datenzeile verfügbar. Wenn "NUMBER ON" gesetzt ist, wird die Zeilennummer ausgegrenzt (bei "NUMBER ON COB" sind die ersten sechs Byte, bei "NUMBER ON" die letzten acht Byte nicht editierbar). Im Nummernmodus für Cobol wird der absoluten Spalte 7 die relative Spaltennummer 1 zugewiesen. Werden Spaltenangaben von einem Bediener bei Aufruf des Makros

mitgeliefert, sind diese mit Vorsicht anzugehen. Im Allgemeinen müssen diese Angaben gemäß dem Nummernmodus konvertiert werden.

Sollen die Zeilennummern als Daten behandelt werden, muss das Makro den aktuellen Nummernmodus speichern, den Modus "NUMBER OFF" einschalten und nach Verarbeitung den alten Zustand wieder herstellen.

Will ein Makro die aktuelle Spaltenposition des Cursors ermitteln, wenn dieser außerhalb der editierbaren Datenzeile steht, wird eine 0 übergeben. Wird die Spaltenposition auf 0 gesetzt, wird der Cursor in das Zeilenkommandofeld der aktuellen Zeile gesetzt.

Makro-Definitionen

Über den DEFINE-Befehl können folgende Aktionen durchgeführt werden:

- Aliasnamen für Edit-Primärbefehle vergeben
- Edit-Befehle durch eigene ersetzen
- Makros als Programm-Makros identifizieren
- Befehle ungültig setzen

Details können im Kapitel "Edit Makro Befehle" nachgelesen werden.

Definition von Aliasnamen

Um einen Aliasnamen für einen Befehl festzulegen, wird erst der Aliasname, dann das Schlüsselwort ALIAS und zuletzt der originäre Name des Befehles kodiert.

```
ADDRESS ISREDIT "DEFINE S ALIAS SAVE"
```

Erlaubt zur Sicherung der bearbeiteten Datei die Eingabe "S" anstelle von SAVE. Generell könnte hier auch die Edit-Command-Table bemüht werden.

Definitionen rücksetzen

Soll die oben angeführte Definition wieder zurückgenommen werden, ist folgende Kodierung nötig:

```
ADDRESS ISREDIT "DEFINE S RESET"
```

Ersetzen von Edit-Befehlen

Um einen Befehl des Editors durch ein Makro zu ersetzen, wird einer der nachfolgenden Befehle kodiert:

```
ADDRESS ISREDIT "DEFINE COPY MACRO PGM CPY"
ADDRESS ISREDIT "DEFINE COPY MACRO CMD CPY"
```

Der obere Befehl muss angewandt werden, wenn es sich um ein Programm-Makro handelt. Ist das Makro in CLIST oder REXX kodiert, ist die zweite Form anzuwenden.

Um die Definition wieder rückgängig zu machen, wird kodiert:

```
ADDRESS ISREDIT "BUILTIN COPY"
```

Suchfolge für Makroaufrufe

Wird im Editor oder Makro ein unbekannter Befehl abgesetzt, wird die SYSPROC/SYSEXEC-Verkettung nach einem gleichnamigen Member durchsucht. Wird eines gefunden, wird automatisch davon ausgegangen, dass es sich um ein Makro handelt.

Wird ein Programm-Makro aufgerufen, wird stellvertretend die Verkettung von ISPLLIB, STEPLIB und LINKLST durchsucht. Bei Programm-Makros muss der Makroname mit einem Ausrufezeichen(!) beginnen. Für den Namen stehen somit nur maximal sieben Byte zur Verfügung.

Anwendung des PROCESS-Befehles und –Schlüsselwortes

Durch den PROCESS-Befehl kann die reguläre Abarbeitung der Makrobefehle verändert werden. Der Befehl steht in Verbindung zur Schlüsselwortangabe in der Kopfzeile des Makros.

PROCESS bewirkt, dass Veränderungen im Full-Screen-Modus und Zeilenbefehle vor Interpretation von Primärkommandos ausgeführt werden.

Bei NOPROCESS wird die Veränderung editierbarer Daten, wie die Ausführung von Zeilenkommandos, zurückgehalten, bis später im Makro eine PROCESS-Anweisung erkannt, oder das Makro beendet wird.

Die Angabe NOPROCESS kann aus zweierlei Gründen sinnvoll sein:

Befehle sollen ausgeführt werden, bevor Veränderungen im editierbaren Teil oder Zeilenkommandos ausgeführt werden.

Im Makro wird ein PROCESS-Befehl kodiert, um festzulegen, wie Zeilenkommandos, die einen Bereich, eine einzelne Zeile, oder beides beschreiben, gehandhabt werden sollen. Auf diese Weise ist eine Prüfung eingegebener Zeilenkommandos sinnvoll möglich (Edit REPLACE arbeitet auf diese Weise).

Bereichsdefinitionen durch den Bediener

Um dem Anwender die Möglichkeit zur Eingabe von Block-Zeilenbefehlen zu geben, muss im Makro folgende Zeile kodiert sein:

```
ADDRESS ISREDIT "PROCESS RANGE argument"
```

Das Argument kann aus einem oder zwei Befehlen bestehen, die akzeptiert werden. Wird beispielsweise kodiert:

```
ADDRESS ISREDIT "PROCESS RANGE Y"
```

kann der Bediener einen Zeilenbefehl in folgender Weise einsetzen:

- Y auf eine einzelne Zeile
- YY als Blockbefehl (paarweise)
- Yn wobei n die Anzahl Zeilen ist, die von dem Befehl betroffen werden. Bei Befehlen, die länger als ein Byte sind, muss die Zahl n hinter dem letzten Buchstaben des Befehlswortes eingegeben werden.

PROCESS erlaubt die Angabe von maximal zwei Zeilenbefehlen. Wird dies genutzt, kann über RANGE_CMD erkannt werden, welcher der beiden Befehle getippt wurde. Der Name der Zeilenkommandos kann bis zu sechs Byte lang sein (Blockkommandos sind dann aber nicht mehr möglich). Die Zeichen BLANK, Minus(-) und Apostroph(') dürfen nicht benutzt werden.

Nachdem der PROCESS-Befehl ausgeführt ist, werden die Variablen .ZFRANGE auf die erste relative Zeilennummer und .ZLRANGE auf die letzte relative Zeilennummer des ausgewählten Bereiches (kann die gleiche wie .ZFRANGE sein) gesetzt.

Wurde kein Bereich durch Zeilenkommandos ausgewählt, werden die beiden Variablen auf die erste und letzte Datenzeile gesetzt und RC weist darauf hin, dass keine Bereiche angegeben wurden.

Zielangaben durch den Bediener

Wird im Makro

```
ADDRESS ISREDIT "PROCESS DEST"
```

kodiert, wird dem Bediener eine Zielangabe in Form der Zeilenkommandos A/B (After oder Before) erlaubt. Die Variable .ZDEST enthält in diesem Fall die relative Zeilennummer, in der die Zielortangabe steht.

Sind weder A noch B angegeben, wird .ZDEST auf die letzte Datenzeile gesetzt und über einen Returncode mitgeteilt, dass die Zielortangabe fehlte.

Recovery Makros

Tritt während einer Edit-Sitzung ein Systemfehler auf, wird dem Anwender bei Wiederanlauf ein Recovery-Menü angeboten. Hier kann, so gewollt, ein Recovery Makro zum Einsatz kommen, das alle Befehlsdefinitionen, Aliasnamen etc. wiederherstellt. Um diesen Service zu aktivieren muss in einem Recoverymakro folgende Zeile kodiert sein:

```
ADDRESS ISREDIT "RMACRO = macroname"
```

macroname ist der Name des Recovery Makros.

Fehlerbehandlung und Test

Grundsätzlich gilt: Am besten ist es, keine Fehler zu machen. Wenn sich aber Arbeit nicht vermeiden lässt, sind die ersten Resultate häufig nicht so, wie man es sich wünscht. Auch einem Makro-Insider unterlaufen Fehler, die zum Abbruch des Makros führen, oder das Makro läuft anders, als gewollt. In diesem Fall ist es am vernünftigsten, ein Makro im Ablauf zu verfolgen, wie man dies von vielen Sprachen her kennt (debugging).

Nachdem in einem Makro unterschiedlichste Elemente vereint werden können, sind auch die Variationen im Ärgernisbereich äußerst facettenreich.

Grundsätzlich können in umfangreicheren Makros folgende Quellen die Ursache für ungewollte Abläufe sein:

- Edit-Befehle
- Makro Befehle
- Dialog Management Services
- Sprachelemente der Funktion (CLIST, REXX etc.)
- TSO-Befehle

Und nicht zu vergessen: Ein Makro macht immer was man kodiert - nie was man will!

Die nachfolgenden Seiten sollen eine kleine Hilfestellung bieten und zeigen, auf welche Arten Fehler in einem Makro, oder in dessen logischem Ablauf, erkannt und behoben werden können.