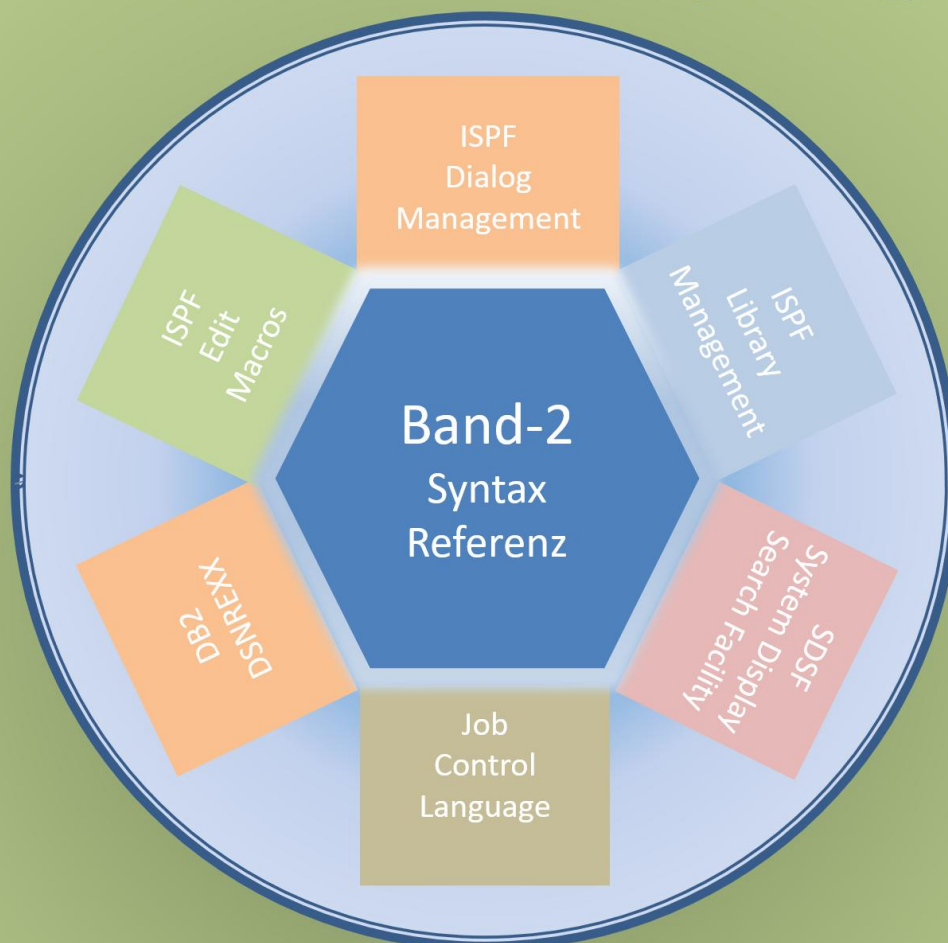


Inside z/OS



Das große TSO REXXikon

Karlheinz Wittemann

Inside z/OS

Das große TSO-REXXikon

Band 2, Befehlsreferenz

<http://www.REXXikon.de>

Geschrieben für alle, denen EDV Freude bereitet und die nicht alles tierisch ernst und verbissen sehen. Keine wissenschaftliche Abhandlung, sondern ein Buch aus der Praxis für die Praxis.

Alle in diesem Buch enthaltenen Programme und Verfahren wurden nach bestem Wissen und Gewissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das im vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Der Autor übernimmt infolgedessen keine Verantwortung und wird keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Musterschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die Übersetzung, des Nachdrucks und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Autors in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet oder vervielfältigt werden.

© 2009, 2024 Service Professional GmbH, Wiesmoor
Satz: Karl-Heinz Wittemann, Wiesmoor
Umschlaggestaltung: Claudia Wittemann, Wiesmoor

Letzter Update: 02.2025

Hinweis Syntaxbeschreibung

Endlich hat die Suche ein Ende. Da in Verbindung mit REXX eine Vielzahl von Befehlen und Funktionen genutzt werden, welche aus anderen Umgebungen und Bereichen stammen. Sollen diese hier ohne Rücksicht auf ihre Herkunft in alphabetischer Reihenfolge beschrieben werden.

Um dennoch zu erkennen, aus welcher Umgebung die Instruktion oder Funktion stammt, ist deren Herkunft beim Namen hinterlegt.

Ich habe selbst mein REXXikon von 2009 häufig in der Praxis benutzt und festgestellt, dass die Aufteilung der Syntaxbeschreibung in verschiedene Kapitel nicht wirklich vorteilhaft war. Beim groben durchsuchen, musste man sich immer im richtigen Kapitel bewegen und ich selbst habe das als eher lästig empfunden. Ja, man kann dazulernen, selbst im hohen Alter noch.

Mit der Überzeugung, dass diese Anordnung schnellere Ergebnisse beim Suchen liefert wünsche ich allen viel Spaß beim Lesen und Stöbern.

Noch ein Hinweis zu den RCs und deren Bedeutung für die ISPF-Services: Viele von denen kommen heute in der Praxis nicht mehr vor, Dennoch sind sie erläutert. Das ist der Preis der Abwärtskompatibilität. Platzprobleme und Speicherengpässe sind heute kein Thema mehr. Die Meldungen stammen zum Teil aber noch aus den Anfangszeiten (von vor ca. 50 Jahren 😊). Man darf das nicht so tierisch ernst nehmen.

Ich bitte auch um Verständnis, dass nicht wirklich jeder Parameter beschrieben wird. Manches ist antiquiert, manch anderes ist sehr exotisch. Wer benötigt in der JCL schon die Parameter zum Laden eines Zeichensatzes oder der Definition der Punktgröße der Schrift beim Druck? Sofern ihr zu den wenigen "Unglücklichen" gehört, muss ich auf das jeweilige IBM-Manual verweisen. Aber tut euch einen Gefallen. Sucht nicht alles in Google. Benutzt das Manual, das zur z/OS-Version eurer Installation passt.

Die Syntaxbeschreibung für die DB2-Befehle fällt etwas mager aus. Zum einen will das REXXikon ja nur die Schnittstelle zu DB2 aufzeigen und keine DB2-Bibel sein, zum anderen würde das jeglichen Rahmen sprengen. Hier sollt ihr gegebenenfalls die DB2-Manuals der IBM hinzuziehen. Und wer auch an dieser Stelle etwas anderes bevorzugt, dem möchte ich das Buch "DB2 in Theorie und Praxis" von Norbert Denne empfehlen. Die 12. Auflage beschäftigt sich mit DB2 Version 12.

Es fehlt etwas?

Wenn euch etwas Grundlegendes fehlt, schreibt mir einfach. Wenn sich der Aufwand in Grenzen hält, kann ich die eine oder andere Erweiterung gerne einfügen.

Lesehilfe zu den Diagrammen:

Generell orientiert man sich immer am Liniengitter. Es zeigt, ob eine Angabe zwingend ist, oder ob sie "umfahren" werden kann. Defaults sind unterstrichen. Hier einige Beispiele:

►►befehl parameter►►

Parameterangabe ist zwingend

►►befehl {parameter}►►

Parameterangabe ist wahlfrei

►►befehl {parameter
parameter}►►

Einer von mehreren Parametern ist zwingend

►►befehl {parameter
parameter}►►

Einer von mehreren Parametern kann angegeben werden

►►befehl {parameter
parameter
parameter}►►

Einer von mehreren Parametern kann angegeben werden. Der unterstrichene ist Default

►►befehl {parameter
parameter}►►

Mehr als ein Parameter können angegeben werden, aber mindestens einer.

►►befehl {parameter
parameter
parameter}►►

Mehr als ein Parameter können angegeben werden Der unterstrichene ist Default.

Inhaltsverzeichnis

.ALARM – ISPF Steuervariable Panel.....	17
.ATTR – ISPF Steuervariable Panel.....	17
.ATTRCHAR – ISPF Steuervariable Panel.....	18
.AUTOSEL – ISPF Steuervariable Panel	19
.CSRPOS – ISPF Steuervariable Panel	19
.CSRROW – ISPF Steuervariable Panel.....	20
.CURSOR – ISPF Steuervariable Panel.....	20
.HELP – ISPF Steuervariable Panel.....	21
.MSG – ISPF Steuervariable Panel.....	21
.PFKEY – ISPF Steuervariable Panel	22
.RESP – ISPF Steuervariable Panel	22
.TRAIL – ISPF Steuervariable Panel.....	23
.TYPE – ISPF Steuervariable Panel.....	23
.WINDOW – ISPF Steuervariable Panel	24
.ZVARS – ISPF Steuervariable Panel	24
)ABC – ISPF Panel Section	25
)ABCINIT – ISPF Panel Section.....	25
)ABCPROC – ISPF Panel Section	26
)AREA – ISPF Panel Section	26
)ATTR – ISPF Panel Section	26
)BLANK – ISPF File Tailoring Steueranweisung	30
)BODY – ISPF Panel Section.....	31
)CCSID – ISPF Panel Section.....	32
)CM – ISPF File Tailoring Steueranweisung.....	33
)DEFAULT – ISPF File Tailoring Steueranweisung	34
)DO – ISPF File Tailoring Steueranweisung.....	34
)DO wert – ISPF File Tailoring Steueranweisung	35
)DO – Iteration – ISPF File Tailoring Steueranweisung	35
)DO FOREVER – ISPF File Tailoring Steueranweisung.....	36
)DO WHILE – ISPF File Tailoring Steueranweisung.....	36
)DO UNTIL – ISPF File Tailoring Steueranweisung.....	37
)DOT – ISPF File Tailoring Steueranweisung.....	37
)ELSE – ISPF File Tailoring Steueranweisung	37
)END – ISPF Panel Section	37
)ENDDO – ISPF File Tailoring Steueranweisung	38
)ENDDOT – ISPF File Tailoring Steueranweisung	38
)ENDREXX – ISPF File Tailoring Steueranweisung	38
)ENDSEL – ISPF File Tailoring Steueranweisung	38
)FIELD – ISPF Panel Section.....	39
)HELP – ISPF Panel Section	43
)IF – ISPF File Tailoring Steueranweisung	43
)INEXIT – ISPF Panel Section	44
)ITERATE – ISPF File Tailoring Steueranweisung	44
)IM – ISPF File Tailoring Steueranweisung	44
)INIT – ISPF Panel Section	45
)LEAVE – ISPF File Tailoring Steueranweisung.....	45
)LIST – ISPF Panel Section	46
)MODEL – ISPF Panel Section	47
)NOP – ISPF File Tailoring Steueranweisung	48

)PROC – ISPF Panel Section	48
)REINIT – ISPF Panel Section	48
)REXX – ISPF File Tailoring Steueranweisung	49
)SEL ISPF – File Tailoring Steueranweisung	50
)SET – ISPF File Tailoring Steueranweisung	50
)SETF – ISPF File Tailoring Steueranweisung	51
)TAB – ISPF File Tailoring Steueranweisung	51
&EVAL() – ISPF File Tailoring Function	52
&LEFT() – ISPF File Tailoring Function	52
&LENGTH() – ISPF File Tailoring Function	53
&RIGHT() – ISPF File Tailoring Function	53
&STR() – ISPF File Tailoring Function	54
&STRIP() – ISPF File Tailoring Function	54
&SUBSTR() – ISPF File Tailoring Function	55
&SYMDEF() – ISPF File Tailoring Function	56
*ENDREXX – ISPF Panel Function	56
*REXX – ISPF Panel Function	56
ABDISPCC – JCL EXEC Parameter	57
ABBREV() – REXX Built-in-Function	57
ABS() – REXX Built-in-Function	58
ADDPPOP – ISPF DM-Service	58
ADDRESS – REXX Instruction	59
ADDRESS() – REXX Built-in-Function	60
ADDRSPC – JCL JOB Parameter	60
ALLOC – TSO Kommando	61
ALTLIB – TSO-Kommando	64
ARG() – REXX Built-in-Function	67
ATTRIB – TSO Command	68
AUTOLIST – Edit Macro	69
AUTONUM – Edit Macro	69
AUTOSAVE – Edit Macro	70
BITAND() – REXX Built-in-Function	71
BITOR() – REXX Built-in-Function	71
BITXOR() – Built-in-Function	72
BLOCKSIZE – ISPF Edit Macro	72
BOUNDARIES – ISPF Edit Macro	73
BPXWDYN – UNIX Command	74
BPXWDYN ALLOC – UNIX Command	75
BPXWDYN CONCAT – UNIX Command	80
BPXWDYN FREE – UNIX Command	81
BPXWDYN INFO – UNIX Command	82
BPXWDYN Return Codes – Unix Command	83
BROWSE – ISPF DM Service	84
BUILTIN – ISPF Edit Macro	85
B2X() – REXX Built-in-Function	85
CALL – REXX Instruction	86
CALL – TSO Command	87
CANCEL – ISPF Edit Macro	88
CANCEL – TSO Command	88
CAPS – ISPF Edit Macro	89
CENTER() – REXX Built-in-Function	90
CHANGE – ISPF Edit Macro	91
CHANGE_COUNTS – ISPF Edit Macro	93
CHARIN() – UNIX Function	93

CHAROUT() – UNIX Function.....	94
CHARS() – UNIX Function.....	95
CLASS – JCL JOB Parameter	95
COMMAND – JCL Special Statement.....	96
COMMIT – DB2 Instruction	96
COMPARE – ISPF Edit Macro	97
COMPARE() – REXX Built-in-Function	98
COND – JCL JOB Parameter.....	99
COND – JCL EXEC Parameter.....	99
CONDITION() – REXX Built-in-Function	101
CONNECT – DB2 Instruction.....	102
CONTROL – ISPF DM Service	102
COPIES() – REXX Built-in-Function.....	105
COPY – ISPF Edit Macro.....	105
CREATE – ISPF Edit Macro	106
CREATE DATABASE – DB2 Instruction	107
CREATE TABLE – DB2 Instruction	107
CREATE TABLESPACE – DB2 Instruction.....	109
CTL_LIBRARY – ISPF Edit Macro	110
CURSOR – ISPF Edit Macro.....	111
CUT – ISPF Edit Macro.....	112
C2D() - Character to Decimal – REXX Built-in-Function.....	113
C2X() – Character to Hex – REXX Built-in-Function.....	113
DATA – JCL DD Parameter	114
DATACLASS – JCL DD Parameter.....	114
DATA_CHANGED – ISPF Edit Macro.....	115
DATA_WIDTH – ISPF Edit Macro	115
DATAID – Edit Macro.....	116
DATASET – ISPF Edit Macro.....	116
DATATYPE() – REXX Built-in-Function	117
DATE() – REXX Built-in-Function.....	118
DBCS() – REXX Built-in-Function	119
DCB – JCL DD Parameter	119
DDNAME – JCL DD Parameter	120
DEFINE – ISPF Edit Macro.....	121
DELETE – DB2 Instruction.....	122
DELETE – ISPF Edit Macro	122
DELETE – TSO Command	123
DELSTACK - TSO extended Command.....	124
DELSTR() – REXX Built-in-Function	125
DELWORD() – REXX Built-in-Function	125
DIGITS() – REXX Built-in-Function	125
DIRLIST – ISPF DM Service	126
DISCONNECT – DB2 Instruction	127
DISP – JCL DD Parameter	127
DISPLAY – ISPF DM Service	129
DISPLAY_COLS – ISPF Edit Macro	130
DISPLAY_LINES – ISPF Edit Macro.....	131
DO FOREVER – REXX Instruction	132
DO wert – REXX Instruction.....	132
DO var = wert TO ende (Iteration) – REXX Instruction	133
DO WHILE UNTIL bedingung – REXX Instruction.....	133
DOWN – ISPF Edit Macro.....	134
DROP – DB2 Instruction	135

DROP – REXX Instruction	135
DROPBUF - TSO extended Command	135
DSINFO – ISPF DM Service	136
DSN – JCL DD Parameter	138
DSNTYPE – JCL DD Parameter	140
DUMMY – JCL DD Parameter	141
DYNAMNBR – JCL EXEC Parameter	141
D2C() – Decimal to Character – REXX Built-in-Function	141
D2X() – Decimal to Hex – REXX Built-in-Function	142
EATTR – JCL DD Parameter	142
EDIT – ISPF DM Service	143
EDIT – ISPF Edit Macro	144
EDIT – TSO Command	145
END – ISPF Edit Macro	145
ERRORTEXT() – REXX Built-in-Function	145
EXCLUDE – ISPF Edit Macro	146
EXCLUDE_COUNTS – ISPF Edit Macro	147
EXEC – TSO Command	148
EXECIO - TSO extended Command	148
EXECUTIL – TSO Command	151
EXIT – ISPF Panel Instruktion	154
EXIT – REXX Instruction	154
EXPDT – JCL DD Parameter	155
EXTERNALS() – NonSAA REXX Built-in-Function	155
FETCH – DB2 Instruction	155
FILEDATA – JCL DD Parameter	156
FIND – ISPF Edit Macro	157
FIND() – REXX Built-in-Funktion	158
FIND_COUNTS – ISPF Edit Macro	159
FLIP – ISPF Edit Macro	159
FLOW_COUNTS – ISPF Edit Macro	160
FORM() – REXX Built-in-Function	160
FORMAT() – REXX Built-in-Function	161
FREE – JCL DD Parameter	161
FREE – TSO Command	162
FTCLOSE – ISPF DM Instruktion	163
FTERASE – ISPF DM Instruktion	164
FTINCL – ISPF DM Instruktion	164
FTOPEN – ISPF DM Instruktion	165
FUZZ() – REXX Built-in-Function	165
GDGORDER – JCL DD Parameter	166
GETMSG – ISPF DM Instruktion	166
GETMSG() – TSO external Function	167
GOTO – ISPF Panel Instruction	168
GROUP – JCL JOB Parameter	168
HEX – ISPF Edit Macro	169
HIDE – ISPF Edit Macro	169
HILITE – ISPF Edit Macro	170
IF – ELSE – ISPF Panel Instruction	173
IF – THEN – ELSE – JCL Special Statement	175
IF – THEN – ELSE – REXX Instruction	179
IMACRO – ISPF Edit Macro	179
INCLUDE – JCL Special Statement	180
INDEX() – NonSAA REXX Built-in-Function	181

INSERT – DB2 Instruction	181
INSERT – ISPF Edit Macro	182
INSERT() – REXX Built-in-Function	182
INTERPRET – REXX Instruction.....	183
ISFACT – SDSF-API-Aufruf	184
ISFBROWSE – SDSF-API-Aufruf	185
ISFEXEC – SDSF-API-Aufruf.....	185
ISFGET – SDSF-API-Aufruf	186
ISFLOG – SDSF-API-Aufruf.....	186
ISFSLASH – SDSF-API-Aufruf.....	187
ITERATE – REXX Instruction.....	188
JCLLIB – JCL Special Statement	189
JOBLIB – JCL Special DD Statement	189
JOBPARM – JCL JES Instruction	190
JUSTIFY() – REXX Built-in-Function.....	191
LABEL – ISPF Edit Macro	191
LABEL – JCL DD Parameter.....	192
LASTPOS() – REXX Built-in-Function.....	193
LEAVE – REXX Instruction	193
LEFT – ISPF Edit Macro	194
LEFT() – REXX Built-in-Function	195
LENGTH() – REXX Built-in-Function.....	195
LEVEL – Edit	196
LIBDEF – ISPF DM Instruction.....	197
LIKE – JCL DD Parameter	198
LINE – ISPF Edit Macro	198
LINEIN() – UNIX Function.....	199
LINENUM – ISPF Edit Macro	200
LINEOUT() – UNIX Function.....	201
LINES() – UNIX Function	202
LINESIZE() – NonSAA-Built-in-Function	202
LINE_AFTER – ISPF Edit Macro	203
LINE_BEFORE – ISPF Edit Macro	204
LINE_STATUS – ISPF Edit Macro	205
LISTALC – TSO Command.....	207
LIST – ISPF DM Service	207
LISTBC – TSO Command.....	208
LISTCAT – TSO Command	208
LISTDS – TSO-Kommando.....	209
LISTDSI() – TSO extended Function.....	210
LMCLOSE – ISPF LM Service	211
LMCOMP – ISPF LM Service.....	212
LMCOPY – ISPF LM Service	213
LMDDISP – ISPF LM Service	214
LMDFREE – ISPF LM Service	216
LMINIT – ISPF LM Service	216
LMDLIST – ISPF LM Service	217
LMERASE – ISPF LM Service	219
LMFREE – ISPF LM Service.....	220
LMGET – ISPF LM Service	220
LMINIT – ISPF LM Service	221
LMMADD – ISPF LM Service.....	223
LMMDEL – ISPF LM Service	224
LMMDISP – ISPF LM Service.....	225

LMMFIND – ISPF LM Service	229
LMMLIST – ISPF LM Service	230
LMMOVE – ISPF LM Service	232
LMMREN – ISPF LM Service	234
LMMREP – ISPF LM Service	234
LMMSTATS – ISPF LM Service	235
LMOPEN – ISPF LM Service	237
LMPRINT – ISPF LM Service	238
LMPUT – ISPF LM Service	239
LMQUERY – ISPF LM Service	240
LMRENAME – ISPF LM Service	243
LOCATE – ISPF Edit Macro	243
LOG – ISPF DM Service	245
LRECL – ISPF Edit Macro	246
LRECL – JCL DD Parameter	246
MACRO – ISPF Edit Macro	247
MACRO_LEVEL – ISPF Edit Macro	248
MASKLINE – ISPF Edit Macro	248
MAKEBUF – TSO extended Command	249
MAX() – REXX Built-in-Function	250
MAXGENS – JCL DD Parameter	250
MEMBER – ISPF Edit Macro	250
MEMLIST – ISPF LM Service	251
MEND – ISPF Edit Macro	252
MESSSAGE – JCL JES Instruction	252
MGMTCLAS – JCL DD Parameter	253
MIN() – REXX Built-in-Function	253
MODEL – ISPF Edit Macro	253
MOVE – ISPF Edit Macro	254
MSG() – TSO extended Function	255
MSGCLASS – JCL JOB Parameter	256
MSGLEVEL – JCL JOB Parameter	256
MVSVAR() – TSO extended Function	256
NEWSTACK – TSO extended Command	257
NONUMBER – ISPF Edit Macro	257
NOP – REXX Instruction	257
NOTE – ISPF Edit Macro	258
NOTIFY – JCL JOB Parameter	259
NULLS – ISPF Edit Macro	259
NUMBER – ISPF Edit Macro	260
NUMERIC – REXX Instruction	262
OPEN – DB2 Instruction	263
OUTPUT – JCL Special Statement	263
OUTTRAP() – TSO extended Function	265
OVERLAY() – REXX Built-in-Function	267
PACK – ISPF Edit Macro	267
PARM – JCL EXEC Parameter	268
PARMDD – JCL EXEC Parameter	268
PARSE – REXX Instruction	268
PASSWORD – JCL JOB Parameter	270
PASTE – ISPF Edit Macro	270
PATH – JCL DD Parameter	271
PATHDISP – JCL DD Parameter	271
PATHMODE – JCL DD Parameter	272

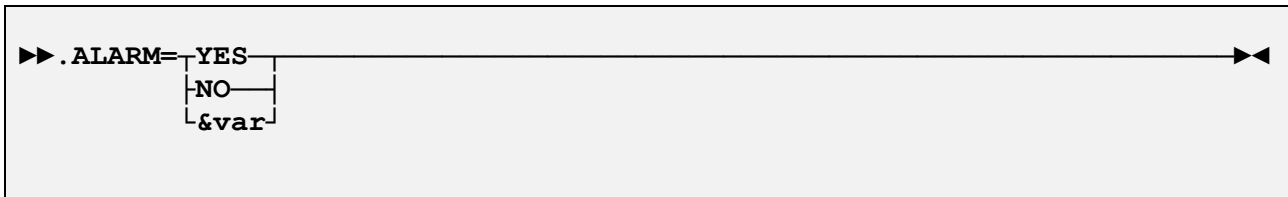
PATHOPTS – JCL DD Parameter.....	272
PEND – JCL Special Statement.....	273
PERFORM – JCL JOB Parameter	273
POS() – REXX Built-in-Function.....	274
PQUERY – ISPF DM Instruction	274
PREPARE – DB2 Instruction	275
PRESERVE – ISPF Edit Macro.....	276
PRIORITY – JCL JES Instruction.....	276
PROC – JCL Special Statement	277
PROCEDURE – REXX Instruktion	278
PROCESS – ISPF Edit Macro.....	279
PROFILE – ISPF Edit Macro.....	280
PROFILE – TSO Command.....	281
PROMPT() – TSO extended Function.....	282
PRTY – JCL JOB Parameter	282
PUSH – REXX Instruction.....	283
QBASELIB – ISPF DM Instruction	283
QBUF - TSO extended Command	284
QELEM - TSO extended Command.....	284
QLIBDEF – ISPF DM Service	285
QTABOPEN – ISPF DM Service.....	285
QUERYENQ – ISPF DM Service	286
QUEUE – REXX Instruction	288
QUEUED() – REXX Built-in-Function.....	288
QSTACK - TSO extended Command.....	288
RANGE_CMD – ISPF Edit Macro	290
RCHANGE – ISPF Edit Macro	290
RECEIVE – TSO Command (privilegiert)	291
RECFM – ISPF Edit Macro	293
RECFM – JCL DD Parameter	294
RECOVERY – ISPF Edit Macro.....	295
REFDD – JCL DD Parameter.....	295
REGION – JCL JOB-/EXEC Parameter	296
REMPop – ISPF DM Service	296
RENAME – TSO Command.....	296
RENUMBER – ISPF Edit Macro.....	297
REPLACE – ISPF Edit Macro	297
RESET – ISPF Edit Macro	298
RETPD – JCL DD Parameter.....	299
RETURN – REXX Instruction.....	299
REVERSE() – REXX Built-in-Function	300
RFIND – ISPF Edit Macro	301
RIGHT – ISPF Edit Macro.....	301
RIGHT() – REXX Built-in-Function	302
RMACRO – ISPF Edit Macro	302
ROLLBACK – DB2 Instruction.....	303
ROUTE – JCL JES Instruction	303
SAVE – ISPF Edit Macro	303
SAVE_LENGTH – ISPF Edit Macro	304
SAY – REXX Instruction.....	304
SCAN – ISPF Edit Macro	305
SCHENV – JCL JOB Parameter	306
SEEK – ISPF Edit Macro	306
SEEK_COUNTS – ISPF Edit Macro.....	308

SELECT – DB2 Instruction.....	308
SELECT – ISPF DM Service.....	309
SELECT – REXX Instruction.....	311
SEND – TSO-Kommando.....	311
SESSION – ISPF Edit Macro.....	312
SET – JCL Spezielle Statement.....	313
SET CURRENT SQLID – DB2 Instruction.....	313
SETLANG() – TSO extended Function.....	314
SETMSG – ISPF DM Service.....	314
SETUNDO – ISPF Edit Macro.....	315
SHIFT COLUMN – ISPF Edit Macro.....	316
SHIFT DATA – ISPF Edit Macro.....	317
SIGN() – REXX Built-in-Function.....	318
SIGNAL – REXX Instruction.....	318
SORT – ISPF Edit Macro.....	319
SOURCELINE() – REXX Built-in-Function.....	321
SPACE() – REXX Built-in-Function.....	322
STATS – ISPF Edit Macro.....	322
STATUS – TSO Command.....	323
STEPLIB – JCL Special DD Statement.....	323
STORAGE() – TSO extended Function.....	323
STORCLAS – JCL DD Parameter.....	324
SYSOUT – JCL DD Parameter.....	324
STREAM() – UNIX Function.....	325
STRIP() – REXX Built-in-Function.....	327
SUBCOM – TSO extended Command.....	327
SUBMIT – ISPF Edit Macro.....	328
SUBMIT – TSO Command.....	329
SUBSTR() – REXX Built-in-Function.....	329
SUBWORD() – REXX Built-in-Function.....	330
SYMBOL() – REXX Built-in-Function.....	330
SYMBOLS – JCL DD Parameter.....	331
SYMLIST – JCL DD Parameter.....	332
SYSABEND SYSMDUMP SYSUDUMP – JCL Special DD Statement.....	332
SYSCPU() – TSO extended Function.....	333
SYSDSN() – TSO extended Function.....	333
SYSVAR() – TSO extended Function.....	334
TABS – ISPF Edit Macro.....	335
TABSLINE – ISPF Edit Macro.....	336
TBADD – ISPF DM Service.....	337
TBBOTTOM – ISPF DM Service.....	338
TBCLOSE – ISPF DM Service.....	339
TBCREATE – ISPF DM Service.....	340
TBDELETE – ISPF DM Service.....	342
TBDISPL – ISPF DM Service.....	343
TBEND – ISPF DM Service.....	348
TBERASE – ISPF DM Service.....	348
TBEXIST – ISPF DM Service.....	349
TBGET – ISPF DM Service.....	349
TBMOD – ISPF DM Service.....	350
TBOPEN – ISPF DM Service.....	351
TBPUT – ISPF DM Service.....	352
TBQUERY – ISPF DM Service.....	353
TBSARG – ISPF DM Service.....	354

TBSAVE – ISPF DM Service	356
SUBSYS – JCL DD Parameter	357
TBSCAN – ISPF DM Service	357
TBSKIP – ISPF DM Service	359
TBSORT – ISPF DM Service	360
TBSTATS – ISPF DM Service	362
TBTOP – ISPF DM Service	364
TBVCLEAR – ISPF DM Service	364
TENTER – ISPF Edit Macro	365
TFLOW – ISPF Edit Macro	365
TIME - JOB-Anweisung	366
TIME() – REXX Built-in-Function	366
TRACE – REXX Instruction	367
TRACE() – REXX Built-in-Function	369
TRANS – ISPF DM Service	369
TRANSLATE() – REXX Built-in-Function	371
TRANSMIT – TSO Command (privilegiert)	371
TRUNC() – REXX Built-in-Function	372
TSOEXEC – TSO Command	372
TSOLIB – TSO Command	373
TSPLIT – ISPF Edit Macro	375
TRANS – ISPF Panel Function	376
TRAPMSG() – TSO extended Function	377
TRUNC – ISPF Panel Function	378
TYPRUN – JOB-Anweisung	378
UNIT – JCL DD Parameter	379
UNNUM – ISPF Edit Macro	380
UP – ISPF Edit Macro	380
UPDATE – DB2 Instruction	381
UPPER – REXX Instruction	382
USERID() – TSO external	382
USER_STATE – ISPF Edit Macro	383
VALUE() – REXX Built-in-Function	383
VER – ISPF Panel Function	384
VERASE – ISPF DM Service	389
VERIFY() – REXX Built-in-Function	389
VERSION – ISPF Edit Macro	390
VGET – ISPF DM Service	391
VIEW – ISPF DM Service	392
VOLUME – ISPF Edit Macro	394
VPUT – ISPF DM Service	395
VSYM – ISPF DM Service	395
WORD() – REXX Built-in-Function	396
WORDINDEX() – REXX Built-in-Function	396
WORDLENGTH() – REXX Built-in-Function	397
WORDPOS() – REXX Built-in-Function	397
WORDS() – REXX Built-in-Function	397
XEQ – JCL JES-Statement	398
XRANGE() – REXX Built-in-Function	398
XSTATUS – ISPF Edit Macro	398
X2B() – REXX Built-in-Function	399
X2C() – REXX Built-in-Function	400
X2D() – REXX Built-in-Function	400
Z-Variablen des Dialog Managers	400

Datum und Zeit	401
Generelle Informationen.....	401
Bildschirm und PF-Tasten.....	401
Scrolling	402
Table Display Service	402
Hilite-Informationen.....	403

.ALARM – ISPF Steuervariable Panel



Kann innerhalb einer Bildschirmmaske in den Sektionen)INIT,)REINIT und

)PROC genutzt werden, um einen Signalton am Terminal abzusetzen. Die Zuweisung kann durch Schlüsselwortangaben, oder über Variable erfolgen.

YES

Das Signal ertönt, wenn das nächste Panel angezeigt wird.

NO

Der Signalton wird unterdrückt.

&var

Der Inhalt der Variablen entscheidet, ob ein Signalton zu hören sein wird, oder nicht.

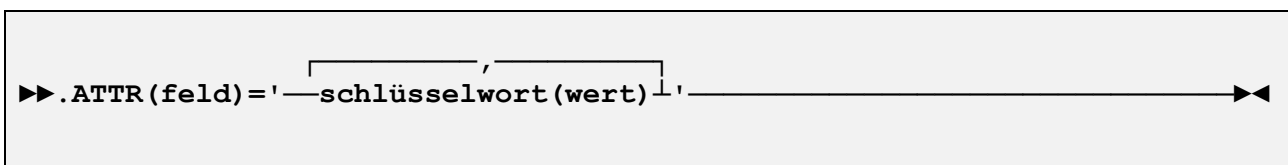
Die Variable kann ausschließlich die Inhalte "YES" oder "NO" haben.

Die Variable kann auch gebildet werden, indem der aktuelle Wert von .ALARM zugewiesen wird. In diesem Fall kann die Steuervariable auch rechts vom Gleichheitszeichen(=) bei der Variablenzuordnung genutzt werden.

Beispiele

```
.ALARM=YES
.ALARM=NO
&SOUND=.ALARM
.ALARM=&SOUND
```

.ATTR – ISPF Steuervariable Panel



Kann innerhalb eines Panels in den Sektionen)INIT,)REINIT und)PROC genutzt werden, einem Ein- oder Ausgabefeld im)BODY-Teil des Panels ein anderes Feldattribut zuzuweisen. Als Feldname kann angegeben werden:

- Jedes Input- oder Outputfeld im)BODY-Teil
- Die Steuervariable .CURSOR
- Eine Dialogvariable (mit führendem Ampersand(&)), deren Inhalt wiederum ein- oder Ausgabefeldname aus dem)BODY-Teil, oder ".CURSOR" sein kann.

Auf diesem Weg das Attributmerkmal eines Feldes zu ändern ist bei Fehlern eine wertvolle optische Hilfe für den Bediener. So lassen sich falsch ausgefüllte Felder farbig, intensiv oder blinkend mit einer Fehlermeldung kombinieren.

schlüsselwort(wert)

Alle Angaben, wie sie im)ATTR-Teil des Panels möglich sind.

Hinweis

Wird ein Attribut in der)INIT-Section überschrieben, bleibt das neue Attribut erhalten, bis es erneut verändert wird. Werden Attributveränderungen im)REINIT- oder)PROC-Teil durchgeführt, sind sie nur für das Re-Display gültig. Hier kann fehlerverursachende Feld auf ROT gesetzt werden, um im Re-Display den Benutzer die Fehlersuche zu erleichtern.

Beispiele

```
.ATTR (.CURSOR) = ' COLOR (YELLOW) HILITE (REVERS) '
```

Das Attribut des Cursorfeldes wird auf GELB invers geändert.

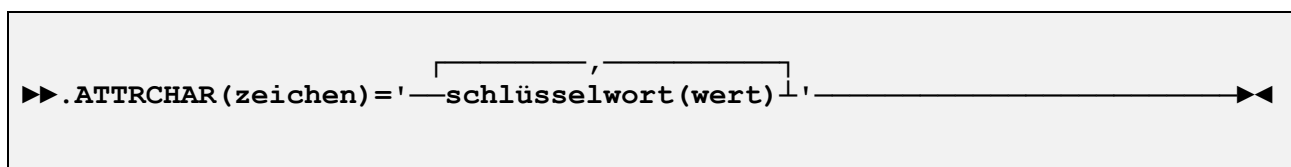
```
)REINIT
```

```
.ATTR (.CURSOR) = ' COLOR (RED) HILITE (USCORE) '
```

```
.ATTR (MSG) = ' HILITE (BLINK) '
```

Diese Angaben sorgen dafür, dass das Feld, in dem ein Fehler erkannt wurde rot und unterstrichen dargestellt wird und die kurze Meldung blinkend erscheint.

.ATTRCHAR – ISPF Steuervariable Panel



Kann innerhalb einer Bildschirmmaske in den Sektionen)INIT,)REINIT und)PROC genutzt werden, allen Feldern, die das angegebene Attributzeichen führen, andere Attribute zuzuweisen.

Zeichen

Eines der Attributkennzeichen im)INIT-Teil, oder der Name einer Dialogvariablen, deren Inhalt wiederum eines der Attributkennzeichen sein kann.

Hinweis

Wird ein Attribut in der)INIT-Section überschrieben, bleibt dieses solange erhalten, bis es erneut verändert wird. Werden dagegen Attributänderungen im)REINIT- oder)PROC-Teil des Panels durchgeführt, sind sie nur für das Re-Display gültig. Hier kann beispielsweise das Feldattribut eines falsch ausgefüllten Eingabefeldes auf ROT geändert werden, um im Re-Display den Benutzer auch optisch möglichst schnell an das falsch gefüllte Feld zu führen. Erfolgt an der entsprechenden eine gültige Eingabe, wird dem Feld wieder sein Standardattribut zugewiesen.

Beispiel

```
.ATTRCHAR($)='INTENS (LOW) CAPS (OFF) '
```

Das Attribut aller Felder, die ein führendes Dollarzeichen(\$) haben wird entsprechend den Angaben geändert.

.AUTOSEL – ISPF Steuervariable Panel

The diagram shows a horizontal panel with a double arrow on the left and a double arrow on the right. Inside the panel, the text ".AUTOSEL=" is followed by a bracketed selection. The selection has two options: "YES" and "NO". The "YES" option is currently selected, indicated by a vertical line to its left.

Diese Steuervariable wird nur in Verbindung mit Bildschirmmasken zur Tabellenanzeige benutzt. Fehlt die Angabe, wird ".AUTOSEL=YES" angenommen. Es wird gesteuert, in welcher Weise die Auswahl von Tabellensätzen bei der Anzeige durchgeführt wird.

Die Details können bei der Beschreibung des Befehles TBDISPL nachgelesen werden.

.CSRPOS – ISPF Steuervariable Panel

The diagram shows a horizontal panel with a double arrow on the left and a double arrow on the right. Inside the panel, the text ".CSRPOS=" is followed by a bracketed selection. The selection has two options: "wert" and "&var". The "wert" option is currently selected, indicated by a vertical line to its left.

kann im)INIT- oder)REINIT-Teil angegeben werden. Es bestimmt die Stelle innerhalb eines Feldes, an welche der Cursor positioniert wird. Die Angabe bezieht sich auf das Feld, auf das der Cursor positioniert ist, gleichgültig, wie diese Positionierung zustande kam (Standard, Fehlererkennung oder .CURSOR).

Die Steuervariable kann auch auf der rechten Seite einer Wertzuweisung genutzt werden.

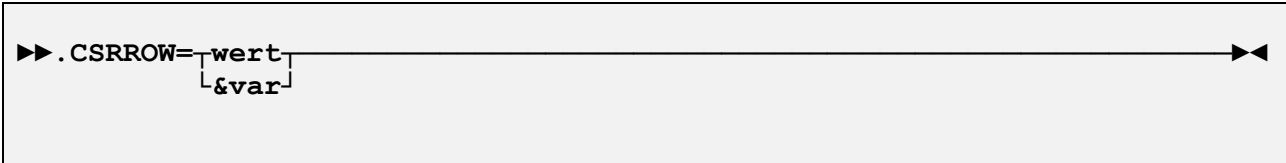
Wert | &var

liefert die Stelle, an der die Positionierung innerhalb des Feldes durchgeführt werden soll. Wird kein Wert angegeben, wird als Cursorposition die Stelle 1 angenommen (die erste Stelle des entsprechenden Feldes wird angesteuert).

Beispiel

`.CSRPOS=4`

Der Cursor wird im aktuellen Eingabefeld auf die vierte Stelle positioniert.

.CSRROW – ISPF Steuervariable Panel

wird in Verbindung mit Panels zur Tabellenanzeige genutzt und erlaubt die Zuweisung des CRP auf einen Tabellensatz des Model Sets, auf den der Cursor positioniert wird.

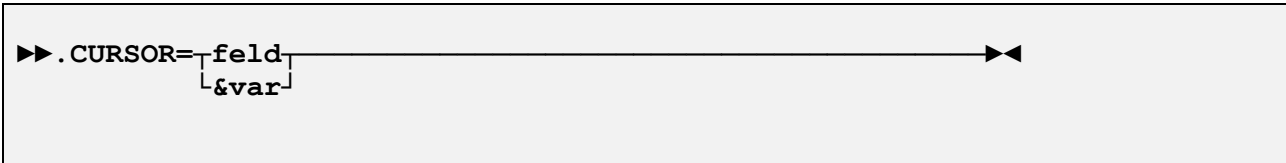
Ist kein entsprechender Satz in der Tabelle vorhanden, wird der Cursor in die Kommandozeile gestellt.

Die Steuervariable kann auch auf der rechten Seite einer Wertzuweisung genutzt werden.

Details können bei der Beschreibung des Kommandos TBDISPL im Kapitel 4 nachgelesen werden.

Wert | &var

liefert die interne Satznummer eines Satzes in der Tabelle, auf den der CRP gestellt werden soll.

.CURSOR – ISPF Steuervariable Panel

kann im)INIT- oder)REINIT-Teil zur Positionierung des Cursors abweichend vom Standard benutzt werden. Wird die .CURSOR-Variable im Panel gesetzt, werden eventuelle Angaben im DISPLAY oder TBDISPL Service ignoriert.

.CURSOR kann auch auf der rechten Seite einer Zuweisung genutzt werden.

Feld | &var

Name des Feldes, in das der Cursor positioniert werden soll. Wird eine Variable angegeben, muss sie den entsprechenden Feldnamen zum Inhalt haben.

Generelles zur Cursorpositionierung

- Wird .CURSOR nicht initialisiert oder BLANK gesetzt, erfolgt die Positionierung des Cursors auf das erste freie Feld nach ZCMD.

- Ist kein ZCMD-Feld kodiert, übernimmt das erste Eingabefeld dessen Funktion.
- Existiert kein freies Eingabefeld, wird der Cursor in das erste Eingabefeld des)BODY-Teils gestellt. Dies ist im Allgemeinen das Befehls-Eingabefeld.
- Verfügt die Bildschirmmaske über keine Eingabefelder, wird der Cursor in die linke obere Ecke des Bildschirms gestellt.

Der Cursor wird an den Beginn des zuletzt angesprochenen Feldes gestellt, wenn eine Nachricht aus einem der nachfolgenden Gründe ausgegeben werden soll:

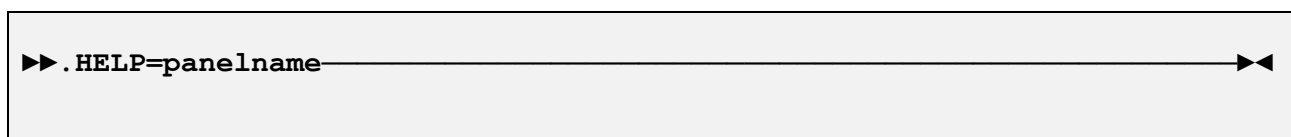
- Negatives Prüfergebnis einer VERIFY-Anweisung, in der eine Nachricht festgelegt wird.
- Im TRANS-Service wird angegeben ".MSG=....."
- .MSG wird explizit gesetzt.

Beispiele

```
&XYZ = TRANS (&A ... MSG=msgnr)
&A = TRANS (&XYZ ... MSG=msgnr)
VER (&XYZ,NB)
```

Vorausgesetzt, das Feld XYZ existiert im)Body, nicht aber das Feld oder die Variable A, wird der Cursor auf das Feld XYZ positioniert, wenn eine Nachricht auszugeben ist.

.HELP – ISPF Steuervariable Panel



erlaubt im)INIT-Teil des Panels festzulegen, welches HELP-Panel gezeigt bei HELPgezeigt wird.

Panelname

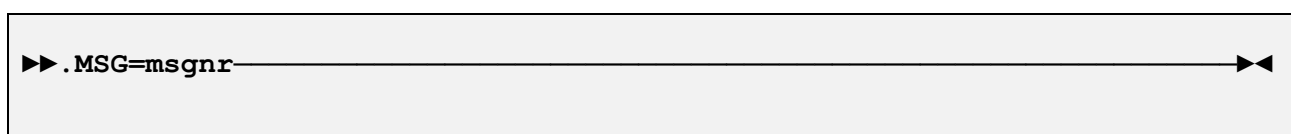
benennt das Panel (muss unter ISPLIB gefunden werden), das gezeigt werden soll.

Beispiel

```
.HELP = ISPTE
```

Ruft die TUTORIAL-Seite auf, wenn der Anwender das HELP-Kommando ausführt.

.MSG – ISPF Steuervariable Panel



erlaubt die Zuweisung einer Nachricht, die ausgegeben werden soll. Im Allgemeinen wird diese Steuervariable im)PROC-Teil des Panels definiert.

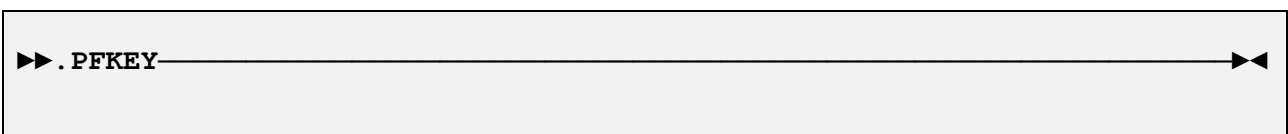
Msgnr

benennt die Nachricht, die ausgegeben werden soll. Der formale Aufbau der Nachricht kann im Kapitel 4 des Kochbuchs nachgelesen werden.

Beispiel

```
.MSG=FEHL001
```

Die entsprechende Nachricht wird ausgegeben.

.PFKEY – ISPF Steuervariable Panel

Diese Steuervariable kann einen der Werte PF01, PF02 bis PF24 zum Inhalt haben, sofern eine der Funktionstasten gedrückt wurde. Die Variable kann im)PROC-Teil des Panels abgefragt werden.

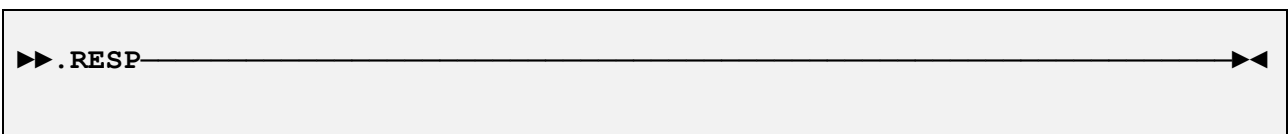
Soll die Information an die aufrufende Funktion übergeben werden, muss der jeweilige Inhalt in eine Dialogvariable im Function Pool übertragen werden.

Wird vom Benutzer keine Funktionstaste gedrückt, ist der Inhalt von .PFKEY BLANK, ebenso während des Durchlaufes der)INIT- und)REINIT-Section.

Beispiel

```
) PROC  
    &FKEY= . PFKEY  
    :
```

Der Inhalt der Steuervariablen wird in eine Dialogvariable übertragen. Diese Variable kann in der aufrufenden Funktion ausgelesen werden.

.RESP – ISPF Steuervariable Panel

Durch diese Variable kann generell festgelegt werden, mit welcher Bedieneraktion das normale oder abnorme Ende eines Panels angezeigt wird. Sie wird von ISPF standardgemäß auf "ENTER" gesetzt (das normale Ende eines Panels wird durch die Taste ENTER angezeigt). Meist wird diese Variable im)PROC-Teil ausgewertet, um noch im Panel feststellen zu können, wie das Ende der Panelanzeige herbeigeführt wurde.

Wird .RESP im)INIT- oder)REINIT-Teil gesetzt, zeigt es keine Auswirkung, wenn eine Meldung ausgegeben werden soll. Dies kann umgangen werden, indem die nachfolgenden Statements im)INIT- oder)REINIT-Teil kodiert werden:

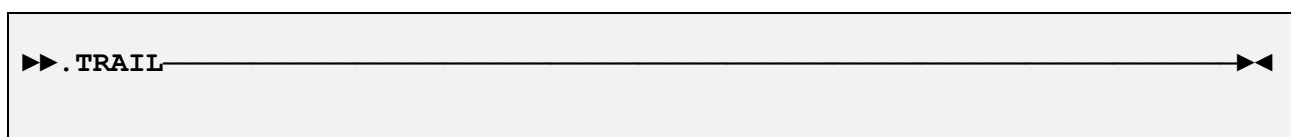
Beispiel

```

)INIT oder )REINIT
  IF (.MSG ^= &Z)
    .MSG=&Z
    .RESP=END

```

Die Variable kann im)PROC-Teil gesetzt werden, um ein END- oder ENTER-Ereignis zu erzwingen. Dies kann sehr hilfreich sein, wenn bei einer Prüfung ein Fehler erkannt wurde und das Panel auch dann neu angezeigt werden soll, wenn die Bedieneraktion END war.

.TRAIL – ISPF Steuervariable Panel

enthält den Rest einer Zeichenfolge, die durch die TRUNC-Anweisung abgeschnitten wurde, wobei das Trennzeichen nicht übernommen wird. Wird die TRUNC-Anweisung mit einer Längenangabe benutzt, liegen alle überzähligen Byte in .TRAIL.

Beispiel

```

)PROC
&TT=TRUNC (DATUM, ' ')
&REST=.TRAIL
&MM=TRUNC (&REST, ' ')
&JJ=.TRAIL
VER (&TT, RANGE, 1, 31, MSG=ERR001A) .CSRPOS=1
VER (&MM, RANGE, 1, 12, MSG=ERR001B) .CSRPOS=4

```

Ausgehend davon, dass im Panel eine Variable DATUM gefüllt wurde, wird diese Variable zerlegt, um eine Plausibilitätsprüfung durchführen zu können.

.TYPE – ISPF Steuervariable Panel

Wählt einen aus vier verschiedenen Nachrichtentypen aus.

Mit Angabe von .TYPE werden Definitionen aus .ALARM überschrieben.

Wird .TYPE=CRITICAL angegeben, ist dies gleichbedeutend mit .WINDOWS=RESP.

Die vier möglichen Angaben werden in nachfolgender Tabelle erläutert.

Type	Farbe	erscheint	Enter zwingend	Alarm
Notify	Weiß	Normal oder als Fenster	Nein	Nein
Warning	Gelb	Normal oder als Fenster	Nein	Nein
Action	Rot	Normal oder als Fenster	Nein	Ja
Critical	Rot	Immer als Fenster	Nein	Ja

.WINDOW – ISPF Steuervariable Panel

```

>>.WINDOW =--RESP-----><
          |--NORESP--
          |--LRESP--
          |--LNORESP--

```

Bewirkt, dass die Nachricht in einem POP-UP-Fenster angezeigt wird.

RESP

Sobald das Nachrichtenfenster erscheint, tritt das aktuelle Panel in den Hintergrund und wird erst durch drücken von ENTER wieder freigegeben.

NORESP

Das aktuelle Panel bleibt aktiv, auch wenn das Meldungsfenster geöffnet wird. Dateneingabe ist möglich, ohne vorher <ENTER> zu drücken.

LRESP

Wie RESP. gilt aber nur für die Long Message.

LNORESP

Wie NORESP. gilt aber nur für die Long Message.

.ZVARS – ISPF Steuervariable Panel

```

>>.ZVARS= ' (---,---) '-----><

```

wird im)INIT-Teil des Panels genutzt, um die Platzhalter-Variablen &Z den entsprechenden Dialogvariablen zuzuweisen. Für die Zuweisung wird der)BODY-Teil von oben links nach unten rechts zeilenweise nach Platzhaltern untersucht. Die gefundenen Platzhalter referieren auf die in der Klammer von links nach rechts angegebenen Variablen.

)ABC – ISPF Panel Section



Die)ABC-Section (ActionBarChoice) definiert einen Aktionsblock innerhalb einer ActionBar. Maximal 40)ABC-Bereiche je Panel sind möglich (je Block in der ActionBar ein)ABC-Bereich).

DESC(ab-text)

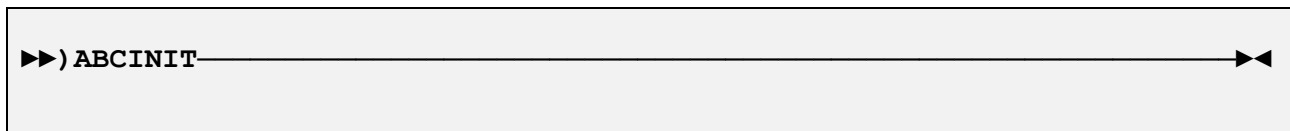
Name des Aktionsblockes. DESC() kann in einem)ABC-Bereich nur einmal kodiert werden und ist zwingend. Der Blockname muss gleichlautend im)BODY-Teil angegeben werden.

PDC DESC(auswahltext)

Wird mit ENTER in der Action Bar ausgewählt, erscheint ein Pull-Down-Menü, in dem weitere Auswahlmöglichkeiten stehen. Alle Auswahlpunkte im PullDownMenü werden in der gleichen)ABC-Section definiert, wie ihr übergeordneter Auswahlblock.

Das Ende des)ABC-Teils wird erkannt, sobald die nächste Panel-Steueranweisung (schließende Klammer ") in Spalte 1) auftritt.

)ABCINIT – ISPF Panel Section



Jedem)ABC-Teil muss eine)ABCINIT-Section folgen (Action Bar Initialization Section.)ABCINIT verhält sich zu)ABC wie)INIT zu)BODY, ist aber zwingend).

Im)ABCINIT-Teil muss die Verbindung zwischen dem Namen des Aktionsblockes und der Variablen für die Auswahl im Pull-Down-Menü über .ZVARS=variable hergestellt werden. Diese Variable enthält nach ENTER eine Zahl, die der Auswahl im gewählten Pull-Down-Menü entspricht. Der Inhalt der Variablen kann im zugehörigen)ABCPROC-Teil überprüft werden.

Das Ende des)ABCINIT-Teils wird erkannt, sobald die nächste Steueranweisung (schließende Klammer ") in Spalte 1) auftritt.

)ABCPROC – ISPF Panel Section

```
▶▶)ABCPROC_____▶◀
```

ist wahlfrei. Diese Section erfüllt den gleichen Zweck, wie die)PROC-Section für ein gewöhnliches Panel. Die Ausführung erfolgt, sobald eine Wahl im Pull-Down getroffen wurde.

Sinnvoll ist die Kodierung des)ABCPROC-Teiles in jedem Fall, da häufig bestimmte Aktionen nach Auswahl im Pull-Down-Menü direkt im Panel durchgeführt werden können.

Das Ende des)ABCPROC-Teils wird erkannt, sobald die nächste Panel-Steueranweisung (schließende Klammer ") in Spalte 1) auftritt.

)AREA – ISPF Panel Section

```
▶▶)AREA bereichsname_____▶◀
```

Beschreibt einen Bereich im)BODY-Teil des Panels. Der Bereich kann größer als der dafür vorgesehene abzubildende Ausschnitt sein. Lässt sich der Bereich nicht in seiner ganzen Größe zeigen, kann innerhalb des Ausschnittfensters mit den Funktionen UP und DOWN geblättert werden.

Als Hinweis über eventuell im)BODY nicht sichtbare Teile des)AREA-Bereiches wird in der obersten Zeile ein SCROLL-Indikator gezeigt. Der Indikator kann folgende Aussagen treffen:

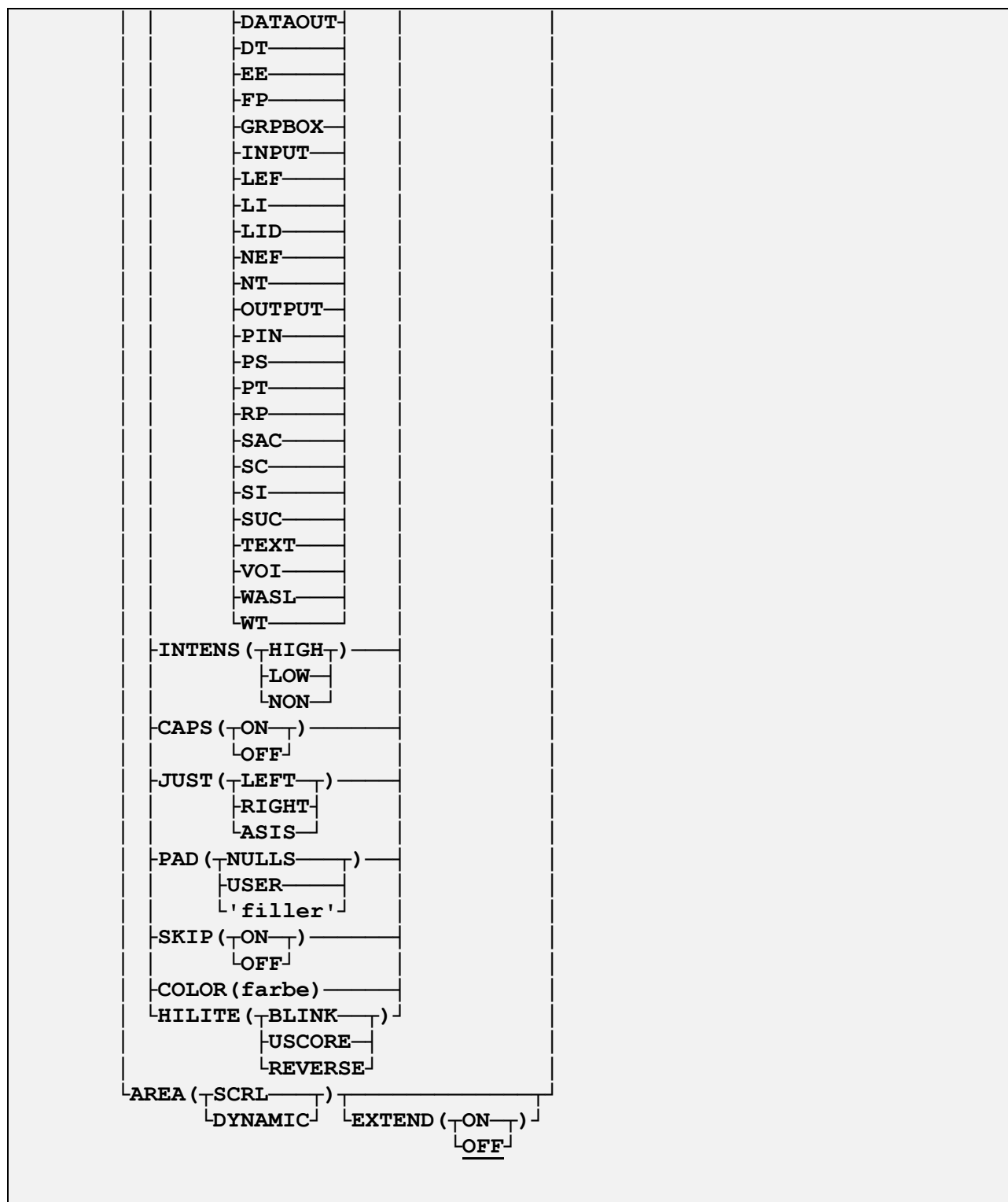
```
MORE:      +      blättern nach unten möglich
MORE:      -      blättern nach oben möglich
MORE:      +-     blättern nach unten und oben möglich
```

)ATTR – ISPF Panel Section

```
▶▶)ATTR_____▶◀
    [DEFAULT (x1 x2 x3)]
```

```

  |
  |  yn  TYPE ( AB
  |      ABSL
  |      CEF
  |      CH
  |      CHAR
  |      Ct
  |      DATAIN
  |
  |_____▶◀
```



Bestimmt die Feld-Eigenschaften. Die Operanden können entfallen, wenn lediglich die Standardattribute mit den vorgesehenen Zeichen genutzt werden. Die Angaben haben folgende Bedeutung:

DEFAULT(x1 x2 x3)

Muss kodiert werden, wenn eines oder mehrere der Standardattribute umdefiniert werden sollen. Die Angaben in der Klammer treffen folgende Aussage:

- x1 Textfeld intensiv (Standard:%)
- x2 Textfeld normal (Standard: +)
- x3 Eingabefeld intensiv (Standard: _)

yn

ist das Attributzeichen, das stellvertretend für alle nachstehenden Attribute in der Bildschirmmaske kodiert wird. Generell kann hier auch eine HEX Angabe erfolgen. Davon ist, wenn dies verhinderbar ist, abzuraten, weil man die Steuerzeichen im)BODY dann nicht sehen kann.

Ein abschreckendes Beispiel ist das ISR@PRIM (ISPF/PDF Primary Option Menu). Lesen und Stauen.

AREA

Legt einen Bereich innerhalb des Panels fest. Die beiden gängigsten Bereichstypen sind:

Areatyp	Bedeutung
SCRL	Der Bereich ist über die Scrolling UP/ DOWN ansteuerbar. Sind nicht alle Daten im sichtbaren Ausschnitt des Bereiches abbildbar, ist blättern möglich.
DYNAMIC	Bereich, dessen Inhalt über eine dynamisch aufgebaute Variable gezeigt werden kann. Ist der Inhalt der zu zeigenden Variablen länger als eine Zeile des dynamischen Bereiches, erfolgt der Zeilenumbruch automatisch.

EXTEND()

bestimmt, ob die Größe des Bereiches der Definition im)BODY-Teil entsprechen soll (OFF), oder ob sich der Bereich im Bedarfsfall selbstständig bis zum unteren Panelrand ausdehnen kann (ON).

TYPE

Bestimmt die Art des Feldes. Die folgende Tabelle benennt alle Attributstypen. Sowohl die klassischen (old style) als auch jene, die erst seit der CUA-Konvention möglich sind. Da die Möglichkeiten im Einzelnen sehr umfangreich sind, bemüht gegebenenfalls das Manual "ISPF Dialog Developer's Guide".

Typ	Bedeutung
AB	Action Bar unselected Choice
ABSL	Action Bar Separator Line
CEF	Choice Entry Field
CH	Column Heading
CHAR	Steuerzeichen bei Shaddow-Variablen innerhalb einer Dynamic Area
CT	Caution Text
DATAIN	Eingabefeld. Festgelegte Attributzeichen werden nicht als Bestandteil der Daten interpretiert, sondern entsprechend ihrer Bedeutung umgesetzt.
DATAOUT	Ausgabefeld. Sind Steuerzeichen enthalten, werden sie nicht als Daten gezeigt, sondern entsprechend umgesetzt (Farbansteuerung, Reverse, etc.)
EE	Error emphasis
ET	Emphasized Text
FP	Field Prompt
GRPBOX	Group Box
INPUT	Beschreibbares Eingabefeld. Die Feldlänge ist abhängig von der Länge des Feldnamens und/oder dem nächsten nachfolgenden Attributzeichen

LEF	List Entry Field
NEF	Normal Entry Field
NT	Normal Text
OUTPUT	Ausgabefeld. Die Länge ist abhängig vom Feldnamen und/oder dem nächsten nachfolgenden Attribut
PIN	Panel Instruction
PT	Panel Title
RP	Reference Phrase
SAC	Select Available Choice
SC	Selected Choice
SI	Scroll Information
SUC	Select Unavailable Choices
TEXT	für wahlfreien Text. Enthält der Text Variable, werden sie in ihrer absoluten inhaltlichen Länge ausgegeben.
VOI	Variable Output Information
WASL	Work Area Separator Line
WT	Warning Text

INTENS

Steuert die Intensität der Darstellung am Bildschirm. Heute wird LOW und HIGH von den meisten Emulationen nicht mehr unterstützt. Die bessere Wahl ist die Angabe COLOR()

INTENS	Bedeutung
HIGH	Die Anzeige erfolgt in maximaler Helligkeit
LOW	Darstellung in Normal
NON	Verdunkeltes Feld (Schwarz auf Schwarz für Password)

CAPS

Umsetzung in Großschreibung (wie CAPS im EDIT). Mögliche Angaben sind:

CAPS	Bedeutung
ON	Bei <ENTER> erfolgt die Umsetzung in UpperCase
OFF	Groß-/Kleinschreibung bleibt erhalten

JUST

Steuert die Ausrichtung der Daten. Die möglichen Angaben bedeuten:

Justify	Bedeutung
LEFT	Linksbündig
RIGHT	Rechtsbündig
ASIS	Übernahme ohne Ausrichtung

PAD

Bestimmt Füllzeichen für Ein- oder Ausgabefelder. Mögliche Angaben sind:

PAD	Bedeutung

NULLS	Füllen mit binären Nullen
USER	Entsprechend den Terminalattributen in ISPF
Filler	Benutzerdefiniertes Füllzeichen

PADC

Wie PAD aber Conditional (abhängig). Leere Felder werden mit Füllzeichen vorbelegt. Vorteil: in gefüllten Feldern ist Einfügen möglich, leere Felder werden durch Füllzeichen in ihrer Länge sichtbar.

SKIP

steuert, dass bei Dateneingabe am Ende eines INPUT-Feldes Text- und Ausgabefelder vom Cursor übersprungen werden.

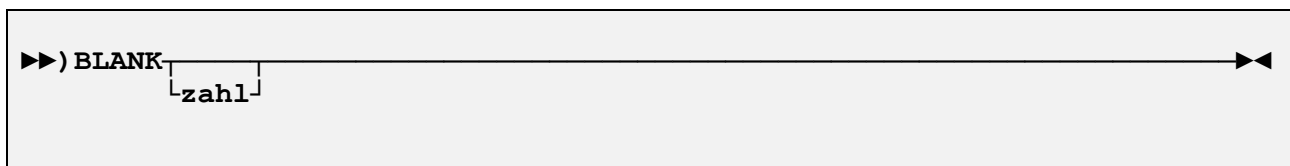
COLOR

legt die Farbe fest, in der ein Feld angezeigt werden soll (siehe auch SETTINGS). Möglichen Farben: RED, GREEN, BLUE, TURC, PINK, YELLOW, WHITE

HILITE

Erlaubt abhängig vom Terminaltyp folgende Aussagen:

HILITE	Bedeutung
BLINK	Lässt den Text blinken (sofern vom Terminaltyp unterstützt)
USCORE	Der Text wird unterstrichen
REVERSE	Vordergrund- und Hintergrundfarbe werden vertauscht

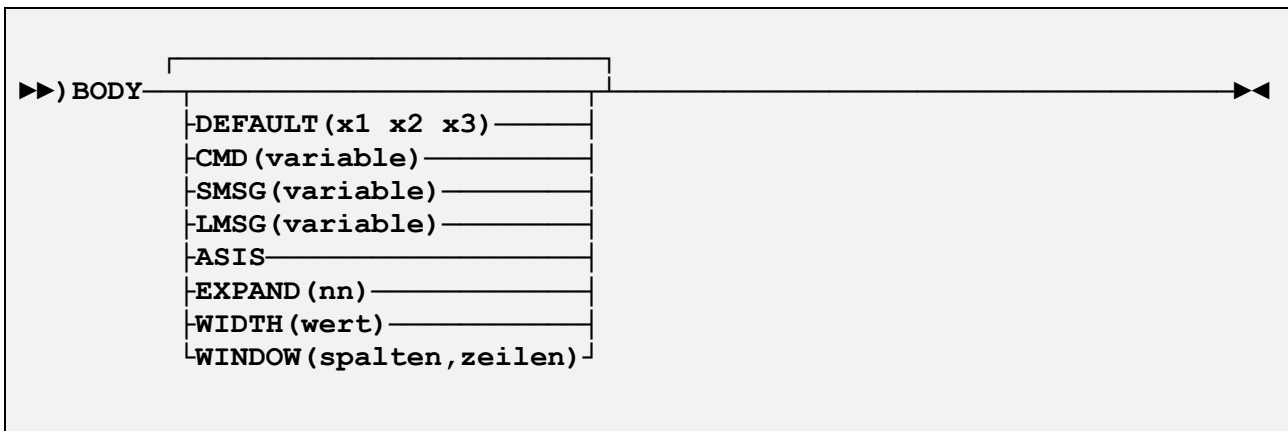
)BLANK – ISPF File Tailoring Steueranweisung

Erzeugt eine Leerzeile bei der Ausgabe des File Tailoring Service.

zahl

bestimmt, wie viele Leerzeilen erzeugt werden sollen. Wird kein Wert angegeben, wird eins angenommen. Der Wert kann durch eine Variable übergeben werden.

)BODY – ISPF Panel Section



Beschreibt den am Bildschirm sichtbaren Teil des Panels.

DEFAULT(x1 x2 x3)

Muss kodiert werden, wenn eines oder mehrere der Standardattribute umdefiniert werden sollen. Die Angaben in der Klammer treffen folgende Aussage:

- x1 Textfeld intensiv (Standard:%)
- x2 Textfeld normal (Standard:+)
- x3 Eingabefeld intensiv (Standard:_)

CMD() SMSG() LMSG()

definiert Variablen, die ZCMD, die Short und Long Message in sich trägt. Dies ist nur notwendig, wenn die Standorte dieser Felder vom allgemein üblichen Standard abweichen sollen. Warum?

ASIS

regelt, dass die Felder CMD und LMSG an der Stelle des Panels gezeigt werden, an der sie kodiert sind.

EXPAND(nn)

zwei Zeichen, die für das Ausrichten von Texten oder Feldern genutzt werden. Dazwischen kann ein beliebiges Füllzeichen stehen.

Beispiel

```

)BODY EXPAND(//)
  +/*/ Zentrierte Überschrift /*/
  
```

WIDTH(wert)

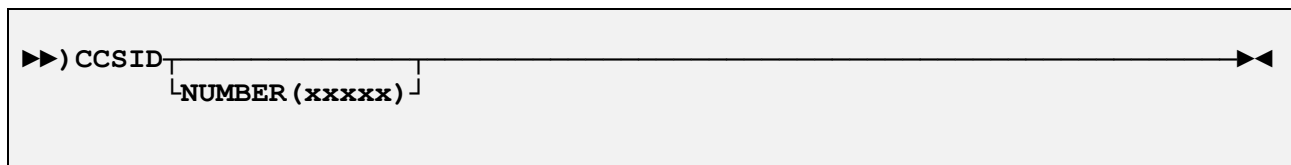
Legt die Spaltenbreite für Terminals fest, deren Zeilenlänge 80 überschreitet, sofern diese Breite genutzt werden soll.

WINDOW(spalten,zeilen)

Definiert die Größe des POP-UP-Fensters. Die Werte in der Klammer müssen mindestens der effektiven Größe des Panels entsprechen. Die Angabe wird ignoriert, wenn das Panel aus REXX ohne ADDPOP aufgerufen wird. Im umgekehrten Fall (keine WINDOW-Angabe im Panel, Aufruf mit ADDPOP-Service) wird das Panel normal dargestellt bzw. in das letzte durch ADDPOP definierte Fenster gesetzt.

Beispiel

```
)BODY WINDOW(40,2)
+Erste Textzeile in diesem Beispiel.
+Zweite und letzte Zeile.
)END
```

)CCSID – ISPF Panel Section

Für die Verarbeitung von Codepages.

NUMBER(xxxxx)

Was hier angegeben werden kann ist in Band-1 beschrieben. Zu finden auch im Manual "ISPF Dialog Developers Guide".

Folgende CCSID-Angaben sind möglich:

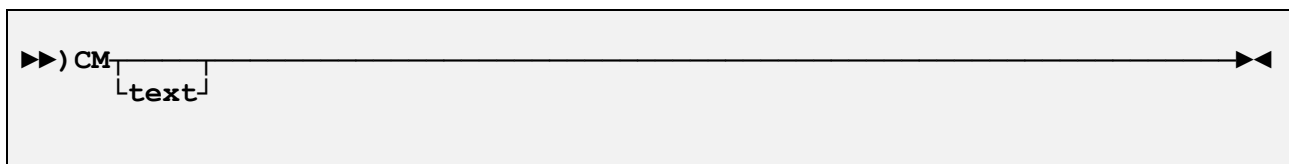
CCSID	Character-Set	Code-page	Extended CCSID1 Unterstützung für Länder
00037	679	37	USA, Kanada, Niederlande, Portugal, Brasilien, Australien, Neuseeland
00273	697	273	Deutschland, Österreich
00277	697	277	Dänemark, Norwegen
00278	697	278	Finnland, Schweden
00280	697	280	Italien
00284	697	284	Spanien, Latein Amerika (spanisch)
00285	697	285	Vereinigtes Königreich
00297	697	297	Frankreich
00420	235	420	Arabisch
00424	941	424	Hebräisch
00500	697	500	Schweiz, Belgien
00838	1176	838	Thailand
00870	959	870	Latin-2
00871	697	872	Island
00875	923	875	Griechenland
00880	960	880	Kyrillisch

01025	1150	1025	Kyrillisch
01026	1026	1026	Türkei
01047	697	1047	Latin-1
01123	1326	1123	Ukraine

CCSID	Character-Set	Code-page	Extended CCSID2 Unterstützung (EURO) für Länder
00924	1353	0924	Latin9
01140	659	1140	USA, Kanada, Niederlande, Portugal, Brasilien, Australien, Neuseeland
01141	659	1141	Österreich, Deutschland
01142	695	1142	Dänemark, Norwegen
01143	695	1143	Finnland, Schweden
01144	695	1144	Italien
01145	659	1145	Spanien, Latein Amerika (spanisch)
01146	659	1146	Vereinigtes Königreich
01147	659	1147	Frankreich
01148	659	1148	Schweiz, Belgien
01149	659	1149	Island
01153	1375	1153	Latin2
01154	1381	1154	Kyrillisch
01155	1378	1155	Türkei
01158	1388	1158	Ukraine
01160	1395	1160	Thailand
04899	1356	0803	Hebräisch
04971	1371	0875	Griechenland
12712	1357	0424	Hebräisch
16804	1461	0420	Arabisch

Auf die Angaben für traditionell Chinesisch und dergleichen möchte ich verzichten

)CM – ISPF File Tailoring Steueranweisung

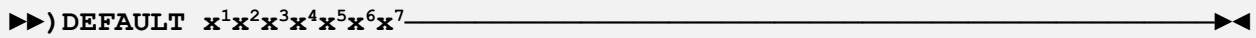


ist eine nicht zu übernehmende Kommentarzeile in einer Skelettdatei des File Tailoring Service. Die Nutzung ist sinnvoll, um Hinweise oder Update History im Skelett zu pflegen.

text

kann als Kommentar angegeben werden

)DEFAULT – ISPF File Tailoring Steueranweisung



Wird eingesetzt, um Standard-Steuerzeichen in einer Skelett-Datei zu verändern. Die Wirkungsweise ist vergleichbar wie in den gleichnamigen Angaben im)ATTR- oder)BODY-Teil eines Panels. Soll auch nur ein Zeichen verändert werden, müssen immer alle sieben angegeben sein. Die Standardwerte sind:)DEFAULT)&?!<|>

x¹

Kennzeichnet die FT-Steueranweisung

x²

Kennzeichnet eine Variable

x³

Fortsetzungszeichen

x⁴

Tabulatorsprung

x⁵

Beginn der ODER-Übertragung

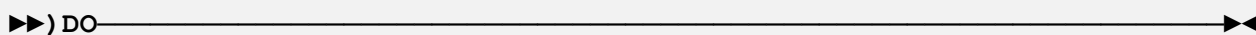
x⁶

Trennzeichen der ODER-Übertragung

x⁷

Ende der ODER-Übertragung

)DO – ISPF File Tailoring Steueranweisung



Leitet im File Tailoring eine Befehlsfolge ein.

Beispiel

```
)IF bedingung THEN )DO  
)CM was auch immer hier geschieht  
)ENDDO  
)ELSE )DO  
)CM hier geschieht etwas anderes  
)ENDDO
```

)DO wert – ISPF File Tailoring Steueranweisung

The diagram shows the syntax `>> )DO wert` followed by a horizontal line with arrowheads at both ends, representing the continuation of the code block.

Leitet eine Verarbeitungsschleife ein.

wert

ist die Anzahl der Schleifendurchläufe

Beispiel

```
)IF bedingung THEN )DO 5  
)CM was auch immer hier geschieht  
)ENDDO
```

)DO – Iteration – ISPF File Tailoring Steueranweisung

The diagram shows the syntax `>> )DO var=wert TO endwert` followed by a horizontal line with arrowheads at both ends. Below the line, the options `BY increment` and `FOR cnt` are shown in brackets, indicating they are optional parts of the syntax.

Zählschleife

var=wert

Initialisierung der Steuervariablen nur beim ersten Mal

TO endwert

Überragt die Steuervariable den Endwert, wird die Schleife verlassen

BY increment

Steuert die Schrittweite der Steuervariablen und kann eine Positive oder negative ganze Zahl sein.

FOR cnt

Regelt, wie viele Male die Steuervariable verändert werden soll

Beispiel

```
)DO I=1 TO 10  
)CM was auch immer hier geschieht  
)ENDDO
```

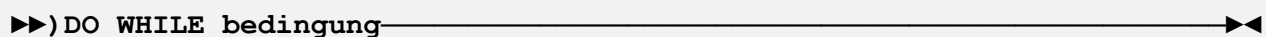
)DO FOREVER – ISPF File Tailoring Steueranweisung

The diagram shows the syntax for a)DO FOREVER loop. It starts with ')>>DO FOREVER' followed by a horizontal line with a double-headed arrow at the end, indicating a continuous loop.

Ist eine Endlosschleife, welche mit)LEAVE verlassen wird

Beispiel

```
)DO FOREVER  
:  
)IF &RC ^= 0 THEN )LEAVE  
:  
)ENDDO
```

)DO WHILE – ISPF File Tailoring Steueranweisung

The diagram shows the syntax for a)DO WHILE loop. It starts with ')>>DO WHILE bedingung' followed by a horizontal line with a double-headed arrow at the end, indicating a loop that continues as long as the condition is true.

Preabweisende Schleife im File Tailoring

Beispiel

```
)SET RC = 0  
)DO WHILE RC = 0  
:  
)ENDDO
```

)DO UNTIL – ISPF File Tailoring Steueranweisung

```
▶▶)DO UNTIL bedingung
```

Postabweisende Schleife im File Tailoring

Beispiel

```
)DO UNTIL RC > 0  
:  
)ENDDO
```

)DOT – ISPF File Tailoring Steueranweisung

```
▶▶)DOT tablename
```

Kennzeichnet den Beginn einer Verarbeitungsschleife, innerhalb derer Tabellendaten eingearbeitet werden sollen. Das Ende der Schleife wird durch)ENDDOT markiert.

)ELSE – ISPF File Tailoring Steueranweisung

```
▶▶)ELSE
```

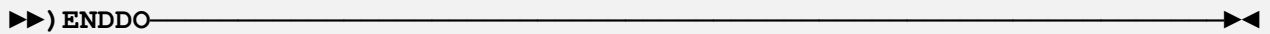
Ist der NEIN-Zweig einer)IF-Abfrage im File Tailoring.

)END – ISPF Panel Section

```
▶▶)END
```

Kennzeichnet das Ende einer Bildschirmmaske. Wird dieses Statement angesprungen, wird die Steuerung an die aufrufende Funktion zurückgegeben.

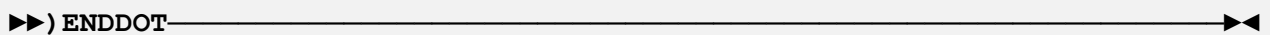
)ENDDO – ISPF File Tailoring Steueranweisung

A horizontal line with a double arrow pointing right at the start and a double arrow pointing left at the end. The text ') ENDDO' is positioned at the start of the line.

►►) ENDDO◄◄

Kennzeichnet das Ende einer Befehlsfolge im File Tailoring, welche durch)DO eingeleitet wurde

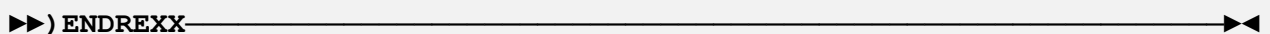
)ENDDOT – ISPF File Tailoring Steueranweisung

A horizontal line with a double arrow pointing right at the start and a double arrow pointing left at the end. The text ') ENDDOT' is positioned at the start of the line.

►►) ENDDOT◄◄

Kennzeichnet das Ende einer Verarbeitungsschleife, innerhalb derer Tabellendaten eingearbeitet werden sollen. Der Beginn der Verarbeitung aller Sätze wird durch)DOT markiert.

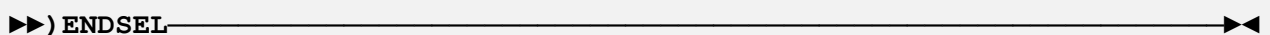
)ENDREXX – ISPF File Tailoring Steueranweisung

A horizontal line with a double arrow pointing right at the start and a double arrow pointing left at the end. The text ') ENDREXX' is positioned at the start of the line.

►►) ENDREXX◄◄

Kennzeichnet das Ende von REXX-Code im Skeleton. Der Beginn der REXX-Kodierung wird durch)REXX markiert.

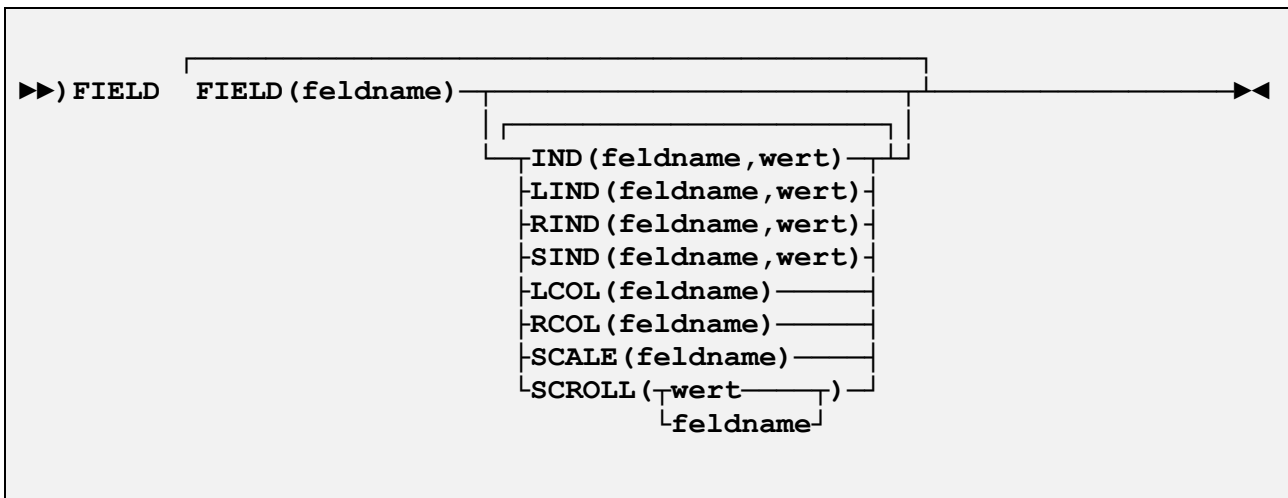
)ENDSEL – ISPF File Tailoring Steueranweisung

A horizontal line with a double arrow pointing right at the start and a double arrow pointing left at the end. The text ') ENDSEL' is positioned at the start of the line.

►►) ENDSEL◄◄

Kennzeichnet das Ende einer Verarbeitungsfolge, die abhängig von einer Bedingung ist. Die Überprüfung der Bedingung erfolgt durch)SEL.

)FIELD – ISPF Panel Section



Erlaubt die Links- und Rechtsverschiebung eines Feldes in einem Panel, sowie die Anzeige über ein PopUp-Window.

Jeder Eintrag der)FIELD-Sektion muss mit FIELD beginnen.

Alle Variablen aller Parameter müssen eindeutig sein.

Feld-Scroll-Unterstützung Panel-spezifisch. Ein Folgepanel, das die gleichen Felder benutzt (nicht scrollable definiert), kann zu Datenverlust führen.

FIELD(feldname)

Feld, für welches die Definitionen erfolgen.

IND(feldname, wert)

Left/Right-Indikator. Feldname muss auf ein 2-Byte-Feld referenzieren, das in der Panelanzeige entsprechend aktualisiert wird.

Paneldefinition:

```

)ATTR
  # TYPE (OUTPUT)
)BODY
+Scroll-Feld: _SCRLFLD          #SCRIND+
)FIELD
  FIELD(SCRLFLD) IND('<>') /* ersetzt +- mit <> */

```

Panelanzeige:

```
Scroll-Feld: EFGHIJKLMNOPQRS <>
```

LIND() | RIND()

Wie IND() aber nur als LEFT-Indicator oder Right-Indicator.

SIND(feldname,wert)

Separator Scroll Indicator Variable.

Paneldefinition:

```
)ATTR
  # TYPE (OUTPUT)
)BODY
+Separator:  #SEPIND
+Scroll-Feld: _SCRLFLD          +SCRIND+
)FIELD
  FIELD (SCRLFLD)  SIND (SEPIND)
```

Panelanzeige:

```
Separator   : <----->
Scroll-Feld: EFGHIJKLMNOPQRS
```

LCOL(feldname) | RCOL(feldname)

Feld, in dem die Startposition für linke Spalte oder rechte Spalte angegeben ist. Das Feld wird vom Scrolling entsprechend aktualisiert.

SCALE(feldname)

Scale Indicator Feld. An der entsprechenden Stelle im Panel wird ein Zeilenlineal angezeigt.

Paneldefinition:

```
)ATTR
  # TYPE (OUTPUT)
)BODY
+Spaltenanzeige: #SCLIND
+Scroll-Feld    : _SCRLFLD          #SCRIND+
)FIELD
  FIELD (SCRLFLD)  SCALE (SCLIND)
```

Panelanzeige:

```
Spaltenanzeige: --+----1----+----2
Scroll-Feld    : CDEFGHIJKLMNOPQRST
```

SCROLL(wert feldname)

Scroll-Steuerfeld. Die Inhalte können ON oder OFF sein. Dementsprechend ist das Scrolling möglich oder nicht.

Primärbefehle für Scroll-Felder

Wenn der Cursor in einem Scrollfeld steht, sind folgende Befehle möglich

Scrollwert	Bedeutung
LEFT	Blättert nach links
RIGHT	Blättert nach rechts

ZEXPAND	Zeigt den Feldinhalt in einem dynamischen PopUp-Window an. Sofern das Feld vom Typ INPUT ist, kann es im Window auch editiert werden. UP/DOWN ermöglicht Scrolling HEX ON/OFF ändert die Darstellung
ZCSFFLD	Setzt den Inhalt des Scrollfeldes auf Blank
Wird ein Scrollfeld gefunden und darin geblättert, wird der definierte Scrollamount berücksichtigt. Die Wirkungsweise ist dabei folgende:	
PAGE	Ist äquivalent zur Feldlänge
DATA	Ist wie Feldlänge -1
HALF	Halbe Feldlänge
CSR	Cursorabhängig
M in der Commandline	Blättert an das linke oder rechte Feldende

Beispiele für)FIELD-Anwendung

Paneldefinition:

```

)ATTR
  # TYPE(OUTPUT) CAPS(OFF) JUST(ASIS)
)BODY expand(\\)
%----- Links/Rechts Scrolling \- \
%Command_ZCMD
+
+
+Feld           Wert           Scroll
+Scrollfeld     :_SCRLFLD      #SFIND
+Links/Rechts   :#SFLIND       #SFRIND
+Linke Spalte   :_SFLCOL
+Rechte Spalte  :_SFRCOL
+Länge          :_SFLEN
)INIT
  .CURSOR = ZCMD
)FIELD
  FIELD(SCRLFLD)
    LEN(SFLEN)
    LCOL(SFLCOL)
    RCOL(SFRCOL)
    IND(SFIND)
    LIND(SFLIND)
    RIND(SFRIND)
    SCROLL(SFCTL)
)END

```

REXX für die Anzeige:

```

/* REXX * Beispiel Scroll-Felder */
SFTTL = arg(1)
SCRLFLD = "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ0123456789"
SFLCOL = 5
SFLEN = 200
Do until RC > 4
    Address ispxec "display panel(SCRLDEMO)"
End
Exit

```

Ergebnis:

```

----- Links/Rechts Scrolling -----
Command ==>

Feld           Wert           Scroll
Wert           : EFGHIJKLMNOPQRSTU -+
Links/Rechts   : -             +
Linke Spalte   : 5
Rechte Spalte  : 22
Länge          : 200

```

Änderungen der Scroll-Indicator im Panel:

```

) FIELD
    FIELD(SCRLFLD)
    LEN(SFLEN)
    LCOL(SFLCOL)
    RCOL(SFRCOL)
    IND(SIND, '<>')
    LIND(SFLIND, '<')
    RIND(SFRIND, '>')
    SCROLL(SFCTL)
) END

```

Erzeugt das folgende Ergebnis:

```

----- Links/Rechts Scrolling -----
Command ==>

Feld           Wert           Scroll
Scrollfeld     : EFGHIJKLMNOPQRSTU <>
Links/Rechts   : <             >
Linke Spalte   : 5
Rechte Spalte  : 22
Länge          : 200

```

Unterstellen wir nun, dass hinter einer PF-Taste der Befehl ZEXPAND liegt und diese Taste gedrückt wird, während der Cursor im Scrollfeld steht, wird ein PopUp-Window aufgerufen, in dem das gesamte Feld in seiner Länge von 200 Byte editierbar wird.

)HELP – ISPF Panel Section

```
►►)HELP FIELD(feldname) PANEL(help-panel-name)◄◄
```

In diesem Bereich werden HELP-Panels definiert. Im Allgemeinen enthalten HELP-Panels nur erklärenden Text. Grundsätzlich kann ein HELP-Panel auch als Fenster gezeigt werden.

HELP-Panels, die im)HELP-Teil definiert werden, beziehen sich auf Felder im Ursprungspanel. Je Feld, für das Hilfe über ein HELP-Panel angeboten werden soll, muss ein Eintrag im)HELP-Teil erfolgen.

FIELD(feldname)

ist das Bezugsfeld im Ursprungspanel.

PANEL(help-panel-name)

Name des Hilfepanels.

Hinweis

Die)Help-Section ist zwischen der)PROC- und)END-Section anzulegen.

Beispiel

)HELP

```
FIELD(feldname) PANEL(hpname)
```

)IF – ISPF File Tailoring Steueranweisung

```
►►)IF bedingung THEN◄◄
```

Ermöglicht in Skeletons Abfragen komfortabler als)SEL, da im)IF Ja-/Nein-Zweige kodiert werden können.

Beispiel

```
)IF var = &Z THEN )DO
)CM Hier liegen die Aktionen für den JA-Fall
)ENDDO
)ELSE )DO
)CM und hier alles für den NEIN-Fall
)ENDDO
```

)INEXIT – ISPF Panel Section

```

>>> INEXIT PGM exit-adresse
          LOAD programmname CACHE

```

Ist eher selten zu sehen. Damit könnte jede Panelzeile vor Verarbeitung an eine Exitroutine übergeben werden.

PGM exit-adresse

Definiert, dass ein Programm, das bei Start von ISPF geladen wurde, gestartet wird. Die exit-adresse wird an ISPF übergeben (ist ein 4 Byte-Feld und kann auch oberhalb der 16MB-Grenze liegen).

LOAD programmname

Das Programm wird dynamisch geladen

)ITERATE – ISPF File Tailoring Steueranweisung

```

>>> ITERATE

```

Ermöglicht in Skeletons für einen Schleifendurchlauf die Befehlsausführung zu unterdrücken. Subjektiv führt)ITERATE einen Rücksprung zum Schleifenkopf durch

Beispiel

```

)DO FOTEVER
:
)IF bedingung THEN )ITERATE
:
)ENDDO

```

)IM – ISPF File Tailoring Steueranweisung

```

>>> IM skelname
          NT OPT EXT
          NOEXT

```

Erlaubt das Einbetten weiterer Skelettdateien. Auf diesem Weg können Skeletons bis zu 15 Ebenen tief ineinander verschachtelt werden.

NT

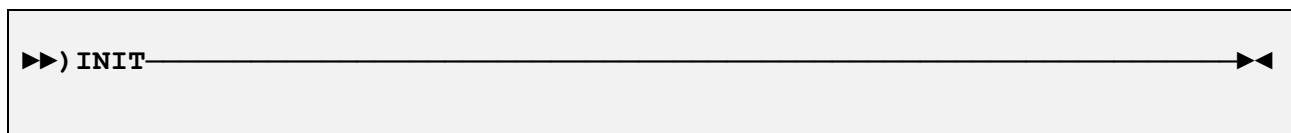
Das Skeleton wird ohne Auflösung von Variablen und Funktionen eingebettet.

OPT

Die Einbettung ist optional, Das Skeleton muss in ISPSLIB nicht vorhanden sein. Ohne OPT und fehlendem Skeleton wird ein schwerer Fehler erzeugt.

EXT | NOEXT

Erlaubt (oder nicht) die Nutzung von Built-in-Functions im Skeleton. Ist keiner der beiden Parameter angegeben, entscheidet die Angabe bei der Installation (i.A. ja).

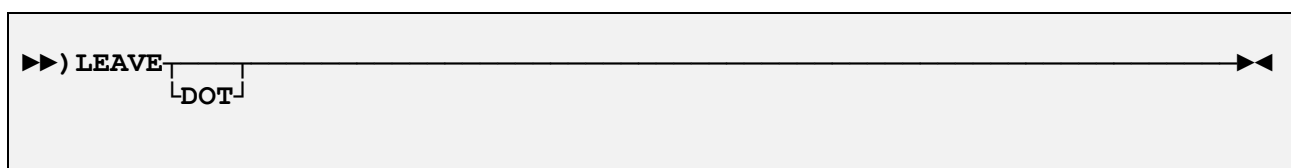
)INIT – ISPF Panel Section

Beschreibt den Initialisierungsteil eines Panels. Hier können Statements wie:

- IF - ELSE (zur Ablaufsteuerung)
- GOTO (zur Ablaufsteuerung)
- EXIT (zur Ablaufsteuerung)
- REFRESH (zur Feldauffrischung)
- VER (zur Feldüberprüfung)
- &var = wert (zur Wertzuweisung)
- VGET (Variable aus POOL lesen)
- VPUT (Variable in POOL schreiben)

abgesetzt werden. Die Möglichkeiten im Detail können bei den jeweiligen Instruktionen nachgelesen werden.

Das Ende des)INIT-Teils wird erkannt, sobald die nächste Panel-Steueranweisung (schließende Klammer ") in Spalte 1) auftritt.

)LEAVE – ISPF File Tailoring Steueranweisung

Absprung aus der aktiven Schleife, in der die Instruktion steht.

DOT

Springt aus der aktiven)DOT-Schleife

Beispiel

```
)DOT ADRTAB
:
)IF bedingung THEN )LEAVE DOT
:
)ENDDOT
```

)LIST – ISPF Panel Section

```
►►)LIST name
    VAL(wert1) CHOICE(auswahl1)
    VAL(wert2) CHOICE(auswahl2)
    VAL(wertn) CHOICE(auswahln)
```

Kann benutzt werden, um eine Auswahlliste zu erzeugen und muss in Verbindung mit einem Eintrag in der)ATTR Section gesehen werden.

Name

Ist der Name der Liste. Dieser muss zu LISTBOX(name), DDLIST(name) oder COMBO(name) eines INPUT-Feldes in der)ATTR Section korrespondieren.

VAL(wert)

Wird nur für Boxen und DropDown Listen benötigt. Die Wertangabe ist auf drei Byte beschränkt und wird, sofern die Angabe länger ist, ohne Hinweis abgeschnitten.

CHOICE(auswahl)

Der Wert kann über eine Variable oder einen String festgelegt werden. Besteht der String aus mehr als einem Wort, muss er zwischen Apostrophen stehen. Die Länge der Angabe ist auf 99 Byte begrenzt und wird, wenn sie länger ist, ohne Hinweis abgeschnitten.

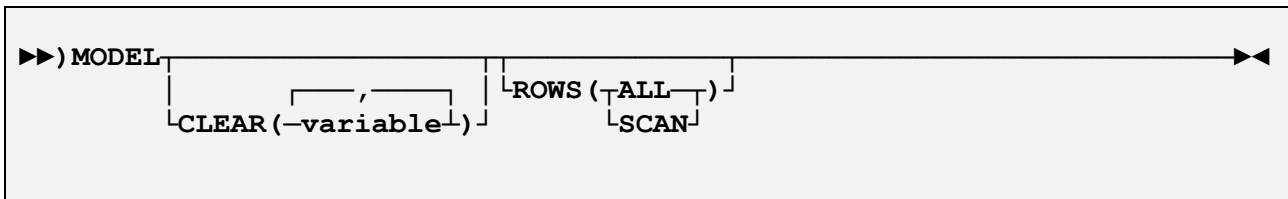
Beispiel

```
)ATTR
  2 TYPE(CEF) DDLIST(name) CSRGRP(x) WIDTH(w) DEPTH(d)
  3 TYPE(SAC) DDLIST(ON) CSRGRP(x)
:
)LIST name
  VAL(wert1) CHOICE(&auswahl1)
  VAL(wert2) CHOICE('Terminal Typ 3278')
```

Hinweis

Ob der Mächtigkeit der möglichen Angaben und Kombinationen lesen Sie bitte im Manual "ISPF Dialog Developer's Guide" nach.

Als praktisches Beispiel sieht ihr euch einfach das SETTINGS-Panel an (PANELID eingeben und den angezeigten Panelnamen mit "DDLIST M name" suchen lassen. Das Panel findet sich in einer IBM Systembibliothek wieder).

)MODEL – ISPF Panel Section

Beschreibt den dynamischen Teil des Tabellenpanels. Es wird das Aussehen eines Satzes für die Anzeige bestimmt (kann mehrzeilig sein). Wird eine Tabelle gezeigt, wird der)MODEL-Teil für jeden Tabellensatz durchlaufen.

Auf einer Bildschirmseite werden im)MODEL-Teil so viele Zeilen gezeigt, als der verbleibende Platz erlaubt (Schirmgröße - Größe des)BODY-Teils : Zeilen je Datensatz = abbildbare Datensätze).

Der)MODEL-Teil kann 1 bis 8 Zeilen groß sein. Abbildbar sind TEXT, Ein- oder Ausgabefelder. Nicht abbildbar sind AREAS (dynamisch, scrollbar oder grafisch), Variable in einer Textzeile oder Action Bars.

CLEAR(var..)

Löschen von Variablen vor Anzeige eines Tabellensatzes. Auf diese Weise können Optionsfelder für die Satzverarbeitung vor Präsentation jedes Datensatzes zurückgesetzt, oder SAVE-Variable, die nicht an jeden Datensatz angehängt wurden, geleert werden.

ROWS(..)

Steuert die Anzeige der Tabelle über zwei mögliche Parameter.

ALL

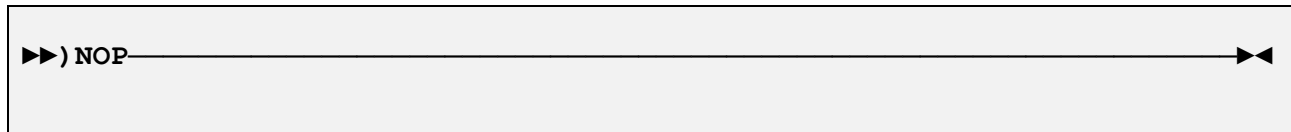
sorgt für die Anzeige aller Tabellensätze.

SCAN

Erlaubt die Filtrierung einer Tabelle. Die Anzeige erweckt den Anschein, als wären nur Sätze vorhanden, die ein bestimmtes Kriterium erfüllen (PLZ-Bereich 8, nur weibliche Mitarbeiter, Kunden mit einem bestimmten Umsatzvolumen).

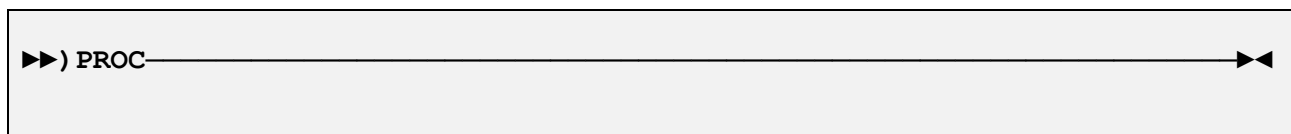
Die Suchargumente können sehr umfangreich gesetzt werden und erlauben neben allen gebräuchlichen Vergleichsoperatoren auch so genannte GENERIC SEARCHES (NAME=W*, führt zur Übereinstimmung, wenn das Namensfeld mit einem W beginnt.)

)NOP – ISPF File Tailoring Steueranweisung



Blindanweisung wie in REXX. Wird in der Regel im JA- oder NEIN-Zweig von)IF-Abfragen benutzt.

)PROC – ISPF Panel Section



Markiert den Teil des Panels, der bei DAT-FREIG durchlaufen wird. Hier werden im Allgemeinen alle Plausibilitätsprüfungen durchgeführt.

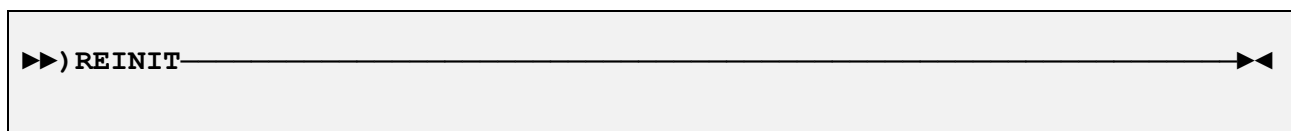
Statements zur Formalüberprüfung innerhalb des)PROC-Teils sind:

- IF – ELSE (zur Ablaufsteuerung)
- GOTO (zur Ablaufsteuerung)
- EXIT (zur Ablaufsteuerung)
- REFRESH (zur Feldauffrischung)
- VER (zur Feldüberprüfung)
- &var = wert (zur Wertzuweisung)
- VGET (Variable aus POOL lesen)
- VPUT (Variable in POOL schreiben)

Die Möglichkeiten im Detail können bei den jeweiligen Instruktionen nachgelesen werden.

Das Ende des)PROC-Teils wird erkannt, sobald die nächste Panel-Steueranweisung (schließende Klammer ")") in Spalte 1) auftritt.

)REINIT – ISPF Panel Section



Dieser Teil des Panels wird durchlaufen, wenn im)PROC-Teil eine Prüfung ein negatives Ergebnis zur Folge hatte. Wie innerhalb)INIT und)PROC können auch hier die Statements

- IF - ELSE (zur Ablaufsteuerung)
- GOTO (zur Ablaufsteuerung)
- EXIT (zur Ablaufsteuerung)

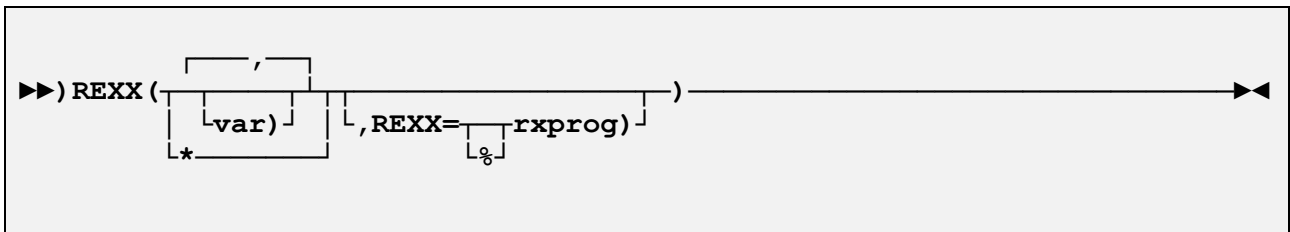
- REFRESH (zur Feldauffrischung)
- VER (zur Feldüberprüfung)
- &var = wert (zur Wertzuweisung)
- VGET (Variable aus POOL lesen)
- VPUT (Variable in POOL schreiben)

genutzt werden.

Die Möglichkeiten im Detail können bei den jeweiligen Instruktionen nachgelesen werden.

Das Ende des)REINIT-Teils wird erkannt, sobald die nächste Panel-Steueranweisung (schließende Klammer ") in Spalte 1) auftritt.

)REXX – ISPF File Tailoring Steueranweisung



Erlaubt die Einbindung von REXX-Logik in ein Skeleton. Dies bietet den Vorteil, dass auch im Skeleton der gesamte Funktionsumfang von REXX verfügbar ist, und darüber hinaus der REXX-Teil des Skeletons mit der REXX-Trace-Option getestet werden kann.

var

Hinter)REXX werden die Variablen angegeben, welche im Skeleton und in REXX ausgetauscht werden sollen. Wird ein Stern(*) angegeben, sind alle Variablen in beiden Richtungen durchlässig.

REXX=rxprog

Die REXX-Instruktionen können auch als externe Routine kodiert werden.

Beispiel

```
:
)REXX bl,#telfn,#telmob,#fax
bl=bl+(strip(tel1)='')+(strip(tel2)='')+(strip(fax)='')
)ENDREXX
:
```

Angenommen, BL ist im Skeleton mit einer 1 initialisiert und soll später eine bestimmte Anzahl Leerzeilen über die)BLANK-Anweisung erzeugen, kann auf diese Weise komfortabel ermittelt werden, wie viele Leerzeilen zu erstellen sind.

Hinweise

REXX-Code im Skeleton kann nur auf die Variablen zugreifen, welche im)REXX-Statement definiert sind.

REXX kann keine ISPF-Services aufrufen

Der REXX-Code muss mit)ENDREXX beendet werden

REXX kann die Variable ZFTXRC auf 0 oder 8 setzen, Im Falle 8 kann ein Grund in ZFTXMSG hinterlegt werden (anderenfalls ist der Default-Grund die Nachricht ISPF223)

)SEL ISPF – File Tailoring Steueranweisung

```
►►)SEL bedingung
```

Prüft eine Bedingung und leitet, sofern das Prüfergebnis positiv ist, eine Verarbeitungsfolge ein.

Das Ende der Verarbeitungsfolge wird durch das Statement)ENDSEL markiert.

)SET – ISPF File Tailoring Steueranweisung

```
►►)SET var =wert  
           [ &var ± wert ]
```

Erlaubt eine Wertzuweisung innerhalb eines Skeletons.

Var

Die zu füllende Variable wird ohne Ampersand(&) geschrieben.

Wert | &var ± wert

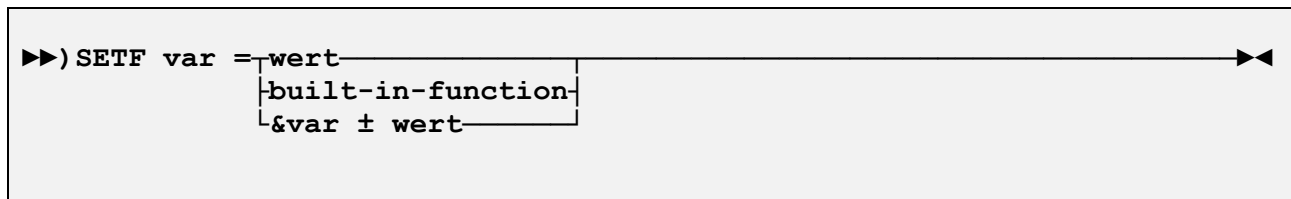
Wird der zu füllende Wert über eine Variable übergeben, wird diese mit Ampersand(&) kodiert, da für die Auflösung nicht der Feldname, sondern der Inhalt benötigt wird.

Als arithmetische Operationen sind nur PLUS(+) und MINUS(-) erlaubt.

Hinweis

Seit zOS 1.8 gibt es auch die)SETF-Anweisung. Diese ist deutlich vielseitiger und sollte heute den Vorzug erhalten.

)SETF – ISPF File Tailoring Steueranweisung



Erlaubt eine Wertzuweisung innerhalb eines Skeletons. Das Statement ist gleichzusetzen mit)SET. Im Unterschied dazu können bei)SETF aber alle Built-In-Functions benutzt werden.

Var

Die zu füllende Variable wird ohne Ampersand(&) geschrieben.

built-in-function

Folgende Built-In-Functions sind verfügbar:

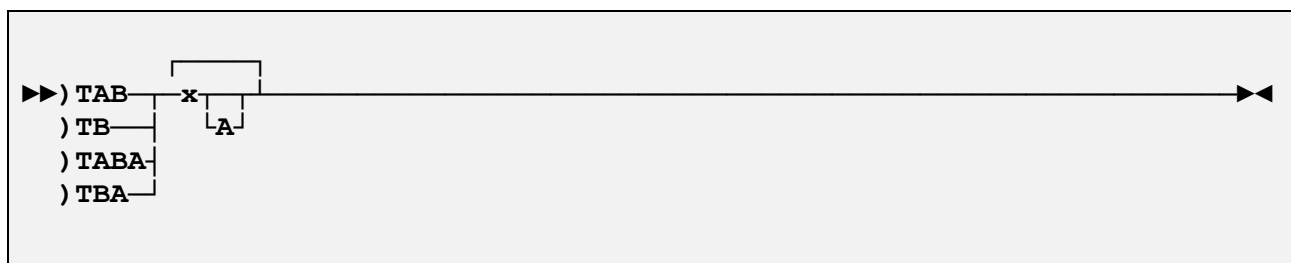
- &EVAL()
- &LEFT()
- &LENGTH()
- &RIGHT()
- &STR()
- &STRIP()
- &SUBSTR()
- &SYMDEF()

Die Funktionsweise kann bei den jeweiligen Funktionen nachgelesen werden.

Wert | &var ± wert

Wird der zu füllende Wert über eine Variable übergeben, wird diese mit Ampersand(&) kodiert, da für die Auflösung nicht der Feldname, sondern der Inhalt benötigt wird.

)TAB – ISPF File Tailoring Steueranweisung



Setzt Tabulatoren für die Textausrichtung. Über das vereinbarte Sonderzeichen (Standard ist das Rufzeichen(!)) können die definierten Tabulator-Haltepunkte angesprungen werden.

X

Liefert durch eine integere Zahl den Haltepunkt des Tabulators. Maximal können 16 Tabulatorstopps definiert werden.

&EVAL() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden, um erweiterte Arithmetik durchführen zu können.

Es können nur ganze Zahlen zwischen -2.147.438.647 und +2.147.438.646 benutzt werden.

Folgende Operatoren können benutzt werden:

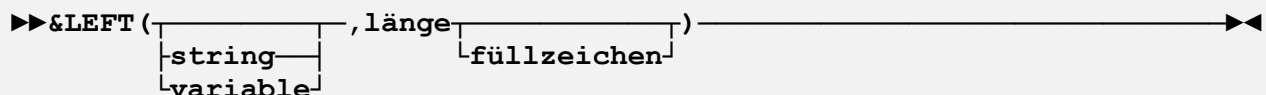
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Potenzierung
//	Divisionsrest

Hinweis

Es sind bis zu 32 ineinander verschachtelte Klammernpaare möglich.

Beispiel

```
)SETF erg = &EVAL(&VAR1 * (&VAR2-1))
```

&LEFT() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden.

String | variable

Zeichenkette oder Variable, die linksseitig beschnitten oder erweitert werden soll.

länge

Anzahl Byte ab Position 1.

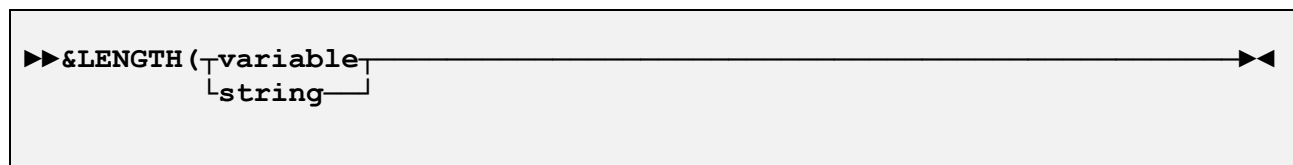
füllzeichen

Ist die Zeichenkette kürzer als die angegebene Länge, kann ein beliebiges Füllzeichen angegeben werden.

Beispiel

```
)SETF erg = &LEFT(&VAR1,5)
```

Überträgt die ersten 5 Byte aus der Variablen VAR1 in die Variable ERG.

&LENGTH() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden.

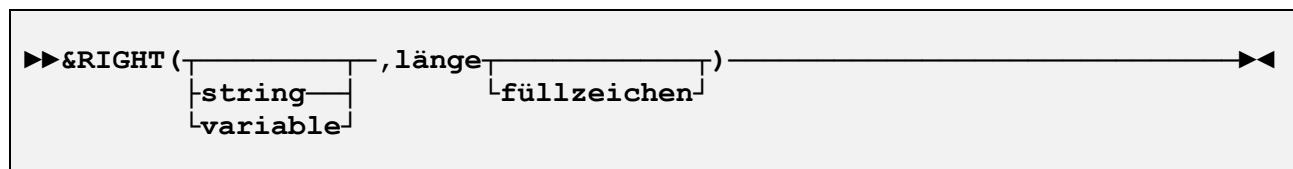
String | variable

Zeichenkette oder Variable, deren Länge ermittelt werden soll.

Beispiel

```
)SETF len = &LENGTH(&VAR1)
```

Überträgt die Länge der Variablen VAR1 in die Variable LEN.

&RIGHT() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden.

String | variable

Zeichenkette oder Variable, die rechtsseitig beschnitten oder erweitert werden soll.

länge

Anzahl Byte ab Position 1 von rechts.

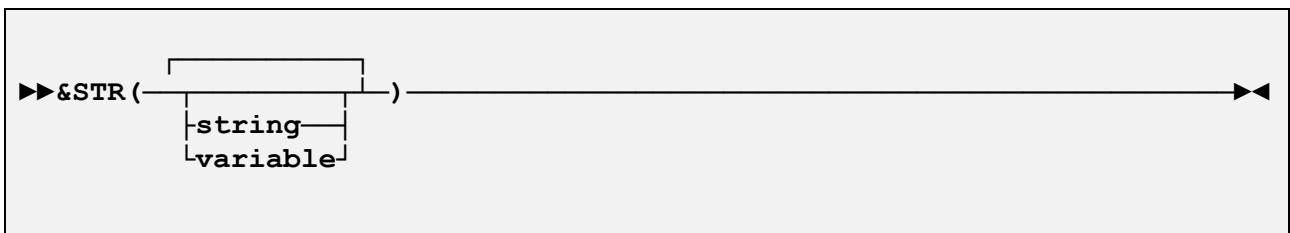
füllzeichen

Ist die Zeichenkette kürzer als die angegebene Länge, kann ein beliebiges Füllzeichen angegeben werden.

Beispiel

```
)SETF erg = &RIGHT(&VAR1,5)
```

Überträgt die letzten 5 Byte aus der Variablen VAR1 in die Variable ERG.

&STR() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden.

String | variable

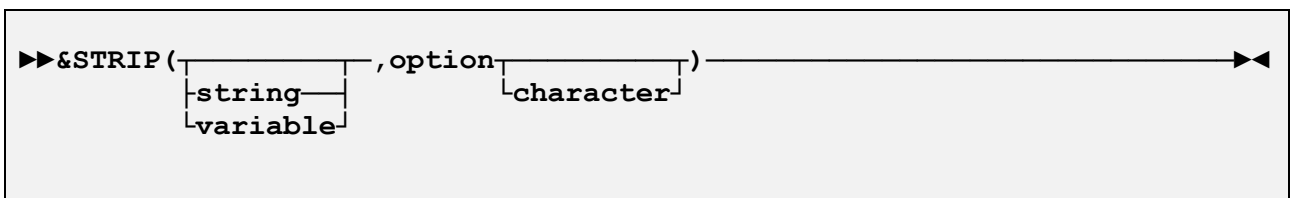
Zeichenkette oder Variable, die durch `&STR` bearbeitet werden soll. Es können eingebettete Blanks enthalten sein.

Ist keins von beiden angegeben, wird ein NULL-String returniert.

Beispiel

```
)SETF erg = &STR(&VAR1 &VAR2 &VAR3)
```

Überträgt die Inhalte der Variablen VAR1, VAR2 und VAR3 in die Variable ERG.

&STRIP() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden um führende und/oder nachlaufende Blanks zu entfernen.

String | variable

Zeichenkette oder Variable, die gestripped werden soll.

option

Ist eines der folgenden Zeichen:

L Leading (führende)
T Trailing (nachlaufende)
B Both (beidseitig)

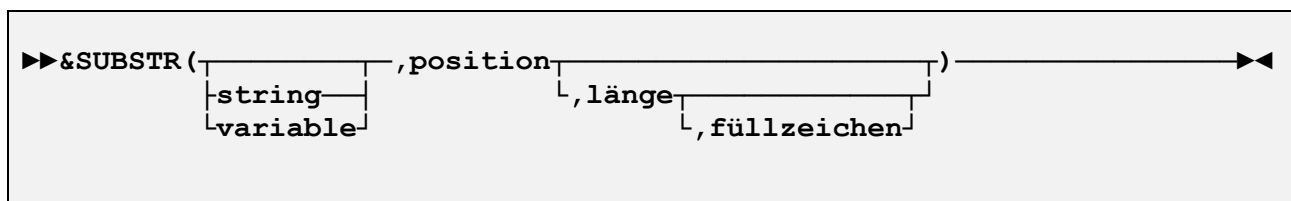
character

Ist das Zeichen, welches beschnitten werden soll. Standard ist BLANK.

Beispiel

```
)SETF erg = &STRIP(&VAR1,B)
```

Überträgt den Inhalt aus der Variablen VAR1 ohne führende und nachlaufende Blanks in die Variable ERG.

&SUBSTR() – ISPF File Tailoring Function

Kann in der)SETF-Anweisung im Skeleton benutzt werden um an einen Teil einer Variablen zu kommen.

String | variable

Zeichenkette oder Variable, die zerlegt werden soll.

position

Beginnposition, ab der eine Teilzeichenfolge übernommen werden soll.

länge

Anzahl Byte, die ab der angegebenen Beginnposition genommen werden sollen. Fehlt diese Angabe, wird ab der angegebenen Position bis zum Ende genommen.

füllzeichen

ragt die Längenangabe über das Ende der Zeichenkette oder Variablen hinaus, wird mit dem Füllzeichen aufgefüllt. Standardfüllzeichen ist Blank.

Beispiel

```
)SETF erg = &SUBSTR(&VAR1,3,2)
```

Überträgt die Stellen 3 und 4 aus der Variablen VAR1 in die Variable ERG.

&SYMDEF() – ISPF File Tailoring Function

►►&SYMDEF(symbolname)◄◄

Kann in der)SETF-Anweisung im Skeleton benutzt werden. Über die Funktion kann jedes definierte Symbol ausgelesen werden.

Welche Symbole zur Verfügung stehen ist von Unternehmen zu Unternehmen unterschiedlich. Die Einträge werden von der Systemprogrammierung festgelegt. Bitte erfragt die verfügbaren Symbole dort. Sofern ihr es zu sehen bekommt, hilft auch das SYM-Panel im SDSF.

symbolname

Name eines System- oder User-Symbols. Wenn unbekannt, wird ein Nullstring zurückgegeben.

Beispiel

```
)SETF erg = &SYMDEF(SYSCLONE)      /* kann sein */
)SETF erg = &SYMDEF(LHHMMSS)      /* kann sein */
```

*ENDREXX – ISPF Panel Function

►►*ENDREXX◄◄

Beendet die REXX-Verarbeitung im Panel. Das Statement muss im Panel nur kodiert werden, wenn über *REXX keine externe Routine aufgerufen wird.

*REXX – ISPF Panel Function

►►*REXX ([var] [,REXX=rxprog])◄◄

Erlaubt REXX-Logik in einem Panel aufzurufen. Das Codeing kann zwischen *REXX und *ENDREXX liegen, oder als externe Routine abgelegt sein.

Var | *

Liefert die Liste aller Variablen, welche von Panel nach REXX (und gegebenenfalls auch von REXX nach Panel durchlässig sein sollen).

Sollen alle Variablen durchlässig sein, kann als Variable auch Stern(*) angegeben werden.

Beispiele

```
) PROC
*REXX (*, (RXDATCHK)
) END
```

Zur Überprüfung eines Datumsfeldes wird die externe Routine RxDatChk aufgerufen. Alle Variablen zwischen Panel und REXX (und auch in der Gegenrichtung) sind an beiden Stellen verfügbar.

```
) PROC
*REXX (tt,mm,jj,datum)
  trace ?r
  parse var datum 1 tt 3 p1 4 mm 6 p2 7 jj 11 .
  /*
  wenn fehler erkannt wird, werden gesetzt:
    ZRXRC > 0
    ZRXMSG msgid
  Sonst
    ZRXRC = 0
  */
*ENDREXX
) END
```

Die Variable DATUM ist im)BODY definiert. TT, MM und JJ werden REXX-seitig erzeugt.

ABDISPCC – JCL EXEC Parameter

```
▶▶ABDISPCC=(cc-GE- )
               ^GT
```

Erlaubt in Bezug auf den DISP-Parameter die Aktion, welche für ABEND vorgesehen ist, auch auszuführen, wenn ein bestimmter Condition Code (Normal End) aufgetreten ist.

Beispiele

```
ABDISPCC=(8,GE)
```

Tritt ein RC 8 oder größer auf, wird die Aktion durchgeführt, welche in der DISP-Angabe für ABEND vorgesehen ist.

ABBREV() – REXX Built-in-Function

```
▶▶ABBREV(string1,string2
           [,länge])
```

Übergibt eine 1, wenn string2 identisch mit dem Beginn von string1 ist und die Länge von string2 mindestens die angegebene Länge hat. Ist keine Länge angegeben, wird 1 angenommen

Wird diese Bedingung nicht erfüllt, erfolgt die Rückmeldung von 0.

Beispiele

```
SAY ABBREV('Print','Pri')      1
SAY ABBREV('PRINT','Pri')     0
SAY ABBREV('PRINT','P',1)     1
```

```
/**REXX*****  
PARSE UPPER ARG TEST .  
IF ABBREV(TEST,'T',1) THEN TRACE '?R'
```

ABS() – REXX Built-in-Function

►►ABS(zahl)– ◀◀

Übergibt den absoluten Wert einer Zahl ohne Rücksicht auf das Vorzeichen und führende Nullen.

Beispiele

```
SAY ABS('005.95')      5.95
SAY ABS('-6.04')       6.04
```

ADDPOP – ISPF DM-Service

►►"ADDPOP"◀◀
┌POPLOC(feldname)┐┌ROW(zeile)┐┌COLUMN(spalte)┐

Window-Modus. Alle folgenden Panels werden in einem Fenster dargestellt. Der Window-Modus bleibt bis auf Widerruf (REMPop) bestehen; weitere ADDPOPs können folgen.

POPLOC(feldname)

Feld-Referenz zum vorangegangenen Panel. Die Positionierung des Fensters (links, oben) erfolgt unter dem angegebenen Feld.

Kann das Fenster nicht auf diese Weise geöffnet werden, wird es oberhalb des Feldes, oder seitlich versetzt angezeigt.

ROW(zeile) COLUMN(spalte)

Gibt über eine Zeilen-/Spaltenkoordinate die Lage der linken oberen Fensterecke innerhalb des letzten Fensters bekannt. Wurde noch kein Fenster geöffnet, bezieht sich die Koordinate auf den physischen Schirm.

Werden die Parameter in Verbindung mit POPLOC(feld) angegeben, erfolgt die Positionierung in Relation zum angegebenen Feld. Die Werte können positiv oder negativ sein.

Beispiele

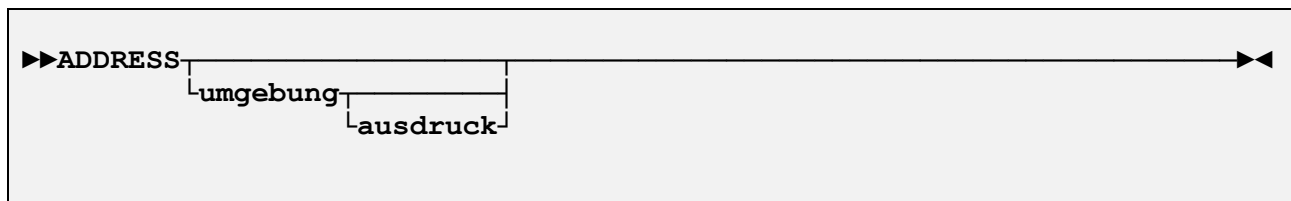
```
"addpop poploc(PLZ) row(2) column(3) "
```

Das nächste Fenster richtet sich am Feld "PLZ" aus, und verschiebt sich um zwei Zeilen nach unten und 3 Spalten nach rechts.

Return Codes

0	Normales Ende
12	ADDPop aufgerufen, ohne vorher ein Panel ohne diesen Service aufgerufen zu haben, oder zwei aufeinander folgende ADDPOPs abgesetzt.
20	Schwerer Fehler

ADDRESS – REXX Instruction



Mit ADDRESS wird die Systemumgebung benannt, an die ein Befehl zu übergeben ist. Erfolgt keine Befehlsangabe hinter der Umgebung, wird das angegebene Environment zum Standard, bis eine erneute Änderung erfolgt.

Umgebung

Unter TSO/E REXX sind folgende Umgebungen durch den ADDRESS-Befehl erreichbar:

TSO	TSO-Befehlsinterpreter
LINK	das LINK-Environment
ISPEXEC	ISPF-Dialog Manager
ISREDIT	ISPF-Editor (für Makroprogrammierung)

Beispiele

```
ADDRESS ISPEXEC "DISPLAY PANEL(PANEL1) "
```

```
ADDRESS ISPEXEC
"EDIT DATASET("varname") "
if RC = 0 then do
  "DISPLAY PANEL(PANEL1) "
  :
```

ADDRESS() – REXX Built-in-Function

►►ADDRESS ()◄◄

Meldet den Namen der Systemumgebung, innerhalb derer die Abarbeitung der externen Befehle durchgeführt wird.

Beispiele

```
SAY ADDRESS()      'TSO'      /* möglich */  
SAY ADDRESS()      'MVS'      /* möglich */  
SAY ADDRESS()      'ISPEXEC' /* möglich */
```

```
:  
OldEnv = ADDRESS()  
Address TSO  
Befehl  
Befehl  
Address OldEnv  
:
```

ADDRSPC – JCL JOB Parameter

►►ADDRSPC=VIRT◄◄
REAL

Entscheidet, ob während des Ablaufs des Jobs Paging durchgeführt werden darf .

VIRT

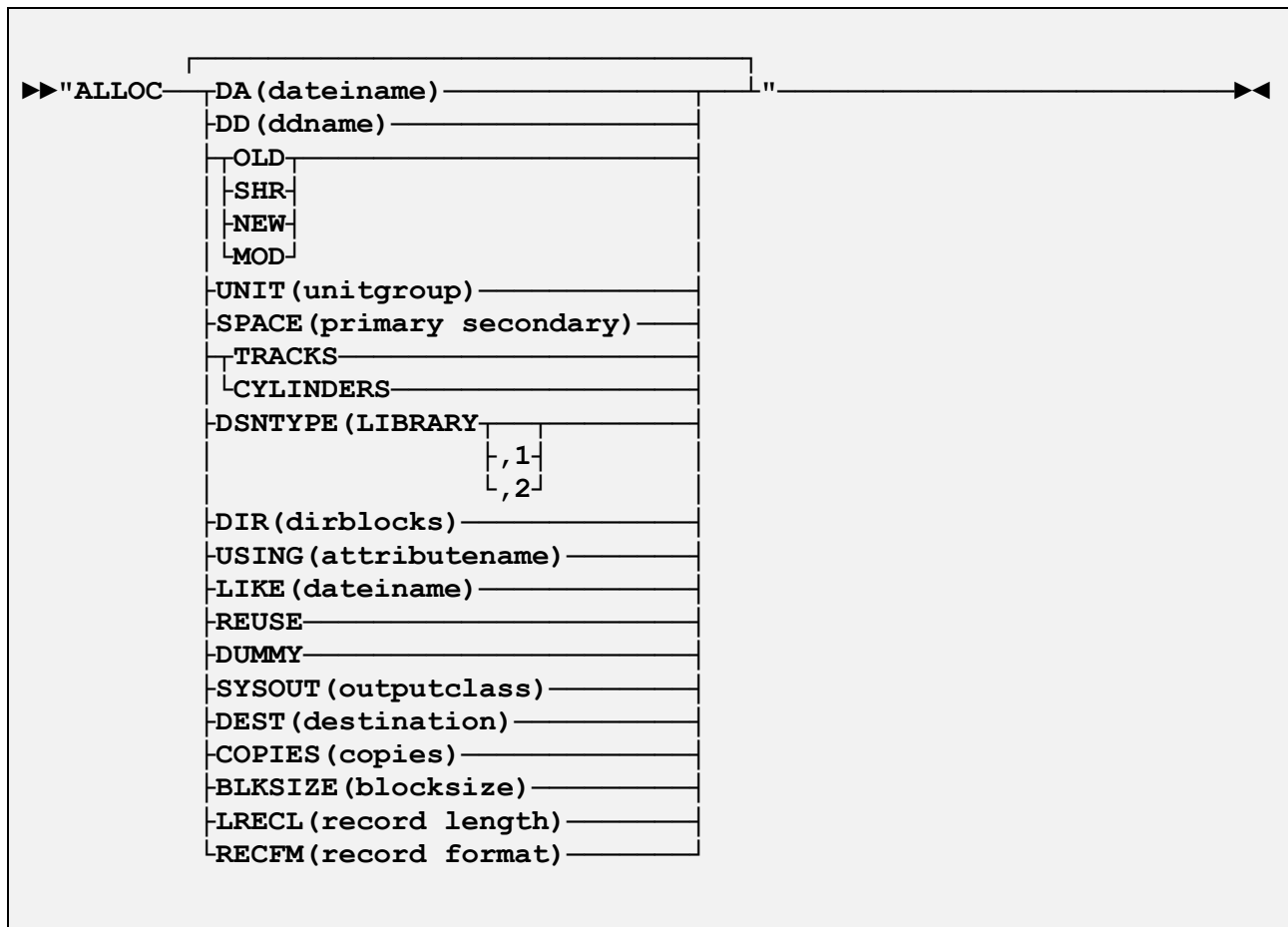
Für den betreffenden Adressraum darf PAGING im Bedarfsfall durchgeführt werden. Das heißt, bei Engpässen im Hauptspeicher kann eine Auslagerung auf die PAGE-Datasets (AUXILIARY STORAGE) durchgeführt werden.

REAL

Verhindert die Auslagerung auf die PAGE-Datasets während der gesamten Laufzeit. Die Verarbeitung ist immer resident.

Diese Angabe ist sinnvoller Weise nicht jedermann erlaubt. Könnte jedermann diese Angabe in der JOB-Anweisung treffen, wäre das virtuelle Speicherkonzept aufgehoben.

ALLOC – TSO Kommando



Dient der Zuordnung von Dateien, die für ein auszuführendes Programm benötigt werden.

DA(dateiname)

Benennt die Datei, die unter einem bestimmten symbolischen Namen zugeordnet werden soll. Sollen mehrere Dateien unter einem symbolischen Namen verkettet werden (DATASET-CONCATINATION), können alle der Reihe nach innerhalb der runden Klammern angegeben werden. Dabei werden die Zugriffe der Reihe nach entsprechend der Schreibweise von links nach rechts durchgeführt. Bei Dateiverkettung ist darauf zu achten, dass die Datei mit der größten Blockgröße als Erste genannt wird.

Wird an Stelle eines Dateinamens Stern(*) angegeben, wird für den symbolischen Namen das Terminal zugeordnet.

DD(ddname)

Symbolischer Name, der in einem Programm oder einer Prozedur benutzt wird. An Stelle von FI ist auch DD möglich.

OLD

Die Datei ist bereits vorhanden und wird für die Dauer der Verarbeitung exklusiv belegt.

SHR

Die Datei ist bereits vorhanden. Während der Verarbeitung können auch andere Benutzer auf diese Datei zugreifen.

NEW

Die Datei wird eingerichtet. NEW beinhaltet das Katalogisieren.

MOD

MOD prüft, ob die Datei vorhanden ist. Wenn ja, wird die existente benutzt. Ist sie nicht vorhanden, wird sie angelegt.

Wird mit Disposition MOD in eine Datei geschrieben, werden die Daten an das Ende angefügt (Dateifortschreibung). MOD darf nicht für Member von PO-Dateien verwendet werden.

UNIT(...)

Beschreibt die Gerätegruppe der Datei. Zu beachten sind die unternehmensspezifischen Regeln unter SMS. (Nur bei Neuanlage)

SPACE(primary secondary)

Legt die Menge der Primärzuweisung(x) und gegebenenfalls der Sekundärzuweisung(y) fest.

TRACKS oder CYLINDERS

Nimmt Bezug auf die Größenangaben im SPACE-Operanden.

DSNTYPE(LIBRARY[,n])

Zeigt auf ein PO/E-Dataset vom Typ 1 oder 2. Ist der Typ nicht angegeben, wird 1 angenommen.

DIR(...)

Anzahl der DIRECTORY-Blöcke bei Erstellung einer PO-Datei. Jeder DIR-Block kann etwa 5 bis 6 Membereinträge aufnehmen (benutzt doch einfach PO/E).

USING(attr)

Ordnet die Dateiattribute zu, die in einem ATTR-Befehl unter dem angegebenen Namen definiert wurden. Dies stammt aus der frühen Vergangenheit und wird heute kaum noch benutzt. Die Dateiattribute können im ALLOC auch direkt angegeben werden.

LIKE(dateiname)

Übernimmt alle Dateiattribute der angegebenen Datei.

REUSE

Überschreibt die Zuweisung eines symbolischen Namens in der Dateizuweisungstabelle. REUSE darf nur mit Disposition SHR erfolgen.

DUMMY

NOP-Anweisung (Null-Operation, Blindzuweisung).

SYSOUT(...)

Beschreibung einer Druckausgabedatei (SPOOLOUT). Die Druckausgabeklasse muss angegeben werden (A-Z und 0-9).

DEST(...)

Benennt den Remotedrucker, auf den die Druckausgabe geleitet werden soll.

COPIES(...)

Anzahl der Gesamt-Druckausgaben.

BLKSIZE(...)

Blockgröße der zu erstellenden Datei.

LRECL(...)

Satzlänge der Datei

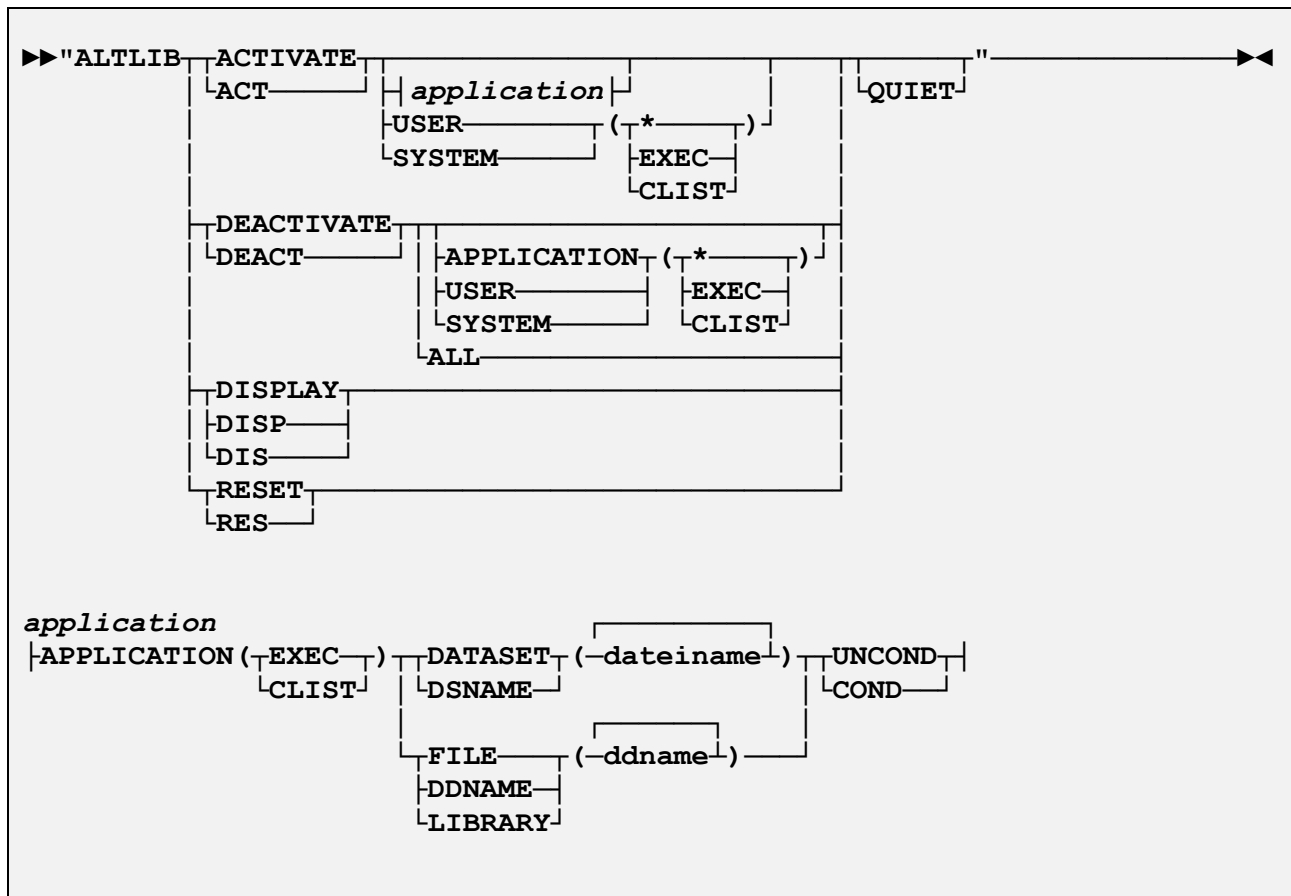
RECFM(...)

Satzformat der Datei.

Return Codes

0	Normales Ende
12	Fehler, Eine Fehlermeldung wird ausgegeben.

ALTLIB – TSO-Kommando



Das ALTLIB-Kommando kann für folgendes benutzt werden:

- Definition alternativer Applikations-Bibliotheken für REXX oder CLIST
- Definition Benutzer-spezifischer Bibliotheken für REXX oder CLIST
- Ausschluss einiger Bibliotheken bei der Suche nach REXX oder CLIST
- Rücksetzen der Definitionen.
- Anzeige der aktuellen Suchreihenfolge

ACTIVATE

Bestimmt, dass die Bibliothek in die Suchreihenfolge für REXX/CLIST einbezogen wird.

DEACTIVATE

Die Bibliothek soll bei der Suche nach REXX/CLIST nicht mehr berücksichtigt werden.

DISPLAY

Zeigt Informationen über die Suchreihenfolge an.

RESET

Setzt die Definitionen für Suchreihenfolge auf den SYSTEM-Level zurück.

USER

USER-Level für die Suche. Dieser wird über die DD-Namen SYSUEXEC/SYSUPROC zugewiesen.

APPLICATION

Der Application-Level wird durch die Zuweisung von Datei- oder DD-Namen zugewiesen.

SYSTEM

Die Definition des SYSTEM-Levels erfolgt über SYSPROC (für REXX und CLIST) oder SYSEXEC (nur für REXX).

ALL

Legt fest, dass alle Levels (SYSTEM, APPLICATION und USER) für REXX und CLIST deaktiviert werden. Möglicherweise hilft dann nur ein erneuter LOGON.

(EXEC)

Bestimmt, dass der entsprechende Level für REXX aktiviert oder deaktiviert wird.

(CLIST)

Bestimmt, dass der entsprechende Level für CLIST aktiviert oder deaktiviert wird.

(*)

Bestimmt, dass der entsprechende Level für REXX und CLIST aktiviert oder deaktiviert wird.

DATASET(datei) | DSNAME(datei)

Bestimmt, dass ein APPLICATION-Level definiert wird. Hier gibt es folgendes zu beachten:

- Die Dateien müssen PO- oder PO/E-Format haben
- Die Anzahl der Dateien ist auf 15 limitiert (DDNAME kann mehr)
- Die Blockung kann unterschiedlich sein. Die Reihenfolge ist beliebig
- Alle Dateien müssen das gleiche Satzformat haben.
- Membernamen dürfen nicht angegeben werden.

FILE | DDNAME | LIBRARY

Bestimmt, dass ein APPLICATION-Level definiert wird. Hier gibt es folgendes zu beachten:

- Der DD-Name muss vor Nutzung in ALTLIB zugewiesen sein.
- Der DD-Name muss mit einer PERMANENT-Attribut zugewiesen sein. Dateien, die während der LOGON-Phase zugewiesen werden, haben dieses Attribut automatisch.

UNCOND

Aktiviert den Application-Level auch dann, wenn bereits ein Application-Level vom gleichen Typ aktiv ist. Bis zu 8 Definitionen können gestacked werden.

COND

Aktiviert den Application-Level nur, wenn keiner vom gleichen Typ aktiv ist. Anderenfalls wird ein RC $\neq 0$ ausgegeben.

QUIET

Unterdrückt bis zu 99 Meldungen. Auf diese Weise kann ein ALTLIB-Display erfolgen und die Information im Programm verarbeitet werden. Siehe hierzu das folgende Beispiel

```
/* REXX
*/
ADDRESS TSO "ALTLIB DISPLAY QUIET"
ADDRESS ISPEXEC "VGET (IKJADM IKJADM1 IKJADM2 IKJADM3...) SHARED"
SAY "IKJADM " IKJADM
SAY "IKJADM1" IKJADM1
SAY "IKJADM2" IKJADM2
SAY "IKJADM3" IKJADM3
```

IKJADM enthält die Anzahl der erzeugten Meldungen. IKJADM1 und folgende enthalten die Meldungen (Satzlänge 151 Byte).

Hinweise

TSO sucht der Reihe nach in der USER- APPLICATION- und SYSTEM-Bibliothek nach externen Funktionen und Subroutinen. Zur Veranschaulichung folgende Tabelle:

Rang- folge	Bibliotheks-Level		DD-Name
1	USER	REXX exec	SYSUEXEC
2	USER	CLIST	SYSUPROC
3	APPL	REXX exec	Definiert mit File/Dataset-Operanden
4	APPL	CLIST	Definiert mit File/Dataset-Operanden
5	SYSTEM	REXX exec	SYSEXEC
6	SYSTEM	CLIST	SYSPROC

Die Standard-Suchreihenfolge kann von der Systemprogrammierung verändert werden.

Mit ALTLIB DISPLAY kann angezeigt werden, welche Bibliotheken durchsucht werden.

ALTLIB mit ISPF

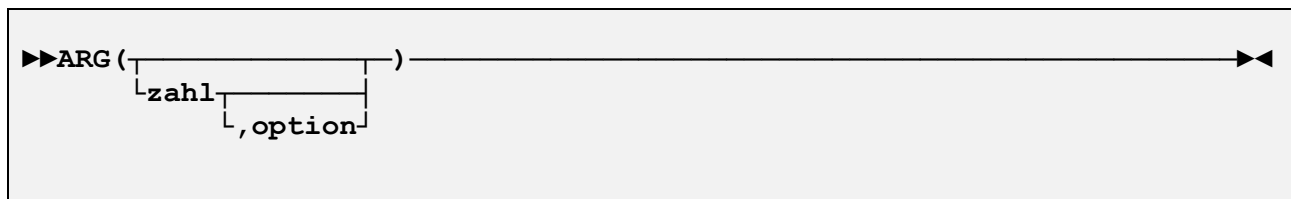
Wenn eine Anwendung unter ISPF per ALTLIB die Bibliothekslandschaft verändert, ist dies nur so lange gültig, als die Applikation lebt. Bei Ende der Anwendung tritt wieder der vorherige Zustand ein. Im Split-Screen-Modus wirken ALTLIB-Definitionen nur in demjenigen logischen Screen, in dem das ALTLIB-Kommando abgesetzt wurde.

Return Codes

- 0 Normales Ende
- 4 Keine alternate Library definiert. Deactivate ignoriert

- 8 Eine Application Library Definition existierte bereits. Altlib activate nicht durchgeführt. Die Meldung tritt nur in Verbindung mit dem COND-Parameter auf
- 10 User oder System Level EXEC konnte nicht aktiviert werden. Informieren Sie Ihre Systemprogrammierung
- 16 Ein benötigter DD-Name war nicht preallokiert
- 20 Schwerer Fehler. Eine Fehlermeldung wurde ausgegeben

ARG() – REXX Built-in-Function



Übergibt eine Folge von Parametern oder Informationen über Parameter entsprechend der angegebenen Option. Wird kein Parameter in der ARG-Funktion angegeben, wird die Anzahl der übergebenen Parameter gemeldet.

Zahl

Wenn angegeben, wird der betreffende Parameter gelistet. Existiert das Argument nicht, wird ein Nullstring rückgemeldet.

Option

Prüft ob ein Argument vorhanden ist (E(xists) liefert eine 1 bei Vorhandensein, sonst 0). Durch O(mitted) wird eine 1 bei Nichtvorhandensein, sonst 0 gemeldet.

Beispiele

Angenommen: %DEMO1 (Keine Parameterangaben)

```

SAY ARG()           0
SAY ARG(1)          ' '
SAY ARG(2)          ' '
SAY ARG(1, 'E')     0
SAY ARG(2, 'O')     1

```

Angenommen: %DEMO1 'a' , , 'b'

```

SAY ARG()           3
SAY ARG(1)          'a'
SAY ARG(2)          ' '
SAY ARG(3)          'b'
SAY ARG(4)          ' '
SAY ARG(1, 'E')     1
SAY ARG(2, 'e')     0
SAY ARG(2, 'o')     1
SAY ARG(3, 'O')     0
SAY ARG(4, 'O')     1

```

ATTRIB – TSO Command

```
►►"ATTRIB name [BLKSIZE (blk)] [LRECL (recl)] [RECFM (recfm)] [DSORG (org)] ◄◄
```

Der Attribut-Befehl erlaubt die Definition diverser Datei-Eigenschaften, welche im ALLOC-Befehl angesprochen werden können. Sollen z.B. 5 Datasets neu angelegt werden, die aber nur 2 unterschiedliche physische Aufbauten haben, ist es einfacher diese beiden Attribute unter wählbaren Namen zu definieren und auf das jeweilige Attribut im ALLOC Bezug zu nehmen.

Insgesamt verfügt der Befehl über eine Reihe weiterer Parameter. Wir wollen hier aber nur diejenigen betrachten, welche in der Praxis häufig vorkommen.

name

Name des Attributs (max. 8-stellig)

BLKSIZE(blk)

Definition der Blocksize

LRECL(recl)

Definition der Satzlänge

RECFM(recfm)

Definition des Satzformates. Möglich sind A, B, D, F, M, S, T, U und/oder V

DSORG(org)

Definition der Dateioorganisation. Möglich sind PO, POU, PS, PSU

Beispiel

```
"attrib psfb80 LRECL(80) BLKSIZE(0) DSORG(PS) "  
"alloc da(EDU.DEMO.DATA) NEW CATALOG SPACE(1) TRACK USING(psfb80) "
```

Return Codes

- 0 Normales Ende
- 12 Fehler. Wird von einer Fehlermeldung begleitet

AUTOLIST – Edit Macro

```
►►" (variable) = AUTOLIST"
►►"AUTOLIST = ON"
               OFF
```

AUTOLIST regelt, ob nach Ende der Edit Session veränderte Daten zur Druckausgabe gelangen. Die möglichen Modi sind ON oder OFF.

=AUTOLIST

Der aktuelle Wert von AUTOLIST wird in einer Variablen abgelegt.

AUTOLIST=

Der AUTOLIST-Modus wird entsprechend im Edit Profil hinterlegt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

AUTONUM – Edit Macro

```
►►" (variable) = AUTONUM"
►►"AUTONUM = ON"
               OFF
```

AUTONUM steuert die automatische Neummerierung der Daten bei SAVE. Dies zeigt nur Auswirkungen, wenn NUMBER aktiv ist.

=AUTONUM

Der aktuelle Wert von AUTONUM wird in einer Variablen abgelegt.

AUTONUM=

Der AUTONUM-Modus wird entsprechend im Profil hinterlegt.

Beispiele

```
"(altanum) = autonum"
```

Der aktuelle Wert für AUTONUM wird in ALTANUM hinterlegt.

```
"autonum = off"
```

AUTONUM wird ausgeschaltet.

Return Codes

0	Normales Ende
20	Schwerer Fehler

AUTOSAVE – Edit Macro



AUTOSAVE steuert das Wirken von END im Editor. Bei ON, werden die Daten bei END zurückgeschrieben, anderenfalls nicht.

=AUTOSAVE

Der aktuelle Wert von AUTOSAVE wird in einer Variablen ablegt.

AUTOSAVE=

Der AUTOSAVE-Modus wird entsprechend im Profil hinterlegt.

ON

Schaltet den AUTOSAVE-Modus ein

OFF PROMPT | NOPROMPT

Schaltet AUTOSAVE aus und stellt eine Frage nach Sicherung (PROMPT), wenn seit dem letzten SAVE geändert wurde und durch END gedrückt wird, oder nicht (mit Vorsicht zu genießen).

Beispiele

```
"(smod,sprompt) = autosave"
```

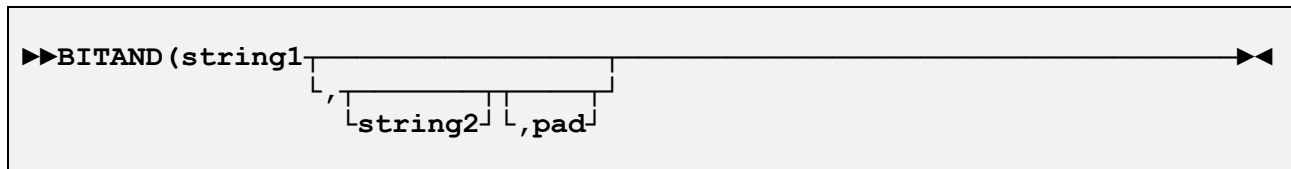
Der AUTOSAVE-Modus wird in der Variablen SMOD hinterlegt. Ist OFF gesetzt, steht in der Variablen SPROMPT der entsprechende Parameter.

```
"autosave = on"
```

AUTOSAVE wird eingeschaltet. Der aktuelle Zustand der Daten wird bei END gesichert.

Return Codes

0	Normales Ende
20	Schwerer Fehler

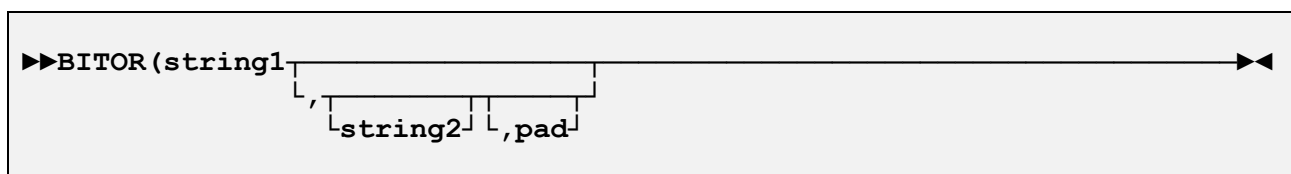
BITAND() – REXX Built-in-Function

Führt eine logische UND-Verknüpfung von zwei Strings auf der Binärebene durch. Die Länge des Ergebnisses wird durch den längeren der beiden Strings bestimmt. Wenn kein Füllzeichen angegeben ist, endet die Operation, wenn die Länge des kürzeren Strings erreicht ist. Der Rest des längeren Strings wird unverändert übernommen.

Ist ein Füllzeichen angegeben, wird der kürzere String mit dem Füllzeichen der Länge des längeren Strings angepasst.

Beispiele

```
SAY BITAND ("16"x)           "16"x
SAY BITAND ("73"x, "27"x)    "23"x
SAY BITAND ("13"x, "5555"x)  "1155"x
```

BITOR() – REXX Built-in-Function

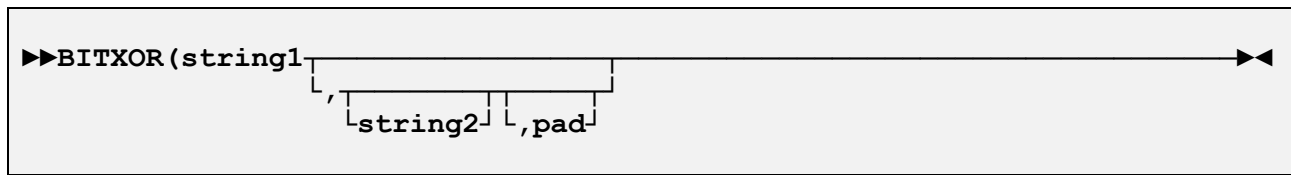
Führt eine logische ODER-Verknüpfung von zwei Strings auf der Binärebene durch. Die Länge des Ergebnisses wird durch den längeren der beiden Strings bestimmt. Wenn kein Füllzeichen angegeben ist, endet die Operation, wenn die Länge des kürzeren Strings erreicht ist. Der Rest des längeren Strings wird unverändert übernommen.

Ist ein Füllzeichen angegeben, wird der kürzere String mit dem Füllzeichen der Länge des längeren Strings angepasst.

Beispiele

```
SAY BITOR ("16"x)           "16"x
SAY BITOR ("15"x, "24"x)    "35"x
SAY BITOR ("15"x, "2456"x)  "3556"x
```

BITXOR() – Built-in-Function



Führt eine logische EXCLUSIV-ODER-Verknüpfung von zwei Strings auf der Binärebene durch. Die Länge des Ergebnisses wird durch den längeren der beiden Strings bestimmt. Wenn kein Füllzeichen angegeben ist, endet die Operation, wenn die Länge des kürzeren Strings erreicht ist. Der Rest des längeren Strings wird unverändert übernommen.

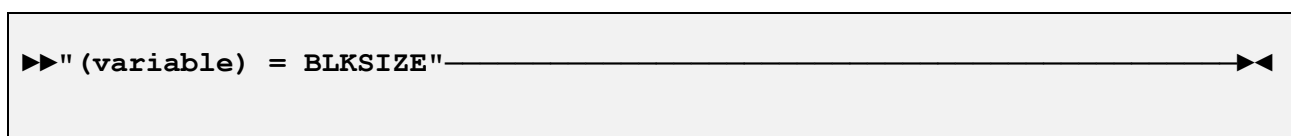
Beispiele

```
SAY BITXOR("16"x)      "16"x
SAY BITXOR("12"x,"22"x) "30"x
```

Die Exklusive Bit-ODER-Verknüpfung kann benutzt werden, um den Inhalt von zwei Feldern zu vertauschen, ohne eine Hilfsvariable benutzen zu müssen.

```
A = "Karlheinz"
B = "Wittmann"
Say a b                Karlheinz Wittmann
A = BITXOR(A,B)
B = BITXOR(B,A)
A = BITXOR(A,B)
Say a b                Wittmann Karlheinz
```

BLOCKSIZE – ISPF Edit Macro



Übergibt die Blockgröße der Daten an die angegebene Variable.

Beispiel

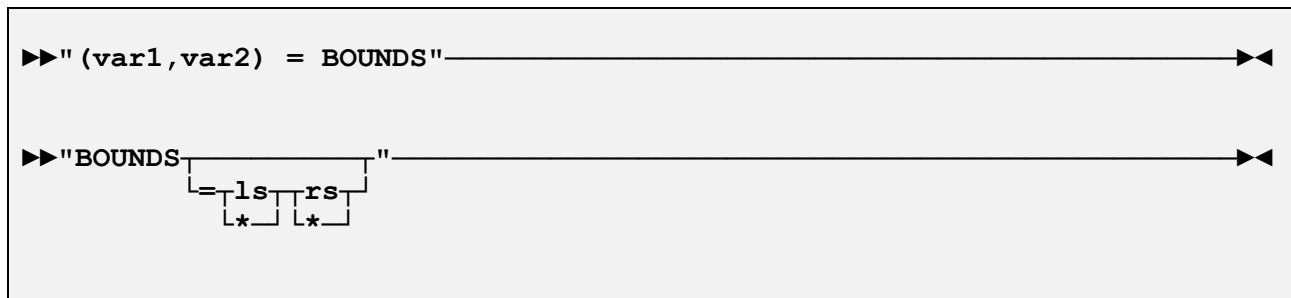
```
"(bsize) = blksize"
if bsize > 132 then ...
```

Die Blockgröße der bearbeiteten Datei wird in der Variablen BSIZE abgelegt und kann im Anschluss auf beliebige Werte hin überprüft werden.

Return Codes

```
0      Normales Ende
20     Schwerer Fehler
```


BOUNDARIES – ISPF Edit Macro



BOUNDS regelt die logischen horizontalen Dateigrenzen in Form eines linken und rechten Randes. Die Grenzen können abgefragt oder gesetzt werden.

=BOUNDS

Die linke und rechte Verschiebegrenze wird der Reihe nach den Variablen in der Klammer zugewiesen.

BOUNDS=

Die Verschiebegrenzen werden gesetzt. Wird nur BOUNDS kodiert, werden die Verschiebegrenzen auf Spalte 1 und Satzlänge gesetzt.

ls rs *

linke (ls) und rechte Spaltengrenze(rs). Wird Stern(*) für die links oder rechts angegeben, bleibt dies so, wie es augenblicklich gesetzt ist.

Beispiele

"bounds = 1 *"

Die linke Boundarygrenze wird auf Stelle 1 gesetzt, die rechte Grenze bleibt unverändert

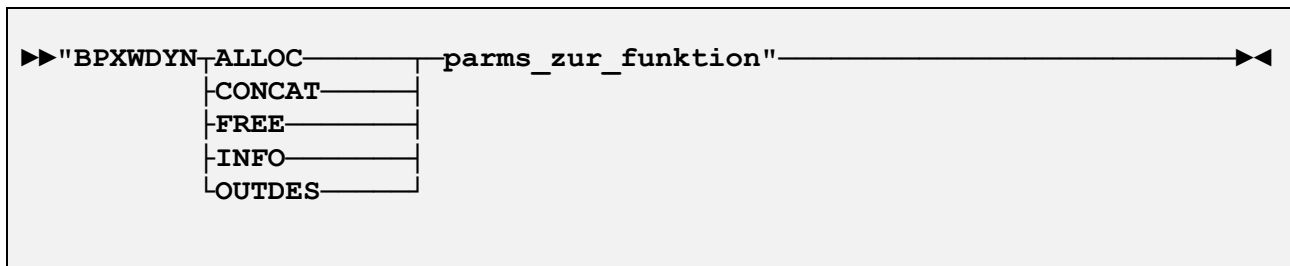
"(,rbnd) = bounds"

Die rechte Boundarygrenze wird in der Variablen RBND abgelegt. Wo die linke Grenze liegt, wird nicht festgehalten.

Return Codes

- 0 Normales Ende
- 4 Rechte Grenze > Default. Default wird gesetzt
- 12 Falsche Angaben
- 20 Schwerer Fehler

BPXWDYN – UNIX Command



Zu den verschiedenen Möglichkeiten sind hier nur die gängigen Parameter erklärt. Im Bedarfsfall finden sich die Details im Manual "Using REXX and z/OS Unix System Services"

ALLOC

Dynamische Zuweisung analog zum ALLOC unter TSO, aber mit umfangreicheren (besseren) Parametern

CONCAT

Konkatenieren von Dateien unter einem DD-Namen. Dafür gibt es im TSO nichts Vergleichbares.

FREE

Freigabe einer Zuweisung

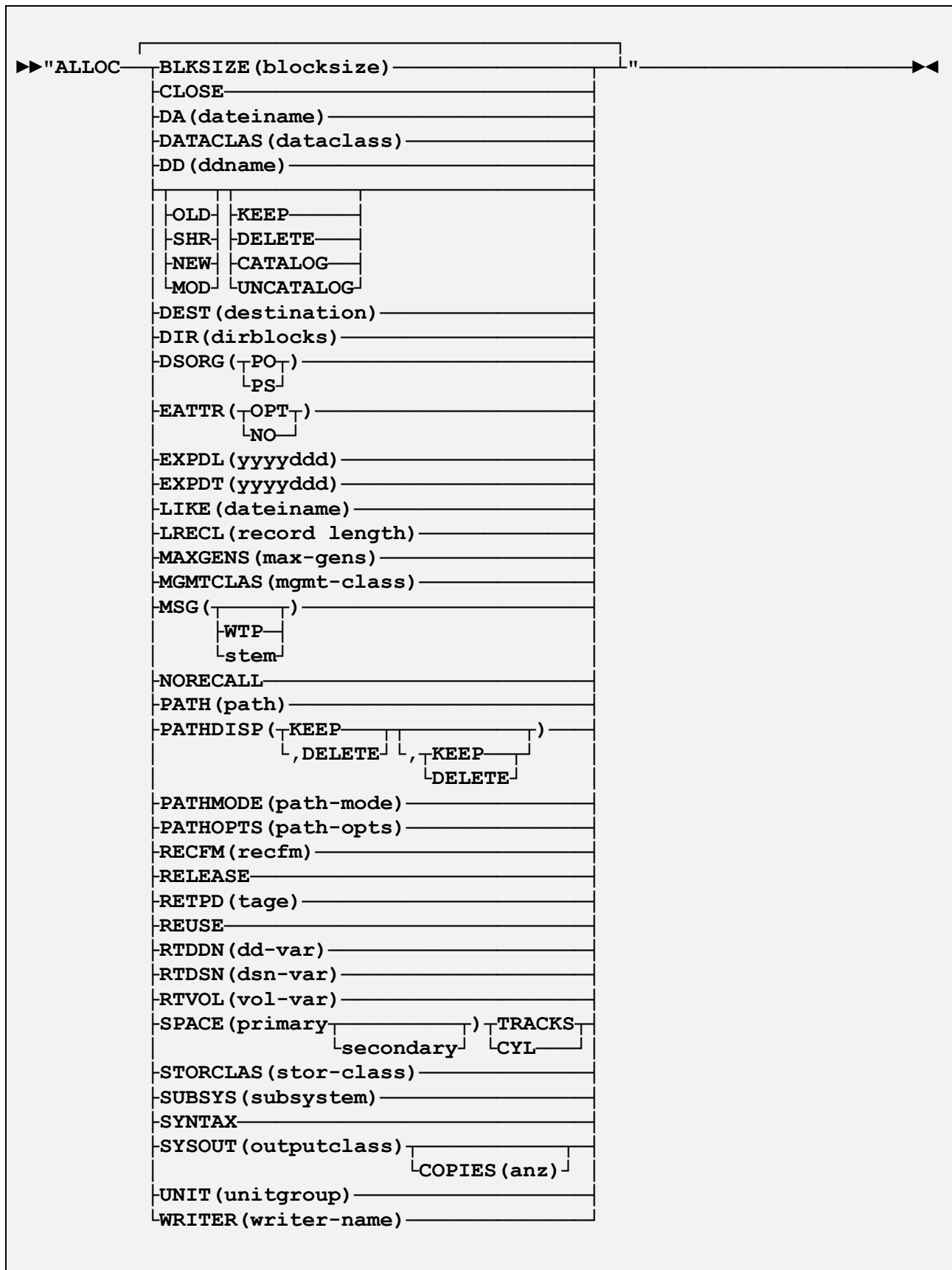
INFO

Informationen zur Datei

OUTDES

Definition eines Output Descriptors. Dieser Teil bleibt hier unberücksichtigt.

BPXWDYN ALLOC – UNIX Command



Die Möglichkeiten des ALLOCs über BPXWDYN sind umfangreicher als jene, welche im TSO möglich sind. Im Wesentlichen werden aber auch alle Funktionen des TSO-Pendants angeboten.

BLKSIZE(...)

Blockgröße der zu erstellenden Datei.

CLOSE

Gibt die Datei beim CLOSE des Programms frei

DA(dateiname)

Benennt die Datei, die unter einem bestimmten symbolischen Namen zugeordnet werden soll. Sollen mehrere Dateien unter einem symbolischen Namen verkettet werden (DATASET-CONCATINATION), können alle der Reihe nach innerhalb der runden Klammern angegeben werden. Dabei werden die Zugriffe der Reihe nach entsprechend der Schreibweise von links nach rechts durchgeführt. Bei Dateiverkettung ist darauf zu achten, dass die Datei mit der größten Blockgröße als Erste genannt wird.

Wird an Stelle eines Dateinamens Stern(*) angegeben, wird für den symbolischen Namen das Terminal zugeordnet.

Die Angabe eines GDGs über seine relative Nummer ist hier ebenfalls möglich

DATACLASS(dataclass)

Diese Angabe wird nur in Verbindung mit SMS unterstützt. Ist SMS nicht installiert oder nicht aktiv, wird die Angabe ignoriert. Durch SMS wird die Angabe ignoriert, sofern es sich dabei um eine bereits katalogisierte oder nicht von SMS gemanagte Datei handelt.

Eine Datenklasse regelt für eine Datei Dinge wie: Satzlänge, Satzformat, Dateityp, Sicherungsfrist und dergleichen.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

DD(ddname)

Symbolischer Name, der in einem Programm oder einer Prozedur benutzt wird. An Stelle von FI ist auch DD möglich.

OLD | SHR | NEW | MOD

Dateistatus. Dabei bewirkt

OLD	Datei bereits vorhanden, Zugriff exklusiv
SHR	Datei bereits vorhanden, Mehrfachzugriff erlaubt
NEW	Datei wird angelegt. Wird keine Disposition kodiert, wird sie auch katalogisiert
MOD	MOD prüft, ob die Datei vorhanden ist. Wenn ja, wird die existente benutzt. Ist sie nicht vorhanden, wird sie angelegt. Mit MOD werden Daten an das Ende angefügt (ist nur für PS-Dateien erlaubt).

KEEP | DELETE | CATALOG | UNCATALOG

Aktion, die bei Verarbeitungsende durchgeführt wird. Dabei bedeuten:

KEEP	Datei bleibt bestehen. Kein Katalogeintrag
DELETE	Datei wird gelöscht
CATALOG	Datei wird im Systemkatalog eingetragen
UNCATALOG	Datei wird im Systemkatalog ausgetragen

Die Angabe muss in einer vernünftigen Kombination mit dem Dateistatus gesehen werden. Ein NEW KEEP wäre eine Todsünde, da die Datei zwar in einem VTOC einer Platte, nicht aber im Systemkatalog eingetragen wäre. Wird das Volume, auf dem sie liegt, vergessen, ist sie mit logischen Mitteln nicht mehr auffindbar.

DEST(...)

Angabe eines JES-Printers. Kommt heute in aller Regel nicht mehr vor.

DIR(...)

Anzahl der DIRECTORY-Blöcke bei Erstellung einer PO-Datei. Jeder DIR-Block kann etwa 5 bis 6 Membereinträge aufnehmen.

DSORG(PO | PS)

Es soll eine PS- oder PO-Datei erstellt werden. Gegebenenfalls wäre zu hinterfragen, in wie weit heute noch mit dem alten PO-Format arbeiten möchte.

DUMMY

NOP-Anweisung (Null-Operation, Blindzuweisung).

EATTR(NO | OPT)

Regelt, ob "extended attributes" benutzt werden können (ist am einfachsten mit der für Platten zuständigen Stelle zu klären).

EXPDL(yyyyddd)

Freigabedatum mit 4-stelligem Jahr

EXPDT(yyddd)

Freigabedatum mit 2-stelligem Jahr

LIKE(dateiname)

Übernimmt alle Dateiattribute der angegebenen Datei (nur in Verbindung mit SMS).

LRECL(...)

Satzlänge der Datei

MAXGENS(...)

Anzahl von Member-Generationen einer Type-2-Library. Kann auch 0 sein. In diesem Fall werden keine Generationen gebildet, es kann aber der Vorteil von Type-2 genutzt werden.

MGMTCLAS(...)

Diese Angabe wird nur in Verbindung mit SMS unterstützt. Ist SMS nicht installiert oder nicht aktiv, wird die Angabe ignoriert. Durch SMS wird die Angabe ignoriert, sofern es sich dabei um eine bereits katalogisierte oder nicht von SMS gemanagte Datei handelt.

Eine Managementklasse regelt für eine Datei Dinge wie Backup und Migration.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

MSG(...)

Routet Meldungen des Alloc-Befehls. Per Default werden die Nachrichten in einem Stem S99MSG. Abgelegt. Ein beliebig anderer Stemname kann angegeben werden. Bei WTP landen die Messages im Joblog, wohin dieses zur Laufzeit auch immer zeigt (im TSO-Foreground wird dafür der Bildschirm benutzt - -WTP = Write To Programmer).

NORECALL

Alloc wird mit RC > 0 beendet, wenn die betreffende Datei migriert ist.

PATH(path)

Liefert die Pfadangabe für die Datei.

PATHDISP(normal-end,abnormal-end)

Bestimmt die Aktion für normales oder abnormales Ende.

PATHMODE(path-mode)

Setzt bestimmte Bits bei Neuanlage (siehe "Using REXX and z/OS Unix System Services").

PATHOPTS(path-opts)

Setzt bestimmte Options für die Allocation (siehe "Using REXX and z/OS Unix System Services").

RECFM(...)

Satzformat der Datei.

RELEASE

Freigabe nicht benutzten Speicherplatzes beim CLOSE auf das Dataset (für PS-Dateien).

REUSE

Überschreibt die Zuweisung eines symbolischen Namens in der Dateizuweisungstabelle. REUSE darf nur mit Disposition SHR erfolgen.

RTDDN(dd-var)

Wurde für die Zuweisung ein dynamischer DD-Name gewählt (kein DD()-Parameter angegeben), liegt der DD-Name in der angegebenen Variablen.

RTDSN(dsn-var)

Wurde für die Zuweisung ein dynamischer Dateiname gewählt (kein DA()-Parameter angegeben), liegt der Dateiname in der angegebenen Variablen.

RTVOL(vol-var)

Wurde kein Volume-Parameter benutzt, liegt der Name des Volumes in der angegebenen Variablen.

SPACE(primary secondary) TRACKS | CYL

Legt die Menge der Primärzuweisung und gegebenenfalls der Sekundärzuweisung fest und bestimmt, ob Spuren oder Cylinders für die Zuweisung berücksichtigt werden.

STORCLAS(stor-class)

Diese Angabe wird nur in Verbindung mit SMS berücksichtigt. Die Storage-Klasse liefert über einen definierten Namen Angaben wie UNIT= und VOL= bei einer NICHT-SMS-gemanagten Datei.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

SYNTAX

Wenn angegeben, wird der Alloc nicht ausgeführt, sondern lediglich auf Syntaxfehler geprüft.

SYSOUT(...) [COPIES(anz)]

Beschreibung einer Druckausgabedatei (SPOOLOUT). Die Druckausgabeklasse muss angegeben werden (A-Z und 0-9). Mit COPIES kann eine Mehrfach-Druckausgabe erfolgen.

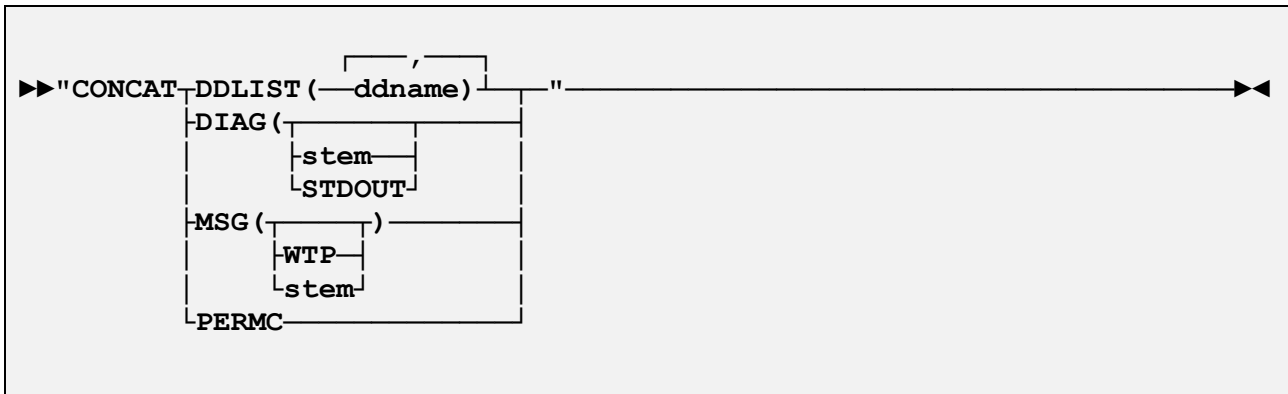
UNIT(...)

Beschreibt die Gerätegruppe der Datei. Zu beachten sind die unternehmensspezifischen Regeln unter SMS. (Nur bei Neuanlage)

WRITER(writer-name)

Name des External Writers.

BPXWDYN CONCAT – UNIX Command



CONCAT liefert eine einfache Möglichkeit, Dateien zu verketten. Unter TSO ist dies Sache des Anwenders (Programmierers) und läuft für gewöhnlich darauf hinaus, dass der Ist-Zustand über die Auswertung einer LISTALC-Abfrage ermittelt und festgehalten wird. Danach eine Erweiterung der Zuweisung stattfinden kann, um am Ende einer Verarbeitung den Urzustand wieder herzustellen (geht natürlich auch und in den meisten Unternehmen wird dafür eine selbstgeschriebene Routine eingesetzt).

DDLIST(ddname-1,ddname-2,ddname-x)

Die Dateinamen, welche mit ddname-2 und folgenden verbunden sind, werden mit ddname-1 verkettet.

DIAG(stem | STDOUT)

Bestimmt, wohin Diagnose-Informationen ausgegeben werden sollen.

MSG(...)

Routet Meldungen des Alloc-Befehls. Per Default werden die Nachrichten in einem Stem S99MSG. abgelegt. Ein anderer Stemname ist möglich. Bei WTP landen die Messages im Joblog, wohin dieses zur Laufzeit zeigt (im TSO-Foreground der Bildschirm - -WTP = Write To Programmer).

PERMC

Legt fest, dass es sich um eine "permanent concatenation" handelt.

Beispiel:

```

/* REXX YBPXWDY4 * Concat a dataset to SYSPROC / SYSEXEC
   The best of BPXWDYN © YCOS Yves Colliard Software */
mydsn=arg(1)
if mydsn="" then do
  parse source . . . . mydsn .
  mydsn="'"strip(mydsn)'"'
end
if sysdsn(mydsn)<>"OK" then do
  say "Error give dataset name" mydsn sysdsn(mydsn)
  exit 8
end

```

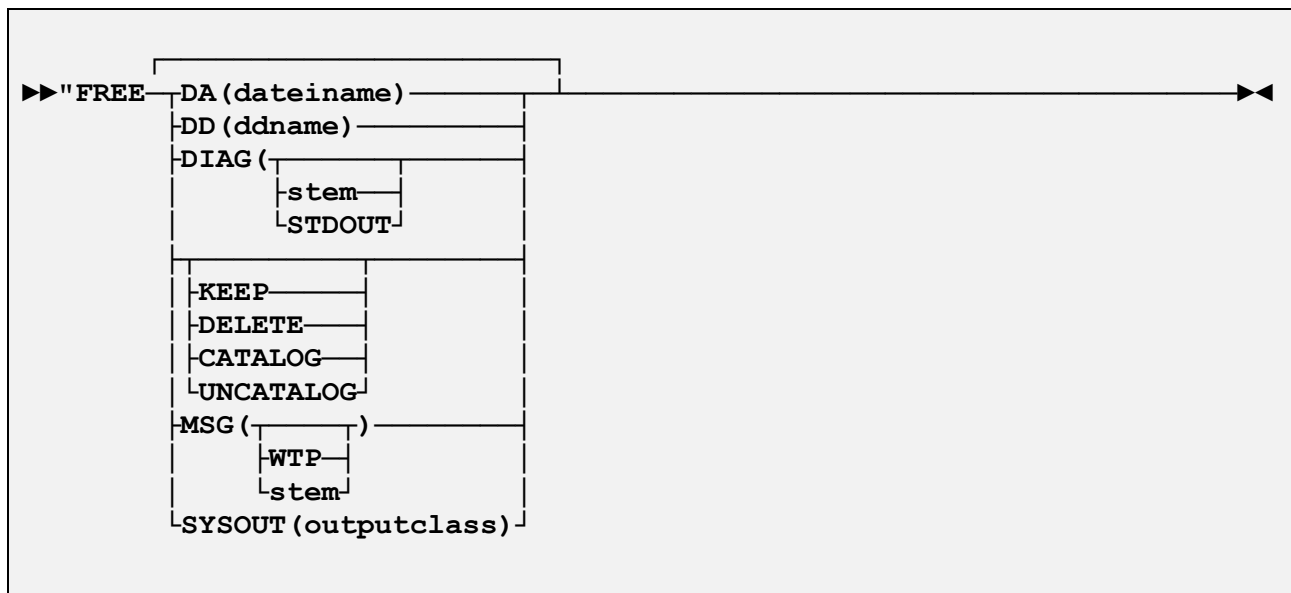


```

/* */
mydsn=strip(mydsn,"B","'")
/* */
if bpxwdyn("alloc da('"mydsn"' ) shr rtddn(mydd)")=0 then do
  if bpxwdyn("concat ddlist(sysproc,"mydd")")>0 then do
    if bpxwdyn("concat ddlist(sysexec,"mydd")")>0 then do
      say "Error:"
      say "Dataset" mydsn "not concatenated to SYSPROC/SYSEXEC"
      exit 8
    end
  end
  else do
    zedlmsg="Dataset" mydsn "concatenated to SYSEXEC"
    ADDRESS ISPEXEC "SETMSG MSG(ISRZ001)"
  end
end
else do
  zedlmsg="Dataset" mydsn "concatenated to SYSPROC"
  ADDRESS ISPEXEC "SETMSG MSG(ISRZ001)"
end
end
else do
  say "Error:"
  say "Dataset" mydsn "not allocated"
  exit 8
end
end

```

BPXWDYN FREE – UNIX Command



FREE

Freigabe einer Zuweisung

DA(dateiname)

Die Datei wird freigegeben

DD(ddname)

Der DD-Name wird freigegeben

DIAG(stem | STDOUT)

Bestimmt, wohin Diagnose-Informationen ausgegeben werden sollen.

MSG(...)

Routet Meldungen des Alloc-Befehls. Per Default werden die Nachrichten in einem Stem S99MSG. Abgelegt. Ein beliebig an derer Stemname kann angegeben werden. Bei WTP landen die Messages im Joblog, wohin dieses zur Laufzeit auch immer zeigt (im TSO-Foreground wird dafür der Bildschirm benutzt - -WTP = Write To Programmer.

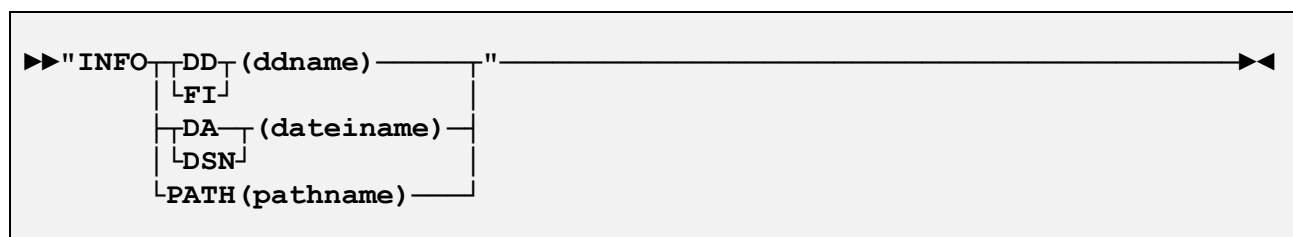
KEEP | DELETE | CATALOG | UNCATALOG

Überschreibt die Dispositionsangabe aus ALLOC

KEEP	Datei bleibt bestehen. Kein Katalogeintrag
DELETE	Datei wird gelöscht
CATALOG	Datei wird im Systemkatalog eingetragen
UNCATALOG	Datei wird im Systemkatalog ausgetragen

SYSOUT(outputclass)

Überschreibt die Outputklasse

BPXWDYN INFO – UNIX Command

Die verschiedenen Parameter liefert alle denkbaren Informationen über eine Datei (siehe folgende Tabelle):

Parameter	Liefert in der angegebenen Variablen
INRCNTL(variable)	CNTL Parameter
INRCACL(variable)	SMS Dataclass
INRDKYL(variable)	Key Label Parameter
INRDSNT(variable)	DSN Type
INRELNO(requestnummer)	Relative Entry Nummer
INRKEYO(variable)	VSAM Key Offset

INRLIKE(variable)	LIKE Parameter
INTMGCL(variable)	SMS Management Class
INRPCDS(variable)	USS Path Abnormal Disposition
INRPMDE(variable)	USS Path Mode
INRPNDS(variable)	USS Path Normal Disposition
INROPT(variable)	USS Path Options
INRREFD(variable)	REFDD Parameter
INRSECM(variable)	RACF Security Profile
INRSEGM(variable)	SYSOUT Segment Parameter
INRSPIN(variable)	SYSOUT Spin Parameter
INRSTCL(variable)	SMS Storage Class
INRTATT(variable)	Dynamic Allocation Attribute
INRTCDP(variable)	Conditional Disposition
INRTDDN(variable)	DD-Name
INTRDSN(variable)	Dateiname
INRTFDAT(variable)	Uss Record Format
INRTLIN(variable)	Number of Dealloc needed for Alloc
INRTLST(variable)	Letzter Eintrag eines GDGs ungleich 0
INRTLST(variable)	Relative Nummer des GDGs
INRTORG(variable)	DS-Org
INRTMEM(variable)	Membername
INRTNDP(variable)	Normal Disposition
INRTPATH(variable)	Path
INRTSTA(variable)	Status Disposition
INRTTYP(variable)	Dataset Type
INRTVOL(variable)	Volume (nur das erste)
INRTRECO	VSAM Record Type

BPXWDYN Return Codes – Unix Command

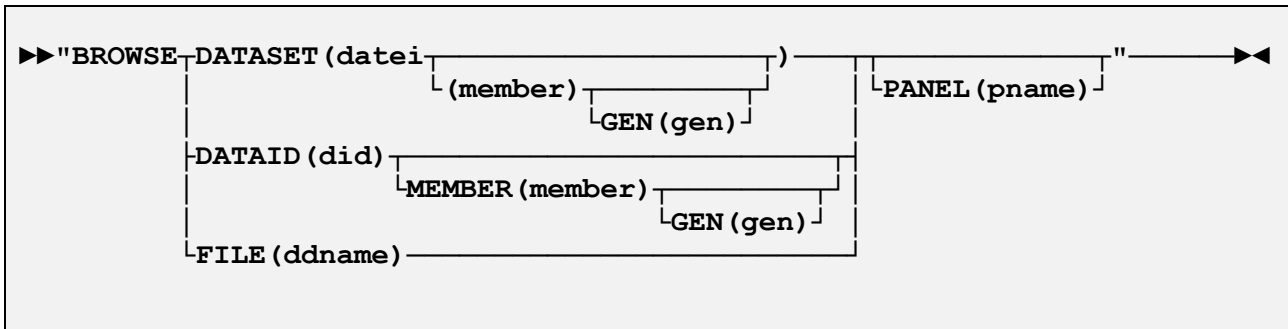
Die Return Codes von BPXWDYN sind sehr vielfältig. Meist ist nur das Problem, sie richtig zu interpretieren (siehe Beispiel in Band-1, Kochbuch). Folgende Return Codes sind möglich:

Return Codes

0	Normales Ende
20	Parameter Error
-21 bis -9999	Parameter Schlüsselwort Error (1 bis nnn)
-100##	Error IEFDB476 hat Code ## zurückgemeldet
>0 oder -1610612737 bis -2147483648	Fehler in Dynalloc Output Descriptor

Weitere Informationen sind in "Using REXX and z/OS UNIX System Services"

BROWSE – ISPF DM Service



ruft den ISPF-Browse und präsentiert gleichzeitig die hinter dem Schlüsselwort DATASET angegebene Datei.

Wird BROWSE aus einer Funktion (REXX etc.) aufgerufen, kann der Anwender nicht wie gewohnt durch =3.4 im PDF-Dialog verzweigen.

BROWSE kann durch END oder RETURN verlassen werden. Die Ablaufsteuerung wird immer an die Funktion im Hintergrund zurückgegeben, die entscheidet, was weiter zu tun ist.

Während der BROWSE-Session sind alle Befehle des BROWSE verfügbar.

Datei | datei(member)

Name der anzuzeigenden Datei. Wird der Name einer sequentiellen Datei angegeben, zeigt BROWSE die Daten.

Wird der Name einer PO-Datei ohne Member angegeben, liefert BROWSE das MembERVERZEICHNIS. Wird das Member mit angegeben, werden die Memberdaten gezeigt.

DATAID(did)

DatenID welche per LMINIT zugewiesen wurde

FILE(ddname)

DD-Name, unter dem das Dataset allokiert ist.

GEN(gen)

Generation eines PO/E-Type2-Datasets mit Angabe der relativen Generationsnummer (0, -1, -2, etc.).

PANEL(pname)

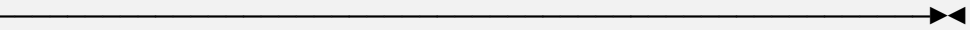
Panelname, sofern ein anderes als das Standardpanel der IBM benutzt werden soll.

Return Codes

- 0 Normales Ende
- 12 Satzlänge 0
- 14 Member oder Generation nicht gefunden
- 16 Keine Members zum Pattern gefunden oder keine Member in der Datei

- 18 Es wurde eine VSAM-Datei angegeben, aber in der ISPF-Configuration-Table ist dies nicht erlaubt.
- 20 Schwerer Fehler

BUILTIN – ISPF Edit Macro

"BUILTIN befehl" 

Erlaubt die Ausführung einer Editfunktion innerhalb eines Makros, die gleichnamig zu einem Makrobefehl oder einem Makronamen ist.

Wird der DEFINE-Befehl benutzt um der Funktion END einen Aliasnamen MACROEND zuzuweisen, wird MACROEND aufgerufen, wenn END kodiert wird. Um innerhalb MACROEND die END-Funktion aufzurufen, muss der Prefix BUILTIN vorangestellt werden, da sich das Macro sonst endlos immer wieder selbst aufrufen würde.

Beispiele

"builtin end"

Die Edit-Funktion END wird ausgeführt.

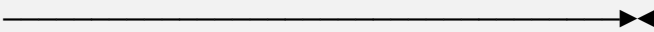
"builtin find ..."

Der Edit Find-Befehl wird ausgeführt.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

B2X() – REXX Built-in-Function

►►B2X(binary-string) 

Liefert den Hexwert eines Binärstrings.

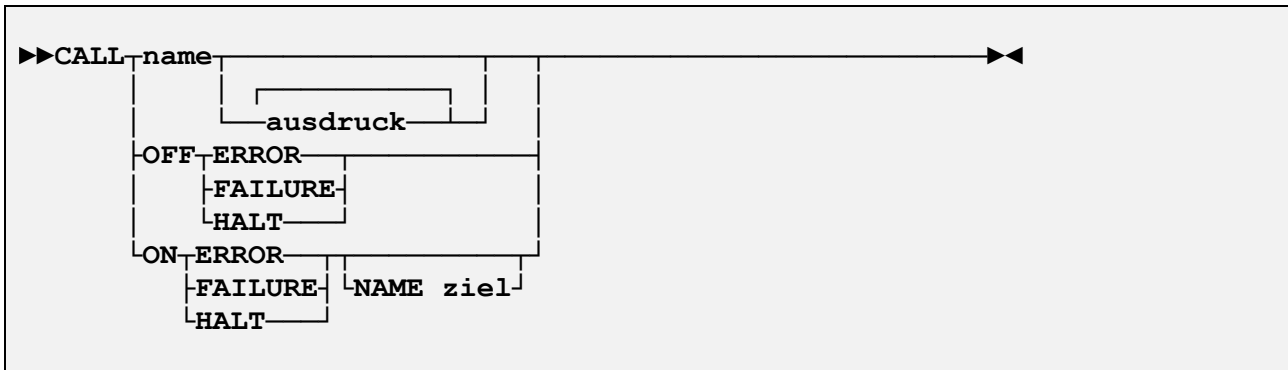
Binary-string

Zeichenkette bestehend aus 0 und 1.

Beispiele

```
SAY B2X("11000011")    "C3"  
SAY B2X("101")         "5"
```

CALL – REXX Instruction



Ruft ein REXX-Programm (intern oder extern) oder einen Event-Handler auf.

Name

Der Name der aufzurufenden Routine. Wird der Name als Literal kodiert, erfolgt immer der Aufruf eines externen Unterprogrammes, auch wenn im Programm ein gleichnamiges Label existiert.

Ausdruck

Übergabeparameter in Form von Literalen oder Variablen.

ON | OFF

Aktiviert oder deaktiviert die Reaktion auf bestimmte Events.

ERROR | FAILURE

Return Code größer oder kleiner 0

HALT

Unterbrechung durch PA-1-Taste (Attention-Key).

NAME ziel

Weist ein anderes Sprungziel zu, als den namensgleichen Event.

Hinweis:

CALL ON kann benutzt werden, wenn es sinnvoll erscheint, nach Abhandeln eines Events in das Programm zurück zu springen. Siehe auch unter SIGNAL.

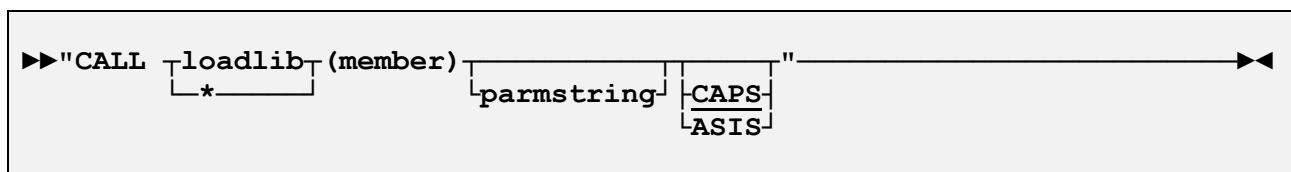
Beispiel:

```

/* REXX * Event-Handling *****/
Call on HALT name PA1
/* Hier liegt die Programmverarbeitung */
EXIT

PA1:
Say "PA-1-Taste ist außer Betrieb"
RETURN

```

CALL – TSO Command

Aufruf eines Lademoduls. Das Programm ist unter dem angegebenen Namen Member der angegebenen PO-Datei.

Soll der Aufruf des Loadmodules über die Linklist erfolgen (ohne zu wissen, in welcher Bibliothek das Modul residiert) kann an Stelle des Bibliotheknamens auch Asterisk(*) kodiert werden.

Parmstring

Übergabeparameter bis 100 Zeichen.

CAPS | ASIS

Parameter werden in Uppercase übergeben (CAPS = Default) oder nicht.

Beispiel Programmaufruf aus einer Prozedur:

```

/* REXX * IEBCGENER *****/
say "Eingabe?"; pull ein
say "Ausgabe?"; pull aus
"ALLOC FI(SYSUT1)  DA("ein")..."
"ALLOC FI(SYSUT2)  DA("aus")..."
"ALLOC FI(SYSIN)   ..."
"ALLOC FI(SYSPRINT)..."
"CALL *(IEBCGENER)"
"FREE FI(SYSUT1 SYSUT2 SYSIN SYSPRINT)"
EXIT

```

Return Codes

- 0 Normales Ende
- 12 Fehler. Eine Fehlermeldung wurde ausgegeben.
- others Return Code des aufgerufenen Programms

CANCEL – ISPF Edit Macro

```
"CANCEL"_____
```

CANCEL beendet die Edit-Sitzung, ohne die aktuellen Daten zu sichern. Im Zusammenhang mit CANCEL sind auch die Befehle DATA_CHANGED, AUTOSAVE und END interessant.

Bei CANCEL erfolgt keine Druckausgabe der Daten, auch wenn AUTOLIST ON gesetzt ist.

Return Codes

0	Normales Ende
20	Schwerer Fehler

CANCEL – TSO Command

```
►►"CANCEL jobname [ (jobnummer) ] [ PURGE ] [ NOPURGE ]"_____
```

Erlaubt den Abbruch einer Verarbeitung.

Sofern der Jobname nicht eindeutig ist, muss die Jobnummer mit angegeben werden.

PURGE

Aller Output wird aus den Spoolausgabedateien gelöscht.

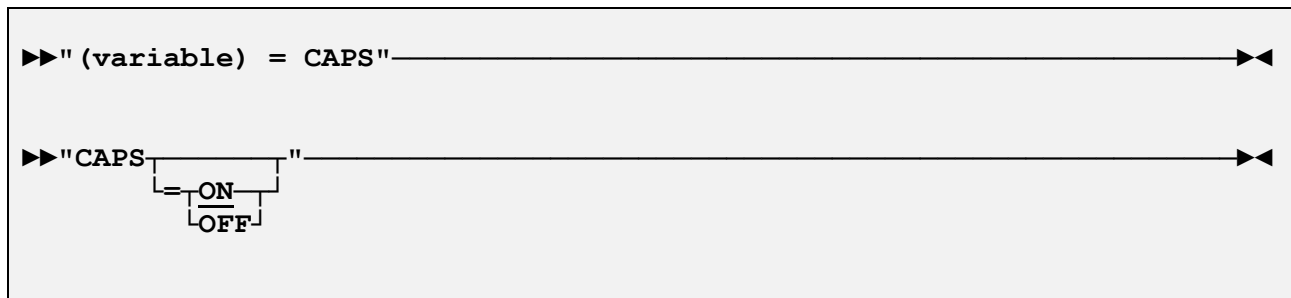
NOPURGE

Der CANCEL erfolgt nur, wenn der Job aktiv ist. Alle Spool-Outputs bleiben erhalten.

Return Codes

0	Normales Ende
12	Fehler. Eine Fehlermeldung wurde ausgegeben.

CAPS – ISPF Edit Macro



CAPS regelt die Umsetzung der Kleinbuchstaben zu Großbuchstaben. Ist die Großschreibung nicht unbedingt zwingend (Job Control oder ähnliches), sollte der Modus ausgeschaltet werden.

=CAPS

Der aktuelle Zustand von CAPS wird in der angegebenen Variablen hinterlegt.

CAPS=

Setzt den Modus für das Umsetzen der Kleinbuchstaben.

ON

Getippte Buchstaben werden bei <ENTER> in Großschreibung umgewandelt. ON wird auch angenommen, wenn nur CAPS kodiert wird.

OFF

Buchstaben werden unverändert so in der Datei abgelegt, wie sie getippt wurden.

Beispiele

```
"(altcaps) = caps"
```

Der aktuelle Wert für CAPS wird in der Variablen ALTcaps hinterlegt. Wird der CAPS-Modus im Macro verändert, kann der alte Zustand wieder hergestellt werden.

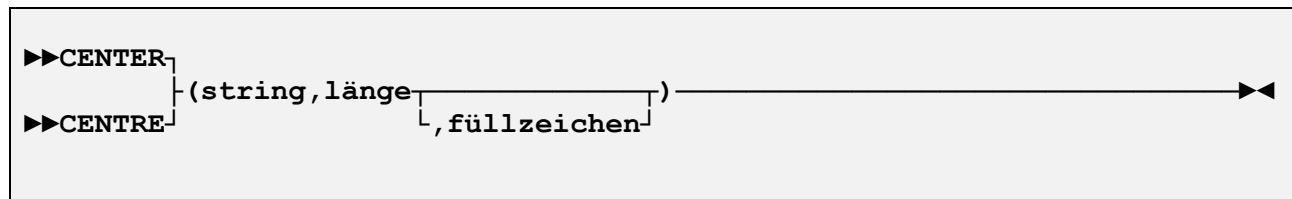
```
"caps"
```

Der CAPS-Modus wird eingeschaltet.

Return Codes

0	Normales Ende
20	Schwerer Fehler

CENTER() – REXX Built-in-Function



Zentriert eine Zeichenfolge.

String

Zeichenfolge die zentriert werden soll.

Länge

Breite, in der der angegebene string zentriert wird.

Füllzeichen

Ist keines angegeben, wird BLANK angenommen.

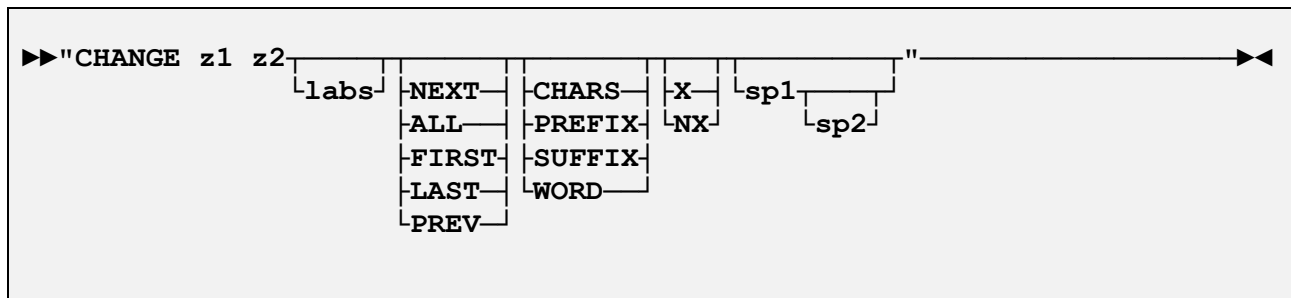
Beispiele

```
SAY CENTRE('HUGO',8,'-')      '--HUGO--'  
SAY CENTER('Feierabend um 4',5)  'abend'
```

```
SAY CENTER(' Das ist eine Titelzeite ',79,'*')
```

Zentriert den Text auf Bildschirmmitte und ersetzt die links und rechts stehenden Leerzeichen durch Stern(*).

CHANGE – ISPF Edit Macro



CHANGE ändert eine Zeichenkette in eine andere. Die Parameter sind die gleichen, wie beim Edit Kommando. Sind Zeilen zum Zeitpunkt des CHANGE-Befehls ausgeblendet und CHANGE vollzieht in ihnen eine Änderung, werden sie automatisch wieder eingeblendet.

z1

Zeichenkette, nach der gesucht wird.

z2

Zeichenkette, die anstelle von z1 eingesetzt wird.

labs

Eine von/bis Angabe in Form zweier Marken (Labels). Die Angabe relativer Zeilennummern ist nicht möglich.

Richtung

NEXT	Suche beginnt ab Cursorposition in Richtung Dateiende
ALL	Suche beginnt am Dateianfang in Richtung Ende. Gesucht und geändert werden alle vorkommenden Begriffe
FIRST	Die Suche beginnt am Dateianfang in Richtung Ende. Geändert wird der erste vorkommende Begriff
LAST	Wie FIRST aber vom Ende in Richtung Anfang
PREV	Wie NEXT aber in Richtung Dateianfang

Position

CHARS	Die Zeichen müssen wie angegeben aufeinander folgen, können aber an beliebiger Stelle vorkommen
PREFIX	ZEICHENFOLGE MUSS EIN Wortanfang sein
PREF	
SUFFIX	Zeichenfolge muss ein Wortende sein
SUF	
WORD	Die Zeichenfolge muss ein Wort sein

X

Der Befehl wirkt nur in ausgeblendeten Zeilen (eXcluded lines).

NX

Der Befehl wirkt nur in eingeblendeten Zeilen (Not eXcluded lines)

sp1 sp2

Spaltenangaben, innerhalb derer der CHANGE-Befehl wirken soll. Die erste Angabe ist immer die linke Spalte, die zweite Angabe bezieht sich auf die rechte Spalte, der zweite Wert muss größer dem ersten sein.

Wird nur ein Wert angegeben, wird er als definitive Spaltenangabe interpretiert, auf der die Zeichenfolge1 starten muss.

Beispiel

```
address isredit
"(MNAME) = MEMBER"
"change word" mname "'Pgm:"mname"'" .a .b 5
```

Der Name des aktiven Members wird ermittelt. Dieser Name wird gesucht und mit dem Prefix Pgm: ergänzt. Der Name muss auf Spalte 5 beginnen und wird nur zwischen den Label .A und .B gesucht.

Hinweise zum CHANGE-Befehl

Wird mit CAPS OFF gearbeitet, wird trotzdem nicht Case-sensitiv gesucht.

Enthalten zeichenfolge1 und/oder zeichenfolge2 Sonderzeichen oder Blanks, oder sind sie gleichnamig zu Operanden des Befehles, müssen sie zwischen Apostrophe (') eingegrenzt werden.

CHANGE wirkt innerhalb der Boundaries. Es ist ggfls. sinnvoll, die aktuellen Grenzwerte zu ermitteln, auf linke und rechte Außengrenze zu setzen, und vor Ende des Makros wieder rückzusetzen.

Wird als Suchrichtung NEXT oder PREV zusammen mit einem Bereich angegeben und der Cursor steht außerhalb des Bereiches, wird der ganze Bereich durchsucht. Ist der Cursor innerhalb des Bereiches, beginnt die Suche ab Cursorposition.

Wird eine Zeichenfolge verändert, ändert sich damit auch die Position des Cursors. Wird sie nicht gefunden, bleibt der Cursor unverändert.

Wird CHANGE ALL angewandt, werden gegebenenfalls zwei mögliche Hinweise gesetzt:

==CHG> für Zeilen in denen eine Änderung durchgeführt wurde

==ERR> für Zeilen, in denen eine Änderung nicht möglich war.

Diese Hinweise können mit LOCATE aufgespürt und durch RESET zurückgesetzt werden.

Soll CHANGE ausgeführt werden und der Status der Zeilen (excluded oder not excluded) ist nicht sichergestellt, kann CHANGE im Verbund mit SEEK und XSTATUS angewandt werden. Die erste in Frage kommende Zeile wird mit SEEK gesucht, dann wird der Zeilenstatus festgestellt. Nachdem CHANGE die Änderung durchführte, kann, sofern die Zeile vorher ausgeblendet war, ihr Status wieder wie vorher hergestellt werden.

```
"seek 'MVS/XA'"
do while rc = 0
  "(x) = xstatus .zcsr"
  "change .zcsr 'MVS/XA' 'MVS/ESA'"
  "xstatus .zcsr =" x
```

```
"seek 'MVS/XA'"  
End
```

Return Codes

0	Normales Ende
4	Suchbegriff nicht gefunden
8	"z2" länger als "z1". Änderung mindestens einmal nicht möglich.
12	Inkonsistente Parameter. Suchbegriff innerhalb der Boundaries nicht gefunden
20	Schwerer Fehler

CHANGE_COUNTS – ISPF Edit Macro

```
►►" (var1,var2) = CHANGE_COUNTS"◄◄
```

Diese Zuweisung liefert die durchgeführten Änderungen des CHANGE-Befehles.

var1

Anzahl der Änderungen.

var2

Anzahl der nicht durchführbaren Änderungen.

Beispiel

```
"(good,bad) = change_counts"
```

Die Anzahl Änderungen liegt in der Variablen GOOD, die Anzahl aufgetretener Fehler in BAD.

Return Codes

0	Normales Ende
20	Schwerer Fehler

CHARIN() – UNIX Function

```
►►CHARIN([name],[start],[länge])◄◄
```

Übergibt eine Anzahl Byte aus einem Datenstrom.

Name

Der Name eines Streams. Fehlt die Angabe, wird von STDIN gelesen.

Start

Byteposition, ab der gelesen werden soll. Fehlt die Angabe, wird ab dem aktuellen Pointer gelesen

Länge

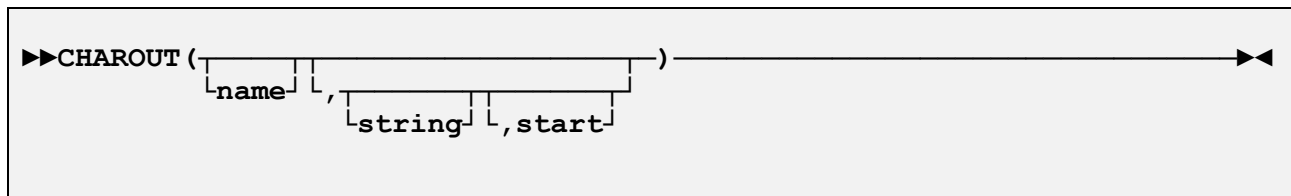
Anzahl der Byte, die gelesen werden sollen

Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Beispiel

```
/* REXX
*/
File = 'unix_file_name'
data = charin(file,,132)
```

CHAROUT() – UNIX Function

Übergibt eine Anzahl Byte aus einem Datenstrom.

Name

Der Name eines Streams. Fehlt die Angabe, wird nach STDOUT geschrieben.

string

Daten, die geschrieben werden sollen. Sind keine Daten definiert, erfolgt kein Schreiben. Der Byte-Positionszeiger wird auf den Wert von start gesetzt und als RC wird 0 zurückgegeben. Ist start auch nicht angegeben, wird der Positionszeiger auf EOF gesetzt.

Start

Byteposition, ab der geschrieben werden soll. Fehlt die Angabe, wird ab dem aktuellen Pointer geschrieben

Hinweis

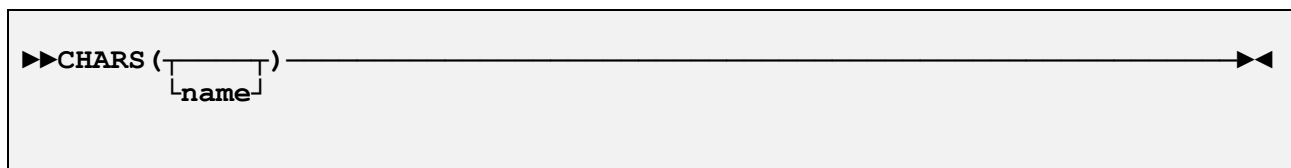
Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Es gibt eine Reihe von Fluchtsequenzen (ESC_?). ESC_N erzeugt eine neue Zeile. Siehe hierzu auch das IBM Manual "Using REXX and Unix System Services"

Beispiel

```
/* REXX
*/
call charout , 'I am the best' esc_n
```

CHARS() – UNIX Function



Liefert die Anzahl Byte, welche in einem Datenstrom noch zur Verarbeitung zur Verfügung stehen.

Name

Der Name eines Streams. Fehlt die Angabe, wird STDIN angenommen.

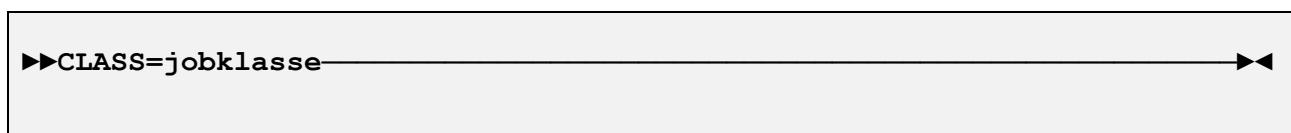
Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Beispiel

```
/* REXX
*/
File = 'unix_file_name'
/* nach diversen Zugriffen via CHARIN() */
Say "Noch offene Bytes:" chars(file)
```

CLASS – JCL JOB Parameter



Es können die Werte A bis Z und 0 bis 9 angegeben werden. Wenn für die Jobklasse kein Initiator verfügbar ist, wird der Job nicht ausgeführt.

COMMAND – JCL Special Statement

```
// [label] COMMAND 'command' -----><
```

Via COMMAND können MVS-Befehle innerhalb einer JCL-Verarbeitung ausgeführt werden. Voraussetzung hierzu ist, dass der Auftraggeber der JCL eine entsprechende Command-Ausführungsbeerechtigung hat. Ist dies nicht der Fall, wird das Kommando an der Operatorkonsole ausgegeben. Das Kommando muss in diesem Fall vom Operator über ein REPLY bestätigt werden.

Hinweise:

das COMMAND-Statement endet in Spalte 71 und wird auf Spalte 16 fortgesetzt.

Beispiel

```
Spalte 71-----|  
// COMMAND 'START ARM='ABC,GHI,JK',TIME=1440,REGI  
//          ON=4096K'  
Spalte 16-----|
```

```
// COMMAND 'S VTAM'  
Startet VTAM
```

```
// COMMAND 'SEND ''START Abrechnung'',USER(XYZ)'  
Apostrophe, die im Kommando vorkommen müssen doppelt kodiert werden.
```

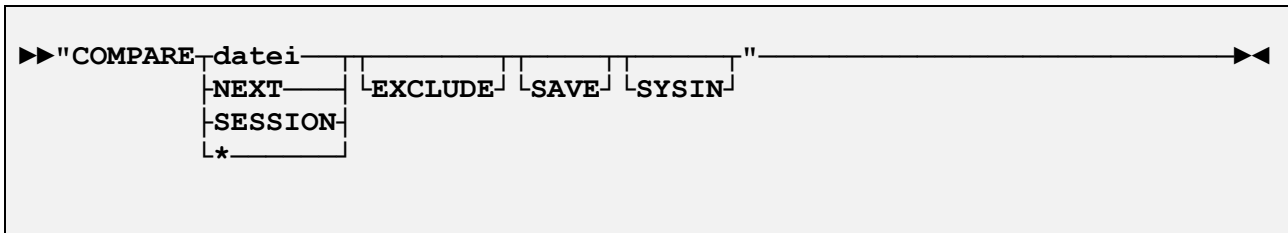
COMMIT – DB2 Instruction

```
>>"COMMIT [WORK]"-----><
```

Schließt eine Transaktion (auch "Unit of Work" genannt) ab und vollzieht die Änderungen im DB2.

Der Parameter WORK hat in DB2 keine Bedeutung, ist aber aus Kompatibilitätsgründen zu anderen SQL-Datenbanken erlaubt.

COMPARE – ISPF Edit Macro



Vergleicht die Daten, welche der Editor im Zugriff hält, gegen eine sequentielle Datei oder ein Member einer anderen oder der gleichen PO-Datei.

datei

Name des Members oder der Datei, welche für den Vergleich herangezogen wird.

Wird in einer PO-Datei editiert und als Vergleich eine andere PO-Datei angegeben, wird dort nach einem gleichnamigen Member gesucht.

NEXT

Bestimmt, dass der Vergleich mit einem gleichnamigen Member stattfinden soll, der in der Verkettungsreihenfolge als nächster gefunden wird.

SESSION | *

Führt einen Vergleich der im Editor gehaltenen Daten gegen die letzte Sicherung dieser Daten auf Platte durch.

EXCLUDE

Legt fest, dass alle übereinstimmenden Zeilen ausgeblendet werden (bis auf eine definierte Anzahl vor und nach unterschiedlichen Zeilen. Diese Anzahl ist im Edit-Compare-Settings-Panel festgelegt – Default = 0).

SAVE

Das Ergebnis soll in einer Datei abgelegt werden. Der Name dieser Datei lautet:

prefix.ISPFEDIT.COMPARE.LIST

SYSIN

Bewirkt, dass das SYSIN-DD-Statement bei Aufruf von SuperC nicht explizit freigegeben wird. Dies erlaubt, explizit Anweisungen für den SuperC auf Datei zu hinterlegen. Es erfolgt keine Prüfung, ob SYSIN richtig allokiert ist.

Beispiel

```

/* REXX - Edit macro
*/
Address isredit
"MACRO (PARM)"          /* Accept input string */
  
```

```

If parm ^= "" Then do      /* Do nothing if no parameters */
  "RESET LABEL"           /* Remove all existing labels */
  "F FIRST "parm          /* Find first string occurrence */
  Do While(rc=0)          /* For each occurrence */
    "LABEL .ZCSR = label()" 0" /* Assign label */
    "RFIND"               /* Find next occurrence */
  End
  "COMPARE X"             /* Exclude everything except */
                          /* Labels and above/below lines */
  "RESET LABEL"           /* Remove all labels */
  "(XSTAT) = XSTATUS .ZFIRST" /* Save satus of line 1 */
  "LOCATE .ZFIRST"        /* Move display to line 1 */
  "XSTATUS .ZFIRST = "xstat /* Restore line1 X-status */
End
Exit 0 /* Always return a zero */
/* ----- */
label:Procedure Expose labelnum /* generate a unique */
  If datatype(labelnum,'N') = 0 Then labelnum=0
  Else labelnum = labelnum + 1
Return translate(right(labelnum,4,'0'), ,
  "ABCDEFGHJIJ","0123456789")

```

Sucht einen String, blendet alle unmatched Lines aus und benutzt die COMPARE EXCLUDE Funktion, um vor und nach den Treffern eine Zeile sichtbar zu machen.

"COMPARE X.Y.Z SAVE"

Vergleicht die aktuellen Daten mit denen in der Datei X.Y.Z und schreibt das Ergebnis in die Datei ISPFEDIT.COMPARE.LIST.

Return Codes

- 0 Normales Ende
- 8 Member, Datei oder Unix File nicht gefunden oder Fehler beim OPEN der Datei
- 12 Kein Parameter angegeben oder Parameter-Fehler wie NEXT oder Memberangabe nicht möglich
- 20 Schwerer Fehler

COMPARE() – REXX Built-in-Function

```

▶▶COMPARE(string1,string2,füllzeichen)◀◀

```

Übergibt 0, wenn die zu vergleichenden Zeichenfolgen identisch sind. Wenn nicht, wird die Position des ersten nicht übereinstimmenden Bytes übergeben. Die kürzere Zeichenfolge wird rechts durch das Füllzeichen verlängert.

Ist kein Füllzeichen angegeben, wird BLANK als Füllzeichen benutzt.

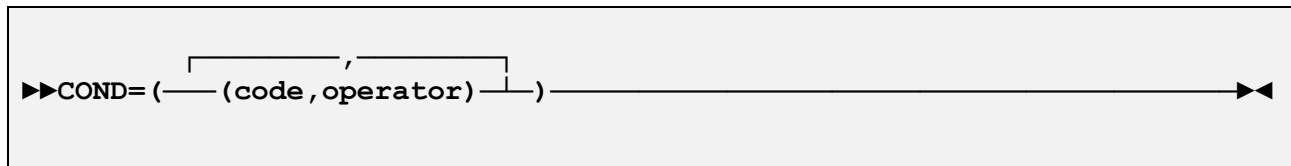
Beispiele

```

SAY COMPARE ('HUGO' , 'HUGO' )           0
SAY COMPARE ('HUGO' , 'HUBERT' )         3
SAY COMPARE ('HUGO' , 'HUGO' )           0
SAY COMPARE ('HUGO' , 'HUGO' , '-' )     5
SAY COMPARE ('HUGO--' , 'HUGO' , '-' )   7

```

COND – JCL JOB Parameter



Wenn eine Bedingung in der JOB-Anweisung angegeben wurde, gilt sie für alle Steps in einem Job (der JOB/STEP wird nicht ausgeführt, wenn die Bedingung zutrifft).

code

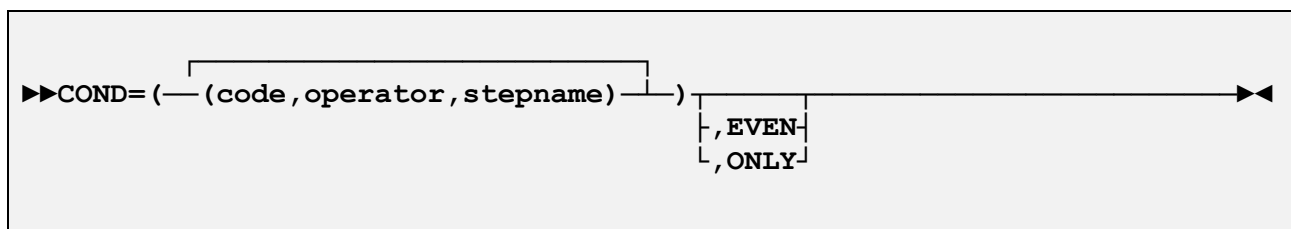
Als Code können die Werte 0 bis 4095 angegeben werden. Als Operator sind folgende Angaben möglich:

operator

EQ gleich (EQUAL)
NE ungleich (NOT EQUAL)
GT größer als (GREATER THAN)
GE größer oder gleich als (GREATER EQUAL)
LT kleiner als (LESS THAN)
LE kleiner oder gleich als (LESS EQUAL)

Der gelieferte Rückkehrcode eines Programms wird mit den Angaben im COND-Parameter verglichen. Wenn die Bedingung zutrifft, wird der JOB/STEP nicht ausgeführt.

COND – JCL EXEC Parameter



Es können maximal acht Bedingungen einschließlich EVEN und ONLY angegeben werden. EVEN und ONLY können nur alternativ benutzt werden. Wird nur EVEN oder ONLY angegeben, können alle Klammern entfallen. Wenn nur eine Bedingung angegeben wird, können die äußeren Klammern entfallen. Wird der Stepname nicht angegeben, entfällt auch das davorstehende Komma.

code

Als Code können die Werte 0 bis 4095 angegeben werden. Als Operator sind folgende Angaben möglich:

operator

EQ gleich (EQUAL)
NE ungleich (NOT EQUAL)
GT größer als (GREATER THAN)
GE größer oder gleich als (GREATER EQUAL)
LT kleiner als (LESS THAN)
LE kleiner oder gleich als (LESS EQUAL)

EVEN

Der Step wird in jedem Fall ausgeführt, auch wenn ein vorangegangener Step mit ABEND beendet wurde.

ONLY

Der Step wird nur dann ausgeführt, wenn ein vorangegangener Step mit ABEND beendet wurde.

Hinweise

Der bei Stepende von einem Programm gelieferte Rückkehrcode wird mit den Angaben im COND-Parameter verglichen. Wenn die Bedingung zutrifft, wird der Step nicht ausgeführt.

Die COND-Angabe in der JCL ist heute nicht mehr sinnvoll. Natürlich wird man sie noch viele Jahre in "alter" JCL sehen, sinnvoller und deutlich besser lesbar ist aber die IF-Instruktion. Zudem bietet sie sehr viel bessere Möglichkeiten zur Ablaufsteuerung.

Bedingungen im COND-Parameter	RC aus vorangegangenem Step	
	Ausführung	Unterdrückung
COND=(code,GT)	RC >= code	RC < code
COND=(code,GE)	RC > code	RC <= code
COND=(code,EQ)	RC <> code	RC = code
COND=(code,LT)	RC <= code	RC > code
COND=(code,LE)	RC < code	RC >= code
COND=(code,NE)	RC = code	RC <> code

Beispiele

COND= (8 , LT)

COND= (9 , LE)

Ein Step soll nur laufen, wenn vorhergehende Steps einen Rückkehrcode kleiner oder gleich 8 liefern.

COND= ((0 , NE , STEP1) , EVEN)

Der Step soll nur laufen, wenn in einem vorangegangenen STEP1 der Rückkehrcode 0 geliefert wird, wobei die Prüfung auch bei ABEND durchgeführt wird.

Subparameter EVEN/ONLY

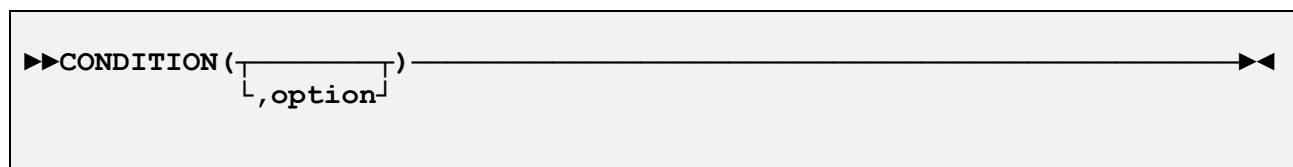
Angabe	ABEND vorausgegangen	RC-Test erfüllt	Ausführung
EVEN	Nein	Nein	Nein
EVEN	Nein	Ja	Nein
EVEN	Ja	Nein	Ja
EVEN	Ja	Ja	Nein
ONLY	Nein	Nein	Nein
ONLY	Nein	Ja	Nein
ONLY	Ja	Nein	Ja
ONLY	Ja	Nein	Nein
Keine	Nein	Nein	Ja
Keine	Nein	Ja	Nein
Keine	Ja	Nein	Nein
Keine	Ja	Ja	nein

Beispiel

COND=ONLY

Der Step soll nur dann ausgeführt werden, wenn bei irgendeinem vorangegangenen Step ein ABEND aufgetreten ist.

CONDITION() – REXX Built-in-Function



Liefert eine Information zum Trapping (Error-Condition o.ä.). Möglich sind folgende Informationen:

- Name der aktuellen Trapped Condition
- Die Instruktion für das Trapping (CALL oder SIGNAL)
- Der Status der Trapped Condition

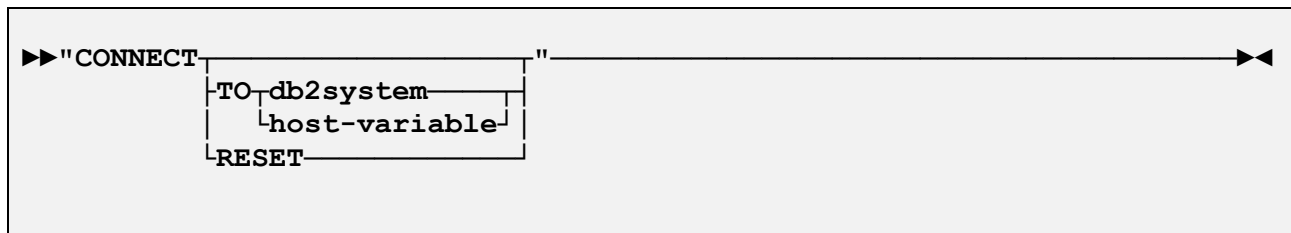
Folgende Optionszeichen sind möglich (UpperCase zwingend):

Option	Bedeutung	Rückgabe
C	Condition name	Name der Trapped Condition
D	Description	Erklärung
I	Instruction	CALL oder SIGNAL
S	Status	Status der Trapped Condition, Dies kann sein ON, OFF oder DELAY

Beispiele

```
SAY CONDITION('I')      'CALL'
SAY CONDITION('C')      'NOVALUE'
SAY CONDITION('S')      'ON'
```

CONNECT – DB2 Instruction



CONNECT stellt eine Verbindung zu einem DB2-System her. Ohne Parameter wird vom CONNECT folgendes veranlasst: Offene Cursor werden nicht geschlossen. Keine Veränderung irgendwelcher Stati. Informationen werden in die SQLCA gestellt.

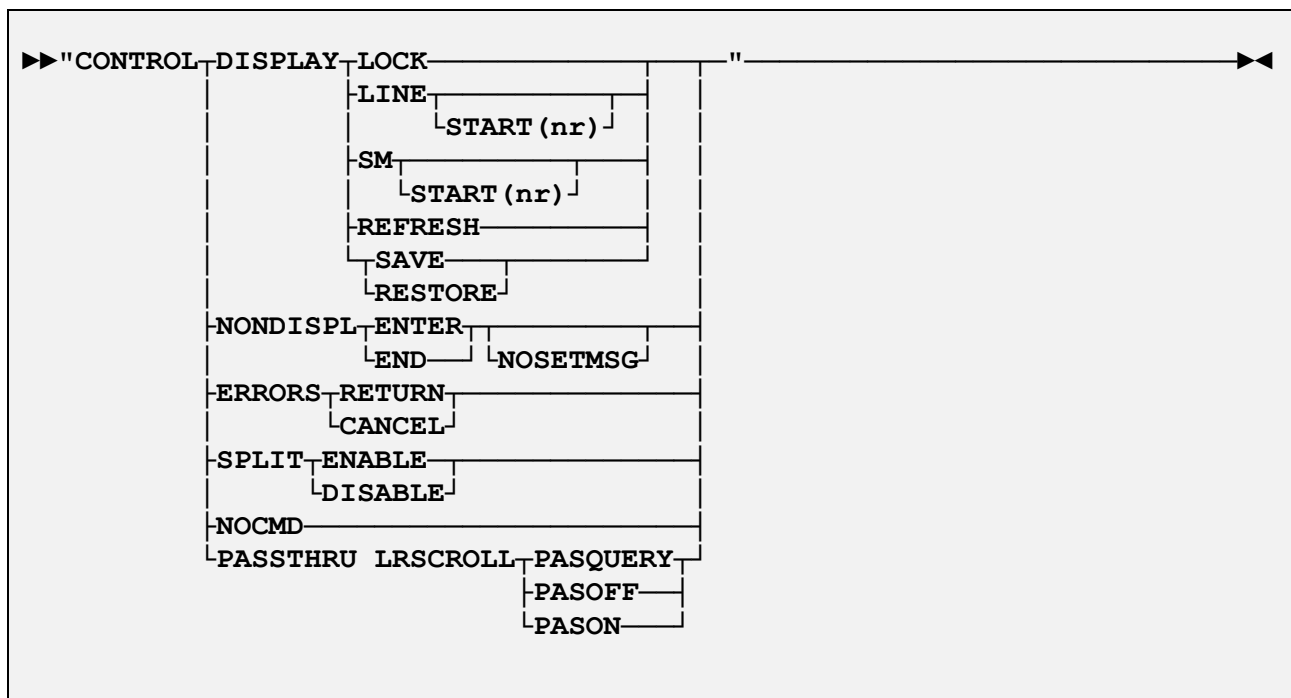
TO db2system

Verbindung zum angegebenen System. Das DB2-System kann über eine Konstante oder eine Variable geliefert werden. Bedingung für den Connect ist, dass das angegebene System in der LOCATIONS-Table eingetragen ist.

RESET

Entspricht einem CONNECT TO

CONTROL – ISPF DM Service



Die Control-Anweisung steuert die Behandlung von Fehlern und die Art der Bildschirmanzeige.

DISPLAY

Steuert den Anzeigemodus. Die möglichen Angaben sind:

LOCK

Bestimmt, dass für das nächste (und nur das) DISPLAY oder TBDISPL das Terminal gesperrt ist. Dies wird für verstreute Meldungen genutzt. ISPF zeigt das nächste DISPLAY, als hätte der Benutzer Die ENTER-Taste gedrückt.

LINE

legt fest, dass der Zeilenmodus aktiv ist. Der FULL-SCREEN-Modus wird beim nächsten ISPF-FULL-SCREEN-Service wieder aktiv und der gesamte Bildschirminhalt wird neu ausgegeben.

Der Zeilenmodus wird automatisch aktiviert, wenn in einer SELECT-Anweisung bei CMD() ein Eintrag erfolgt, außer, der Eintrag beginnt mit einem führenden Prozentzeichen(%).

START(nr)

Legt die Nummer der Bildschirmzeile fest (die oberste Zeile hat die Nummer 1), von der ab der Zeilenmodus aktiv werden soll. Von dieser Zeile ab wird der Bildschirm bis zum unteren Rand gelöscht. Wird die Zeilennummer nicht angegeben, wird als Standard UNTERER TERMINALRAND - 3 angenommen. Die Angabe dieses Operanden ist nur sinnvoll im LINE-Modus. Im SM-Modus kann der Wert angegeben werden, da unter Umständen in den LINE-Modus gewechselt wird, wenn der Session Manager nicht aktiv ist.

SM

Aktiviert den TSO-Session-Manager (ursprünglich zeilenorientierte Befehle können im FULL-SCREEN-Modus eingegeben werden). Kann der Session Manager nicht geladen werden, wird in den Zeilenmodus umgeschaltet.

REFRESH

bewirkt, dass bei der nächsten FULL-SCREEN-Operation von ISPF der gesamte Bildschirm neu aufgebaut wird. Dieser Service sollte immer genutzt werden, wenn ein Programm Bildschirmausgaben ohne die dazu von Dialog Manager zur Verfügung gestellten Mittel durchführt.

SAVE / RESTORE

Save wird genutzt in Verbindung mit DISPLAY, TBDISPL, BROWSE und EDIT, um Informationen über den aktuell bearbeiteten Bildschirm zu sichern.

Durch RESTORE werden die abgespeicherten Werte wieder verfügbar. Die Angaben SAVE und RESTORE müssen immer in Verbindung paarig auftreten.

Eine mögliche Anwendung ist bei TBDISPL die Abarbeitung der Tabellenzeilen über ZTDSELS, bei der aus Sicherheitsgründen zu Beginn der Einzelverarbeitung gesichert, am Ende wieder rückgespeichert werden sollte.

NONDISPL

Steuert, dass die nächste Bildschirmausgabe unterdrückt wird. Für die übernächste und alle weiteren Bildschirmausgaben erfolgt wieder normale Verarbeitung.

ENTER

simuliert das Drücken der ENTER-Taste.

END

simuliert das Drücken der END-Funktion.

NOSETMSG

Eine eventuell definierte Meldung wird für das nächste Panel unterdrückt

ERRORS

steuert, wie Fehler innerhalb der ISPF-Service behandelt werden.

CANCEL

bewirkt, dass der gesamte Dialog beendet wird, sobald ein Fehlercode > 12 von einem ISPF-Service gemeldet wird. In diesem Fall wird eine Meldung in das ISPF-LOG-File geschrieben.

RETURN

regelt, dass bei jedem beliebigen Return Code die Steuerung an den Dialog zurückgegeben wird. Die Fehlersituation kann somit in der aufrufenden Funktion abgeklärt werden, die Entscheidung über den weiteren Verlauf bleibt bei der Funktion.

SPLIT

steuert, ob ein Benutzer in den SPLIT-SCREEN-Modus verzweigen darf oder nicht.

ENABLE

bewirkt, dass der Benutzer den SPLIT-SCREEN-Modus aufrufen kann. Läuft die Verarbeitung im BATCH ab, wird bei SPLIT ENABLE ein schwerer Fehler (RC=20) ausgegeben, da hier der SPLIT-Service nicht verfügbar ist.

DISABLE

bestimmt, dass der SPLIT-SCREEN-Modus nicht durch den Benutzer eingestellt werden kann. Beendet sich der Benutzer bereits im SPLIT-SCREEN-Modus, wird ein RC 8 ausgegeben und der SPLIT-Modus auf ENABLE gesetzt.

NOCMD

legt fest, dass für den nächsten angezeigten Bildschirm jedes Kommando, das in der Kommandozeile eingegeben wird, oder jeder Befehl, der über eine PF-Taste ausgeführt werden soll, ignoriert wird.

PASSTHRU LRSCROLL

Steuert generell, wie mit Left-/Right-Scrolling verfahren wird.

PASQUERY

Liefert eine Aussage über RC. 0 = Left/Right wird nicht an die Funktion durchgereicht, 1 = wird durchgereicht.

PASOFF

Left/Right soll nicht an die Funktion durchgereicht werden.

PASON

Left/Right soll an die Funktion durchgereicht werden.

Return Codes

- 0 Normales Ende
- 8 Splitscreen Modus bereits aktiv. Die Meldung kommt nur in Verbindung mit SPLIT DISABLE
- 20 Schwerer Fehler

COPIES() – REXX Built-in-Function

```
►►COPIES(string,anzahl)◄◄
```

Übergibt aufeinanderfolgende Kopien von string. Anzahl muss eine positive Zahl sein.

Beispiele

```
SAY COPIES('Achtung ',3)      'Achtung Achtung Achtung '  
SAY COPIES('TRALLALA',0)    ''
```

COPY – ISPF Edit Macro

```
►►"COPY member[AFTER|BEFORE]zz[bereich]"◄◄
```

COPY kann ein Member einer PO-Datei in die aktuellen Daten importieren.

member

Name des Member der PO-Datei, das importiert werden soll.

AFTER | BEFORE

Zielort der zu importierenden Daten in Bezugnahme auf einen Zeilenzeiger.

zz

Zeilenzeiger (Linepointer) als Zielort. Die importierten Daten können vor oder hinter die entsprechende Zeile kopiert werden. Zeilenzeiger kann ein Label oder eine relative Zeilennummer sein.

bereich

Von/bis-Zeilennummernangaben des Importmembers, sofern nicht alle Zeilen importiert werden sollen.

Beispiele

```
"copy adr1 after .zlast"
```

Fügt den Inhalt des Member ADR1 an das Ende der Daten an.

```
"copy adr1 before .zfirst 1 8"
```

Kopiert die ersten acht Zeilen des Members ADR1 an den Anfang der Daten.

Return Codes

0	Normales Ende
8	EOF erreicht, bevor der letzte Satz gelesen wurde
12	Falscher Linepointer oder Member nicht gefunden oder BLDL Fehler
16	EOF vor dem Lesen des ersten Datensatzes
20	Syntax Fehler (falscher Name oder unvollständiger Bereich oder I/O Error)

CREATE – ISPF Edit Macro

```
"CREATE member von bis"_____▶◀
```

Exportiert Daten des Editors und erzeugt damit ein neues Member. Existiert das zu erzeugende Member bereits, ist der Befehl REPLACE zu benutzen.

member

Name des zu erstellenden Members.

von bis

legen den Bereich fest, der zur Erstellung des neuen Members benutzt werden soll. Diese Angaben können entweder in Form von Label oder relativen Zeilennummern geliefert werden und müssen immer paarig sein.

Beispiele

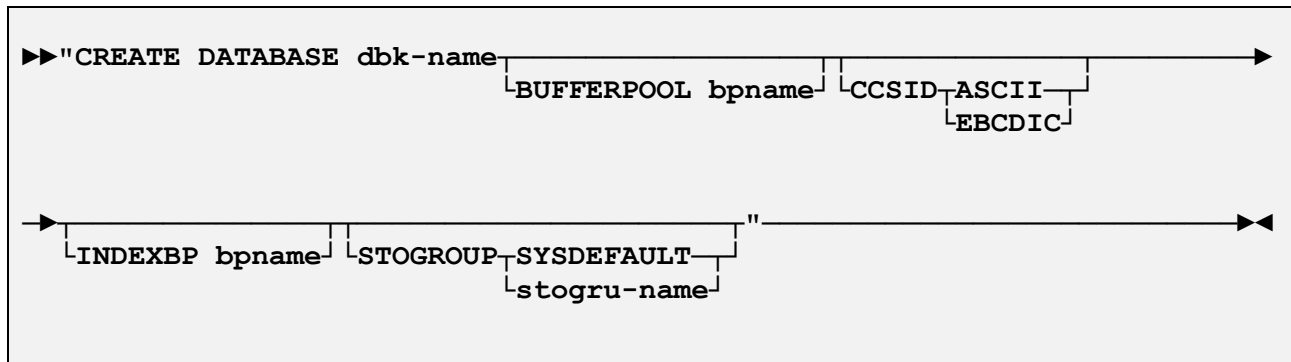
```
"create adrneu 1 10"
```

Das Member ADRNEU wird mit den ersten zehn Zeilen der Edit-Daten erzeugt.

Return Codes

- 0 Normales Ende
- 8 Member bereits vorhanden
- 12 Falscher Linepointer
- 20 Schwerer Fehler

CREATE DATABASE – DB2 Instruction



Erzeugt eine Datenbank auf dem aktuellen Server.

Dbk-name

Name der zu erstellenden Datenbank

BUFFERPOOL bpname

Der Default Name des Buffer Pools, welcher auch bei Anlage eines Tablespace herangezogen wird.

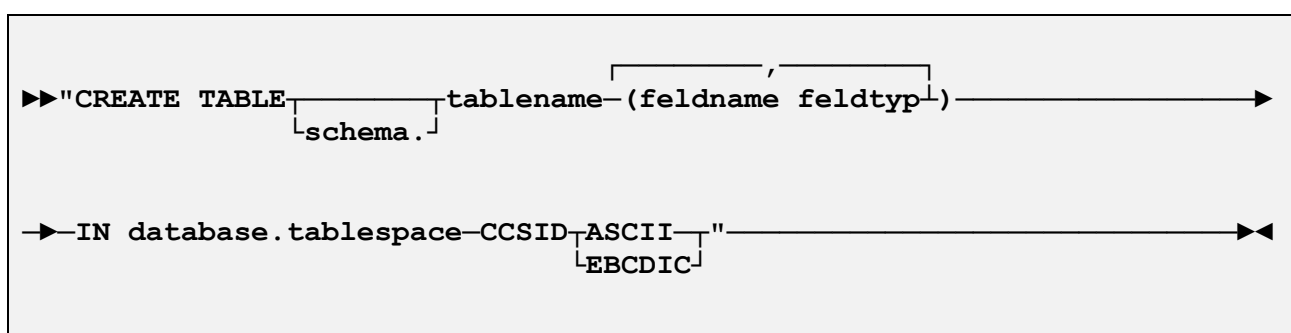
CCSID(ASCII | EBCDIC)

Code Character Set Identifier. Bestimmt, nach welchen Regeln Character-Daten behandelt werden.

INDEXBP bpname

Wird bei einem CREATE INDEX berücksichtigt.

CREATE TABLE – DB2 Instruction



Erzeugt eine Tabelle innerhalb einer Datenbank und legt die Namen und Charakteristiken für Tabelle und Spalten fest.

Schema

Ist die logische Zusammenfassung von Objekten in einer Datenbank. Ein Schema wird benutzt, wenn man mehrere gleichnamige Datenbanken (oder Tabellen) benötigen würde (was aber nicht möglich ist), um beispielsweise Daten aufzusplitten um parallele Verarbeitung durchführen zu können.

Ist kein Schema angegeben, wird in der Regel die UserID als Schemaname angenommen.

tablename

Name der anzulegenden Tabelle

fieldname

Name des Tabellenfeldes

feldtyp

Merkmale des Tabellenfeldes. Hier einige Beispiele (für die vollständige Beschreibung aller Feldtypen bemüht bitte die DB2-Dokumentation der IBM).

Typ	Bedeutung
CHAR(xx)	Feld beliebigen Inhalts in der angegebenen Länge (1 bis 255 Byte)
VARCHAR	Character Feld in variabler Länge
DEC(xx[,yy])	Dezimal mit Angabe der Vor- und Nachkommastellen
DATE	Datum im Format YYYYMMDD (kann extern abweichen)
TIME	Zeit im Format HHMMSS (kann extern abweichen)
TIMESTAMP	Zeitstempel im Format YYYYMMDDHHMMSSTTTTT (kann extern abweichen)
INT	31Bit Binärzahl + Vorzeichen (intern 4 Byte)
SMALLINT	15Bit Binärzahl + Vorzeichen (intern 2 Byte)
ROWID	Eindeutiges Erkennungsmerkmal einer Tabellenzeile. Kann in einer Tabelle nur EINMAL existieren und ist durch DML nicht veränderbar

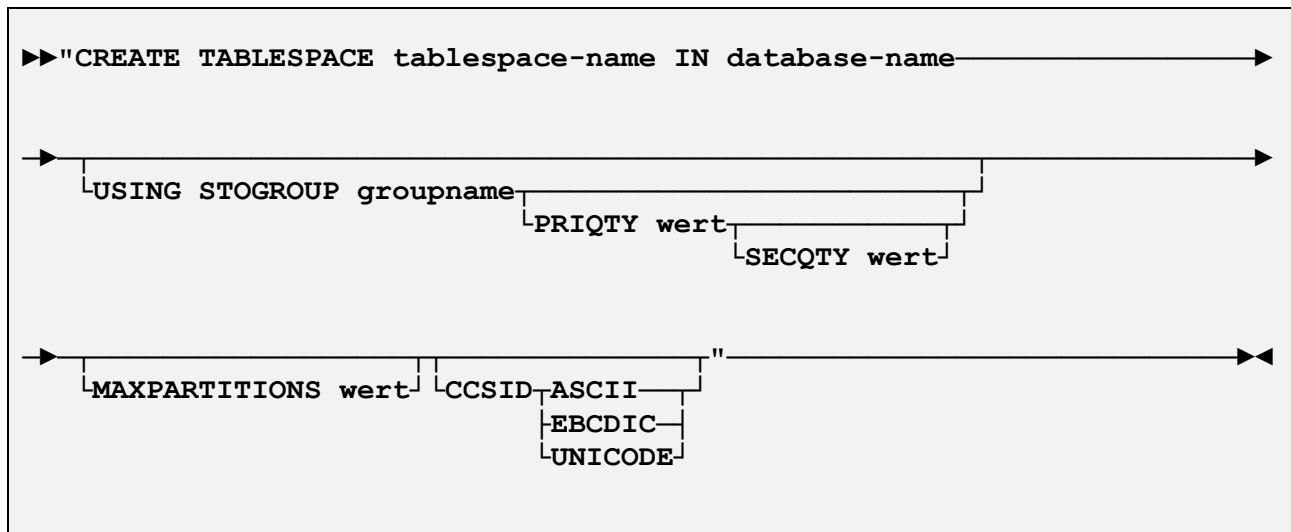
IN

Benennt Datenbank und Tablespace, in welchem die Tabelle angelegt werden soll.

CCSID(ASCII | EBCDIC)

Legt die Regeln fest wie Character-Daten behandelt werden sollen.

CREATE TABLESPACE – DB2 Instruction



Erstelle einen Tablespace innerhalb einer Datenbank

Tablespace-name

Name des Tablespace

IN database-name

Name der Datenbank innerhalb derer der Tablespace angelegt werden soll.

USING STOGROUP groupname

Legt die Storage Group im DB2 fest. Sprecht hierzu mit Ihrem DB-Verantwortlichen.

PRIQTY wert

Bestimmt die Primärgröße des DB2-managed Datasets. -1 benutzt den definierten Defaultwert.

SECQTY wert

Bestimmt die Sekundärgröße des DB2-managed Datasets. -1 benutzt den definierten Defaultwert.

MAXPARTITIONS wert

Bestimmt, ob der Tablespace gegebenenfalls aus mehreren Partitionen bestehen soll.

CCSID(ASCII | EBCDIC | UNICODE)

Regent den Umgang mit Datenfeldern.

CTL_LIBRARY – ISPF Edit Macro

```
"(var1,var2) = CLT_LIBRARY"—————▶◀
```

Übergibt den Status einer kontrollierten Bibliothek. Wird im Allgemeinen in Initial-Makros kodiert, um die Nutzung kontrollierter PO-Member zu regeln.

Controlled Libraries sind heute eine Ausnahme. Deshalb seien hier nur die wichtigsten Informationen beschrieben (Details siehe Manual Edit Makros).

var1 var2

LOCK-Status des Members. Die möglichen Inhalte sind:

Var1	Var2
OBTAINED	UserID dessen, der das Member belegt
UNAVAILABLE	UserID / DEACTIVATED / LOCKED
ERROR	Blanks
NOCHECK	FIRSTLIB oder Blanks

OBTAINED

Member in kontrollierter Bibliothek gefunden, wird gesperrt. Wird bei nächster Ausführung als LOCK-Status NOCHECK ausgegeben. Wird nicht gesichert, wird der LOCK-Status aufgehoben.

UNAVAILABLE

Das Member wurde in einer kontrollierten Bibliothek gefunden und ist gesperrt. Var2 kann einen der folgenden Werte enthalten:

Wert	Bedeutung
USERID	Benutzer, welcher das Member sperrt
DEACTIVATED	Bibliothekskontrolle wurde deaktiviert
*LOCKED	Member verfügbar, existiert aber in einer tieferen Strukturebene, die nicht der Edit-Dateiverkettung vorgeschaltet ist (Pseudo-Lock).

Beispiele

```
"(zstat,sstat) = ctl_library"
```

Die Stati für Zugriff und Sperrung werden den Variablen übergeben.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

CURSOR – ISPF Edit Macro

```

▶▶"(var1,var2) = CURSOR"
▶▶"CURSOR = zeile spalte"

```

Übergibt die Zeilen-/Spaltenkoordinate in der sich der Cursor befindet, oder setzt den Cursor entsprechend. Die Cursorposition zeigt Auswirkungen bei FIND, SEEK, CHANGE, EXCLUDE und TSPLIT.

=CURSOR

Die Position des Cursors wird ausgelesen

var1

Sechsstellig numerisch mit führenden Nullen mit der relativen Zeilennummer, in der sich der Cursor befindet.

var2

Spaltennummer, dreistellig mit führenden Blanks. Steht der Cursor in der Kommandozeile, wird 1 übergeben, sitzt er im Zeilenkommandofeld, wird 0 ausgegeben.

Beide Werte sind auf 0 gesetzt, wenn die Information bei einem leeren Member eingeholt wird.

CURSOR=zeile spalte

Zeilen-/Spaltenkoordinate, auf die der Cursor gesetzt werden soll.

Liegt die Spaltenangabe außerhalb der Boundaries, wird der Cursor an die erste Stelle des Zeilenkommandofeldes der nächsten Zeile gesetzt.

Wird der Cursor positioniert, sollte sichergestellt sein, dass die angegebene Zeilennummer auch wirklich existiert.

Zeilen- /Spaltennummern, abhängig von der aktuellen Cursorposition.

Cursor steht in	Zeile	Spalte
COMMAND➔	Oberste sichtbare	0
Zeilennummernfeld	Zeile des Cursors	0
Linke LineSequenceNumber	Zeile des Cursors	0
Rechte LineSequenceNumber	Zeile des Cursors	0
Datenteil	Zeile des Cursors	Spalte des Cursors
Zeile ohne Zeilennummer	Zeile über Cursor	Spalte des Cursors
LineCMD-Bereich-Hinweiszeile	Zeile unter Cursor	0

Beispiele

```
"(zeile) = cursor"
```

In der Variablen ZEILE wird die Zeilenposition des Cursors hinterlegt.

```
"cursor = .beginn"
```

Der Cursor wird in die Zeile des Labels .BEGINN gesetzt, ohne dabei seine Spaltenposition zu ändern.

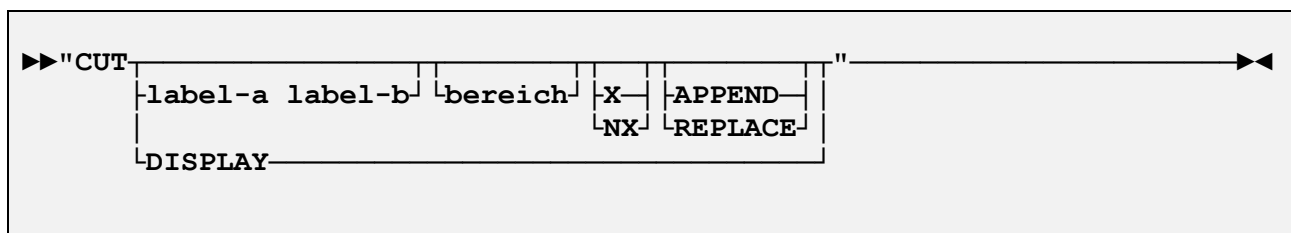
```
"cursor = 1 1"
```

Setzt den Cursor in Spalte 1 der ersten Zeile.

Return Codes

0	Normales Ende
4	Spalte außerhalb des Datenbereichs, Zeilennummer wird erhöht
12	Falsche Zeilennummer
20	Schwerer Fehler

CUT – ISPF Edit Macro



CUT kopiert eine Anzahl Zeilen in einen Zwischenbereich.

=label-a label-b

Die zu kopierenden Zeilen können über selbstdefinierte Labels oder relative Zeilennummern definiert werden. Wird kein Bereich angegeben ist der Standard .ZFIRST und .ZLAST (von der ersten bis zur letzten Zeile).

bereich

Ist der Name der Zwischenablage, in dem die Daten abgelegt werden. Ist kein Bereichsname angegeben, wird DEFAULT angenommen.

X | NX

Kopiert nur excluded oder nicht-excluded Lines. Default ist sowohl als auch.

APPEND

Schreibt den Zwischenbereich fort.

REPLACE

Überschreibt den Zwischenbereich.

DISPLAY

Ruft den Clipboard-Manager auf. In wie weit das in einem Macro Sinn macht, ist fraglich 😊

Beispiele

```
"cut .zfirst .zlast origin replace"
```

Kopiert alle Daten in die Zwischenablage unter dem Namen ORIGIN. Sofern dieser Bereich bereits Daten enthält, werden diese überschrieben.

Return Codes

0	Normales Ende
12	Parameter Fehler oder Speicherüberlauf oder keine weiteren Zwischenablagen verfügbar
20	Schwerer Fehler

C2D() - Character to Decimal – REXX Built-in-Function

►►C2D(string) —————►◄

Wandelt eine Zeichenfolge in Dezimal um. Ist string eine Leerzeichenfolge, wird 0 zurückgegeben.

Der Eingabestring darf maximal 250 Byte lang sein.

Beispiele

```
SAY C2D('03'X)           3
SAY C2D('81'X)           129
SAY C2D('FF81'X)        65409
SAY C2D('a')            129      EBCDIC
```

C2X() – Character to Hex – REXX Built-in-Function

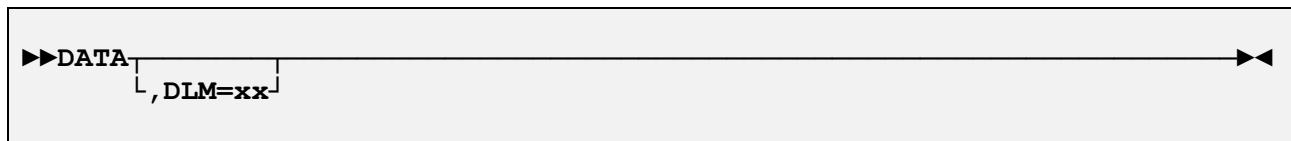
►►C2X(string) —————►◄

Übersetzt einen Character-String in seine hexadezimale Bedeutung. Die Länge des Eingabestrings ist beliebig.

Beispiele

```
SAY C2X('38s')          'F3F8A2'      EBCDIC
SAY C2X('327'X)          '327'
```

DATA – JCL DD Parameter



DATA

Die DD-Data-Anweisung beschreibt Instream-Daten, welche aber mit Slashes auf Stelle 1 und 2 beginnen dürfen.

Als Ende der Daten ist /* zwingend zu kodieren.

,DLM=xx

Ist "/*" auf Stelle 1 und 2 ebenfalls Bestandteil der Daten, kann ein beliebiges Endemerkmale gesetzt werden (seit zOS 2.5 bis 18 Byte Länge)

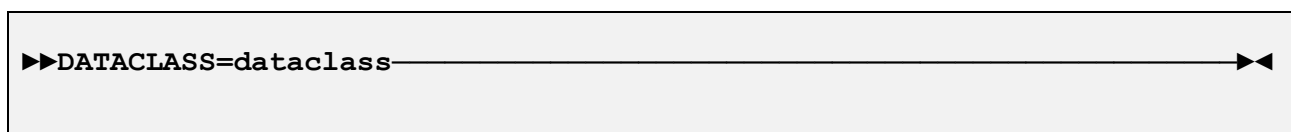
Beispiel

```
//SUBMIT EXEC PGM=IEBGENER,COND=(0,NE,stepname)
//SYSUT1 DD DATA,DLM=MYEND
//SYSUT2 DD SYSOUT=(A,INTRDR)
//jobname JOB job-card-operanden
//stepname EXEC PGM=progname
// DD
```

MYEND

Bei RC 0 eines vorangegangenen Steps wird eine Folgeverarbeitung an den Internal Reader übergeben.

DATACLASS – JCL DD Parameter



Diese Angabe wird nur in Verbindung mit SMS unterstützt. Ist SMS nicht installiert oder nicht aktiv, wird die Angabe ignoriert. Durch SMS wird die Angabe ignoriert, sofern es sich dabei um eine bereits katalogisierte oder nicht von SMS gemanagte Datei handelt.

Eine Datenklasse regelt für eine Datei Dinge wie: Satzlänge, Satzformat, Dateityp, Sicherungsfrist und dergleichen.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

DATA_CHANGED – ISPF Edit Macro

```
►►" (variable) = DATA_CHANGED"◄◄
```

Wird die Variable abgefragt, kann erkannt werden, ob Daten verändert wurden, oder nicht.

variable

Enthält entweder den Wert YES oder NO, abhängig davon, ob Daten verändert wurden, oder nicht. NO wird zu Beginn der Editsitzung, sowie nach jedem SAVE-Befehl gesetzt.

Beispiele

```
" (updated) = DATA_CHANGED"  
if updated = "YES"  
    then /änderungen */  
    ELSE /*keine veränderung */
```

Lässt erkennen, ob Änderungen durchgeführt wurden.

Return Codes

0	Normales Ende
20	Schwerer Fehler

DATA_WIDTH – ISPF Edit Macro

```
►►" (variable) = DATA_WIDTH"◄◄
```

Erkennt die Datenbreite einer Datei. Ist NUMBER OFF gesetzt, entspricht der ermittelte Wert der Satzlänge. Ist der Nummernmodus aktiv, gilt:

Nummerntyp	Datenbreite
STD	Satzlänge minus 8
COBOL	Satzlänge minus 6
STD COBOL	Satzlänge minus 14

Beispiele

```
" (max) = data_width"  
address isredit find "variable" 1 "max"  
Der FIND-Befehl sucht zwischen den abgegebenen Spalten.
```

Return Codes

0	Normales Ende
20	Schwerer Fehler

DATAID – Edit Macro

```
►►" (variable) = DATAID"◄◄
```

Hinterlegt in der Variablen den DD-Namen, unter dem die Datei vom Editor angesprochen wird. Wird der Name nicht bei Aufruf des Editors angegeben, wird der DD-Name vom LMINIT-Service festgelegt.

variable

Enthält den DD-Namen der aktiven Datei unter Edit.

Beispiele

```
"(dd) = DATAID"  
address ispxec "lmmfind dataid(&dd) member(adr1)"  
if rc = 0 ,  
    then /* Member gefunden */  
    else /* Member nicht gefunden */
```

Erkennt über LMMFIND, ob das Member ADR1 vorhanden ist.

Return Codes

0	Normales Ende
20	Schwerer Fehler

DATASET – ISPF Edit Macro

```
►►" (variable) = DATASET"◄◄
```

Ermittelt den Namen der im Edit aktiven Datei.

variable

Enthält den vollqualifizierten Namen der aktiven Datei des Editors ohne Quotes.

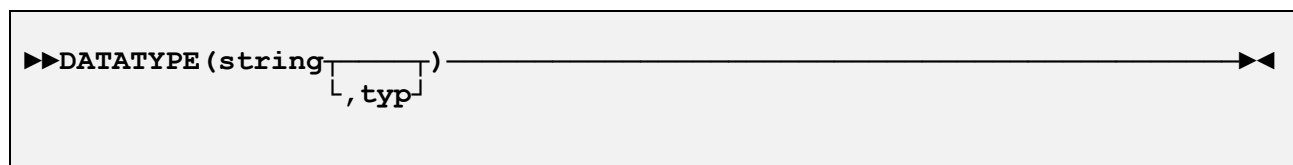
Beispiele

```
"(datei) = DATASET"
parse var datei prefix '.'
if prefix = userid(),
    then /* eigene Datei */
    else /* fremde Datei */
```

Erkennt, ob die Datei dem Anwender gehört, oder nicht.

Return Codes

0	Normales Ende
20	Schwerer Fehler

DATATYPE() – REXX Built-in-Function

Ist nur ein String angegeben, erfolgt von REXX eine Rückmeldung von CHAR (bei Datentyp Charakter) oder NUM (bei numerisch).

Wird typ angegeben wird von REXX mit einer 1 im Falle JA, ansonsten mit 0 geantwortet. Als typ sind folgende Angaben möglich:

TYP	1 wird zurückgemeldet bei
A	Alphanumeric, A-Z, a-z und 0-9
B	Bits, Nullen und Einsen
L	LowerCase (Kleinbuchstaben)
M	MixedCase (Groß- und Kleinbuchstaben)
N	Numbers (Ziffern)
S	Symbol (A-Z, 0-9 und @\$%.!?)
U	UpeprCase (Großbuchstaben)
W	Whole number (Zahl integer)

Beispiele

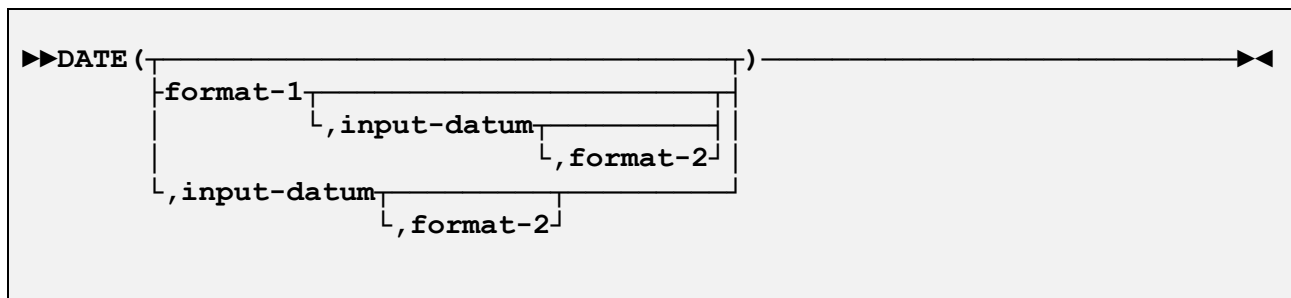
SAY DATATYPE(' 36 ')	NUM
SAY DATATYPE(' ')	CHAR
SAY DATATYPE('352\$')	CHAR
SAY DATATYPE('45.5', "N")	1
SAY DATATYPE('August', "L")	0
SAY DATATYPE('August', "M")	1
SAY DATATYPE('August', "U")	0
SAY DATATYPE('a01f', X)	1
SAY DATATYPE('A 01f', X)	1
SAY DATATYPE('test#2', "S")	1

```

/* REXX */
say 'Alter angeben'
do forever
  pull alter
  if datatype(alter,"W") then leave
  say 'Eingabe falsch, bitte wiederholen'
end

```

DATE() – REXX Built-in-Function



Liefert vielfältige Informationen und Funktionalitäten zum Datum.

Formate

Format	Bedeutung
B	Basedate (Tage seit dem 1. Januar 0001)
C	Century (Tage seit dem 1. Januar 2000)
D	Days (Der laufende Tag des Jahres)
E	European (Datum im Format tt/mm/jj)
J	Julian (Datum im Format jjtt)
M	Month (Ausgeschriebener Monatsname in Englisch)
N	Normal (Entspricht dem Standard wenn nichts angegeben ist. Z.B.: 5 Oct 2021)
S	Sortable (jjjjmmmtt)
U	USA (US-Amerikanisch mm/tt/jj)
W	Weekday (Englische Schreibweise des Wochentags)

Beispiele

```

SAY DATE()           '5 Oct 2009'
SAY DATE('B')       733105
SAY DATE('E',date('B')+1,'B') 06/10/09 /* morgen */
SAY DATE('E')       05/10/09'
SAY DATE('N')       '5 Oct 2009'
SAY DATE('S')       '20091005'
SAY DATE('W')       'Monday'

```

Hinweis

Bei Zugriffen auf Datum oder Uhrzeit wird gewährleistet, dass bei mehrfacher Verwendung dieser Werte in einem Ausdruck Datum oder Zeit nicht voneinander abweichen.

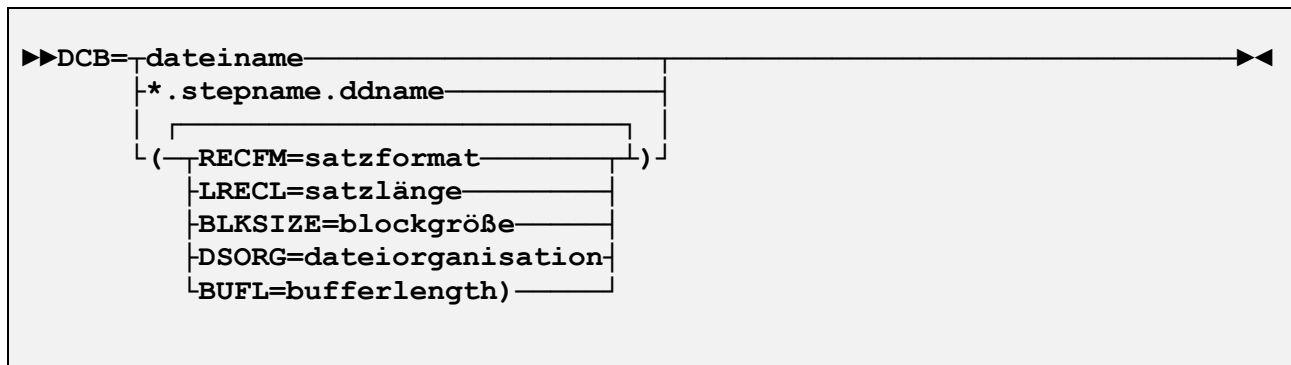
DBCS() – REXX Built-in-Function

Die Double-Byte-Character-Set Funktionen stellen einen sehr exotischen Teil von REXX dar. Bitte lest diese im IBM-Manual TSO/E-REXX Reference, Appendix A, "Double-byte character set(DBCS) support," nach.

Namentlich sind dies die Funktionen:

DBADJUST	DBCJUSTIFY	DBRLEFT	DBTOSBCS	DBWIDTH
DBBRACKET	DBLEFT	DBRRIGHT	DBUNBRACKET	
DBCENTER	DBRIGHT	DBTODBCS	DBVALIDATE	

DCB – JCL DD Parameter



Beim OPEN einer Datei werden die Angaben des DCBs (Dateisteuerblock) durch die hier gemachten Angaben ergänzt. Voraussetzung ist, dass der jeweilige DCB-Parameter nicht innerhalb des Programmes angegeben wurde.

Wird eine katalogisierte Datei gelesen, kann auf die DCB-Angabe verzichtet werden, da diese aus dem Katalog entnommen werden können.

Wird ein NO-LABEL-Band gelesen, müssen die DCB-Angaben durch die JCL übergeben werden, außer, sie werden im Programm direkt angegeben.

Bei Erstellung einer Datei müssen die DCB-Werte in der JCL angegeben werden (Ausnahme sind die MVS-Utilities wie IEBGENER. Wird hier keine DCB-Angabe für die Ausgabe gemacht, wird der DCB der Eingabe übernommen.

DCB=dateiname

Die DCB-Angaben werden von dieser Datei (muss katalogisiert sein) übernommen.

DCB=*.stepname.ddname

Mit dieser Angabe kann auf die DCB-Angaben einer Datei Bezug genommen werden, die unter ddname beschrieben ist. Die Bezugnahme auf ein GDG ist nicht erlaubt.

RECFM=satzformat

Die möglichen Satzformate sind:

Satzformat	Bedeutung
F	Feste Länge
FB	Feste Länge, geblockt
FBA	Feste Länge, geblockt mit ANSI-Steuerzeichen
V	Variabel
VB	Variable Länge, geblockt
VBA	Variable Länge, geblockt mit ANSI-Steuerzeichen

Gängige ANSI-Angaben bei Drucksteuerung:

Spalte 1	Bedeutung
Blank	1 Zeile Vorschub
0	2 Zeilen Vorschub
-	3 Zeilen Vorschub
+	Kein Zeilentransport
1	Vorschub auf Seitenanfang

RECFM= kann unter SMS auch außerhalb der DCB-Angabe kodiert werden.

LRECL=satzlänge

Anzahl Byte, die zu einem Datensatz gehören (z.B.: bei einer Adressdatei die Anzahl Byte, die eine Adresse bilden). Bei Übernahme von Datensätzen von einer in eine andere Datei ist darauf zu achten, dass die Satzlänge der übernehmenden Datei gleich oder größer der Übergebenden ist, da sonst Daten verloren gehen. LRECL= kann unter SMS auch außerhalb von DCB kodiert werden.

BLKSIZE=blocklänge

Anzahl Byte, die zu einer physikalischen Einheit zusammengefasst werden. Diese Wertangabe wird bei allen Schreib- und Lesevorgängen im System berücksichtigt. Bei fester Satzlänge muss die Blockgröße ein Vielfaches der Satzlänge sein. Zur Optimierung kann beigetragen werden, wenn die Blockgröße, oder ein Vielfaches davon möglichst nahe unter der Spurkapazität der Platte liegt, auf der die Datei wohnt. BLKSIZE= kann unter SMS auch außerhalb von DCB kodiert werden. Dabei ist die Angabe BLKSIZE=0 die Auftrag an das SMS die beste Blockung selbst zu berechnen.

DSORG=dateiorganisation

Mögliche Angaben sind PO oder PS. DSORG= kann unter SMS auch außerhalb der DCB-Angabe kodiert werden.

BUFL=bufferlength

Hier wird die maximale Blocklänge angegeben (wird nur in Verbindung mit Satzformat "U" benötigt (Lademodul-Bibliotheken)).

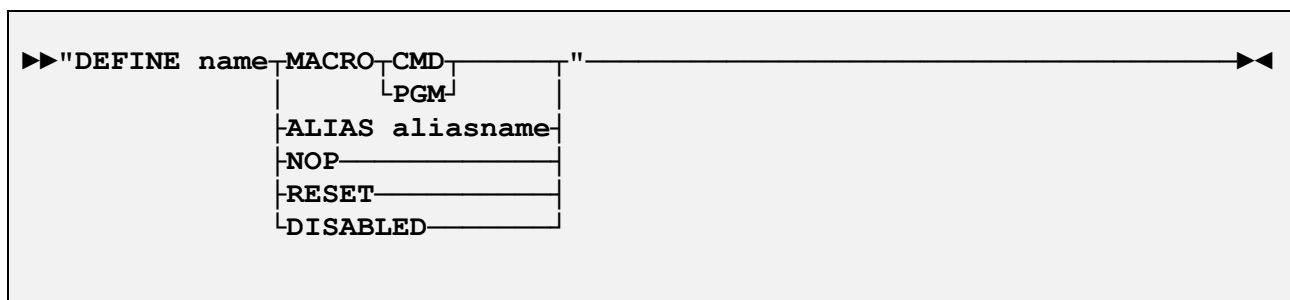
DDNAME – JCL DD Parameter

►►DDNAME=ddname

Wird benutzt, um Dateien durch vorwärtsreferenzierte Angaben zu beschreiben. Diese Variante ist nur möglich, wenn es sich um Angaben innerhalb des gleichen Steps handelt.

Achtung:

Liegen zwischen der Referenz und der Dateiverkettung andere DD-Statements, wird bei Bezugnahme über DDNAME= nur die erste Datei der Verkettung berücksichtigt. Alle anderen Dateien werden mit dem letzten vor der Verkettung definierten DD-Namen verknüpft

DEFINE – ISPF Edit Macro

Der **DEFINE**-Befehl kann

- Macro oder **BUILT-IN**-Befehl mit einem Aliasnamen versehen
- ein Macro oder ein **BUILT-IN**-Kommando sperren
- einen **BUILT-IN**-Befehl durch ein gleichnamiges Macro ersetzen
- Programme als Edit Makros ausweisen

Im Unterschied zum Primärbefehl, ist die Wirkungsdauer von **DEFINE** in einem Macro auf die Dauer der Edit Session begrenzt.

name

Name , über den ein Befehl aufgerufen werden soll.

MACRO CMD

Weist name als CLIST- oder REXX-Macro aus.

MACRO PGM

Weist name als Programm-Macro (Lademodul) aus.

ALIAS aliasname

Bestimmt name als Alias von aliasname. Ist aliasname bereits ein Alias, wird es durch name überschrieben. Es ist nicht möglich, Alias von Alias zu bilden.

NOP

Setzt NO OPERATION. Aliase werden ebenfalls betroffen.

RESET

Führt Grundstellung herbei. Wurde für name NOP gesetzt, ist der Befehl nach RESET wieder ausführbar. Dies betrifft auch alle Aliase.

DISABLED

Verhindert die Ausführung des Befehles und dessen Aliase. Diese Aussage bleibt gültig, bis Edit beendet wird und kann durch einen erneuten DEFINE-Befehl mit Parameter RESET nicht rückgängig gemacht werden.

Beispiele

```
"define rxm#such macro"
```

Definiert die CLIST oder das REXX-Programm SUCH als Macro.

```
"define such alias rxm#such"
```

Legt für das Macro RXM#SUCH einen Zweitnamen SUCH fest, über den es aufgerufen werden kann.

Return Codes

0	Normales Ende
20	Schwerer Fehler

DELETE – DB2 Instruction

```
▶▶"DELETE FROM [schema.]table WHERE ausdrück"▶▶
```

Löscht eine oder mehrere Zeilen unter Bezug auf die WHERE-Klausel aus einer Tabelle.

DELETE – ISPF Edit Macro

```
▶▶"DELETE-zz"▶▶  
[bereich]  
[ALL] [X] [bereich]  
[NX]
```

Löscht eine oder mehrere Zeilen unter Bezug auf ein Label, einen Labelbereich, oder aus- oder eingeblendete Zeilen.

zz

Kann ein Label oder eine relative Zeilennummer sein.

bereich

Benennt einen Von/Bis-Bereich, der zu löschen ist. Die Angaben können entweder über Label oder über relative Zeilennummern erfolgen.

ALL

Stellt unter Bezugnahme auf ein- oder ausgeblendete Zeilen eine Beziehung zu allen betreffenden Zeilen her.

X oder NX

Nimmt auf ausgeblendete (eXcluded) oder sichtbare (Not eXcluded) Zeilen Bezug.

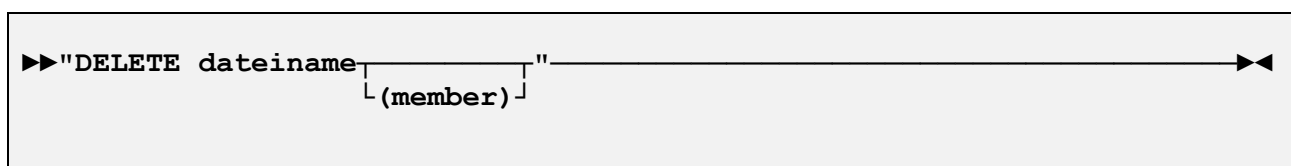
Beispiele

```
"reset x .anf .end"  
"exclude all '/' 1 .anf .end"  
"delete all x .anf .end"
```

Zwischen :ANF und :END werden zunächst alle ausgeblendeten Zeilen wieder eingeblendet. Anschließend werden innerhalb dieses Bereiches Zeilen, die mit /* auf Spalte 1 beginnen, ausgeblendet und gelöscht.

Return Codes

0	Normales Ende
4	Keine Zeilen gelöscht
8	Keine Datensätze vorhanden
12	Falsche Zeilennummer
20	Schwerer Fehler

DELETE – TSO Command

An dieser Stelle gehen wir nur auf die schlichte Art des Delete Befehls ein. Generell können vom Delete auch GDGs und VSAM-Cluster, sowie Kataloge betroffen sein. Dies sind aber eher exotische Dinge für welche ich bitte, das IBM-Manual zu Rate zu ziehen.

Löscht eine Datei oder ein Member einer PO-Datei. Gleichzeitig wird der Dateieintrag aus dem Katalog entfernt.

Bei Angabe des Dateinamens sind die TSO-Dateinamenskonventionen zu beachten.

Bei Angabe des Membernamens sind generische Angaben möglich.

Hinweis

Die aktuellen Einstellungen des TSO-Benutzerprofiles (PREFIX/NOPREFIX) werden berücksichtigt.

Zum Löschen eines Members wird das gesamte PO-Dataset mit Disposition OLD belegt. Ist der exklusive Zugriff auf das Dataset nicht möglich, kann der LM-Service LMMDEL benutzt werden.

Für das Löschen einer Datei steht ebenfalls ein LM-Service zur Verfügung: LMERASE. Der Service liefert bessere Fehlermeldungen.

Return Codes

0	Normales Ende
4	Datei gelöscht, aber ein Warnhinweis wurde ausgegeben
8	Nicht gelöscht, weil nicht vorhanden
12	Nicht gelöscht, weil belegt
20	Schwerer Fehler

DELSTACK - TSO extended Command

►► "DELSTACK" ◀◀

Löscht den aktuellen STACK-Bereich, der durch NEWSTACK erzeugt wurde, mit all seinen Puffern und Elementen.

Wurde kein zweiter STACK erzeugt, werden alle Elemente des "Original-STACK-Bereiches" gelöscht. Um dies nicht ungewollt zu erreichen, könnte in diesem Fall eine Routine aufgerufen werden, die die Auflösung des Original-STACK verhindert (siehe nachfolgendes Beispiel).

Beispiel

```

:
"NEWSTACK"          /* STACK 2 wird erzeugt          */
:
"NEWSTACK"          /* STACK 3 wird erzeugt          */
:
"NEWSTACK"          /* STACK 4 wird erzeugt          */
:
"QSTACK"            /* QSTACK füllt RC mit der        */
DO rc-1             /* Anzahl der STACK-Bereiche.     */
    "DELSTACK"      /* Bis auf STACK-1 werden        */
END                 /* alle STACK gelöscht           */
:

```

DELSTR() – REXX Built-in-Function

►►DELSTR(string,von└┬,länge┘)◄◄

Löscht den Teil einer Zeichenfolge ab der Stelle von in der in länge angegebenen Anzahl Byte. Wird länge nicht angegeben, wird ab der angegebenen Stelle bis zum Ende des Strings gelöscht.

Beispiele

```
SAY DELSTR('Hans ist doof',5)      'Hans'
SAY DELSTR('Hans ist doof',6,4)    'Hans doof'
```

```
Text = "Ist keine ganze Zahl"
Zahl = "34.5"
If ^datatype(zahl,"W") then Text = delstr(text,5,1)
Say text
```

DELWORD() – REXX Built-in-Function

►►DELWORD(string,ab└┬,menge┘)◄◄

Löscht Worte (Zeichenfolgen, die durch BLANK voneinander getrennt sind) in einem String. ab bestimmt das x-te Wort im String, ab dem gelöscht werden soll, menge die Anzahl der zu löschenden Worte. Wird menge nicht angegeben, wird der String ab dem betreffenden Wort bis zum Ende gelöscht.

Beispiele

```
SAY DELWORD('Hans ist doof',2)      'Hans'
SAY DELWORD('Hans ist doof',2,1)    'Hans doof'
SAY DELWORD('Hans ist doof',4)      'Hans ist doof'
```

DIGITS() – REXX Built-in-Function

►►DIGITS()◄◄

Übergibt den aktuellen Wert aus NUMERIC DIGITS

Soll dieser Wert im Verlauf einer Prozedur geändert werden, ist es sinnvoll, den aktuellen Wert in einer Variablen abzulegen, um nach Beendigung einer Verarbeitung auf den vorher gesetzten Wert zurückkehren zu können.

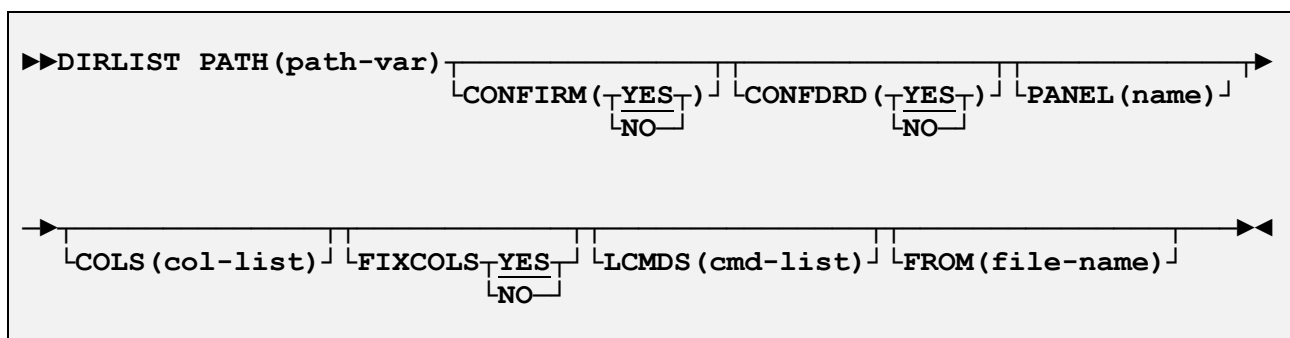
Wird mit sehr großen Zahlen gearbeitet (kommt häufig vor, wenn man mit der Storage-Funktion Speicheradressen verarbeiten will), kann durchaus sein, dass die Rechengenauigkeit (Default 9) erhöht werden muss.

Der Wert hat auch Auswirkung bei Vergleichen numerischer Werte und oftmals wundert man sich, wenn zwei Zahlen offenbar als gleich gesehen werden, welche es aber definitiv nicht sind.

Beispiel

```
/* REXX */
alt_digit=digits()
numeric digits 30
/*
Verarbeitung mit neuem Wert für Digits
*/
numeric digits alt_digit
exit
```

DIRLIST – ISPF DM Service



DIRLIST erlaubt das Schreiben eines eigenen z/OS UNIX Directory List Dialoges vergleichbar zu ISPF/PDF 3.17, bei dem die Dialoganwendung die Steuerung behält.

Path-var

Enthält den Pfadnamen eines Unix Directories

CONFIRM(YES | NO)

Regelt, ob im Falle eines Delete Befehls ein Confirmation-Panel aufgerufen wird (falls der Anwender seine Meinung ändern könnte).

CONFDRD(YES | NO)

Regelt, ob im Falle eines Delete Directory-Befehls ein Confirmation-Panel aufgerufen wird (falls der Anwender seine Meinung ändern könnte).

PANEL(name)

Sofern ein anderes als der DIRLIST-Default-Panel benutzt werden soll.

COLS(col-list)

Spaltennamen, die angezeigt werden sollen. Fehlt die Angabe, wird die vom Benutzer definierte Column-List angewandt (siehe Directory List Column Arrangement Auswahl im UNIX Directory List Utility Options PullDown Menü).

FIXCOLS(YES | NO)

Steuert, ob die Column-Liste änderbar sein soll.

LCMDS(cmd-list)

Liefert die Linecommands, welche der Dialog zulassen soll. Das erste Wort ist der Name des Linecommand-Processors (kann eine REXX oder ein Lademodul sein). Erfolgt beispielsweise die Angabe LCMDS(MYLINECM B) Wird das Zeilenkommando B nicht von DIRLIST in Form von Browse interpretiert, sondern die REXX "MYLINECM" kümmert sich um die Umsetzung.

Return Codes

- 0 Normales Ende
- 8 Fehler beim Aufbau der DIRLIST. Der Grund liegt in einer der ISPF Dialogvariablen
- 12 Falsches Schlüsselwort
- 20 Schwerer Fehler

DISCONNECT – DB2 Instruction

►►"DISCONNECT"◄◄

Beendet die Verbindung zu DB2

DISP – JCL DD Parameter

►►DISP=(

NEW
OLD
SHR
MOD

,

CATLG
KEEP
DELETE
UNCATLG
PASS

,

CATLG
KEEP
DELETE
UNCATLG

)◄◄

DISP kennt drei Unterparameter. Der erste ist der Dateistatus. Der zweite und dritte bestimmen, was mit einer Datei bei normalem Ende, bzw. bei ABEND geschieht. Zu beachten ist, dass ein Programmende mit RC <> 0 ein normales Ende ist, auch wenn die Verarbeitung unter Umständen nicht das gewünschte Ergebnis gebracht hat.

DISP bezieht sich immer auf die Datei, nie auf ein Member. Soll in eine existente PO-Datei geschrieben werden, ist als DISP immer OLD anzugeben. Ist das Member vorhanden, wird es überschrieben, ansonsten angelegt.

Wird der DISP-Parameter nicht angegeben, wird **NEW,DELETE,DELETE** angenommen. Wird nur **DISP=NEW** angegeben, so gilt für den zweiten und dritten Unterparameter ebenfalls **DELETE**.

Wenn eine Datei in einem nachfolgenden Step erneut verwendet wird, sollte sie möglichst mit PASS übergeben werden. Damit wird eine neue Zuordnung (ALLOCATION) vermieden.

NEW

Die Datei soll eingerichtet werden. Bei Plattendateien muss zusätzlich der SPACE-Parameter angegeben werden.

OLD

Die Datei ist bereits vorhanden. Wird sie beschrieben, so werden die alten Daten überschrieben. Bei der Dateizuordnung wird diese Datei exklusiv belegt.

MOD

Datensätze werden beim Schreiben an das Ende der Datei angefügt (nur bei sequentiellen Dateien erlaubt). Ist die Datei noch nicht vorhanden, wird sie eingerichtet.

SHR

Die Datei besteht und kann auch von anderen Jobs eröffnet werden. SHR sollte für das Lesen einer Datei benutzt werden.

DELETE

Der Speicherplatz dieser Datei wird freigegeben. Wenn die Datei katalogisiert war, wird auch der Katalogeintrag gelöscht, vorausgesetzt, die VOLUME-Information wurde aus dem Katalog entnommen.

KEEP

Die Datei bleibt weiterhin gesichert.

PASS

Die Datei bleibt bestehen und wird an einen nachfolgenden Step innerhalb des gleichen Jobs übergeben.

CATLG

Die Datei bleibt bestehen und wird katalogisiert.

UNCATLG

Die Datei bleibt bestehen, der Katalogeintrag wird gelöscht.

Beispiele

DISP=(NEW,CATLG,DELETE)

Neuanlage. Bei regulärem Programmende wird die Datei katalogisiert, bei ABEND wird sie gelöscht.

DISP=(OLD,DELETE,KEEP)

Datei ist bereits vorhanden. Bei ordnungsgemäßigem Programmende wird die Datei gelöscht, bei ABEND bleibt sie erhalten (Restart).

Hinweise:

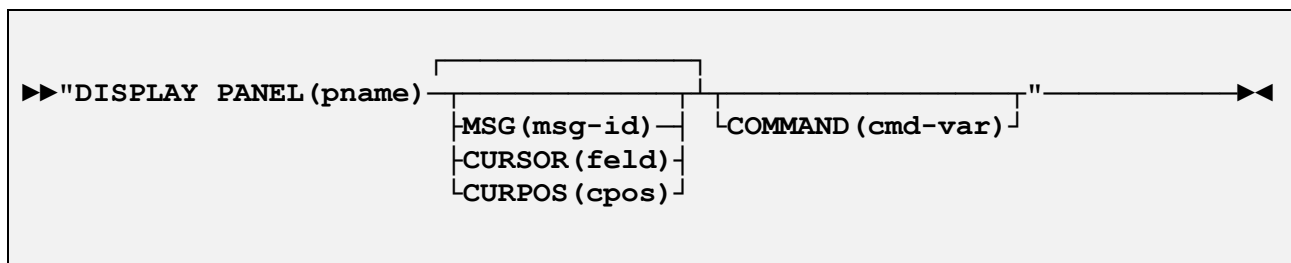
Wird nur der Status angegeben, können die Klammern entfallen.

Fehlt der zweite oder dritte Parameter, wird vom System so ergänzt, dass nach Verarbeitung die gleiche Situation vorhanden ist, wie vor Verarbeitung (bei NEW wird DELETE, bei OLD und SHR wird KEEP angenommen).

Wird bei Neuanlage einer Datei kein ordnungsgemäßer Katalogeintrag durchgeführt, z.B.: DISP=(NEW,KEEP), kann über den Systemkatalog nicht auf diese Datei zugegriffen werden.

Seit z/OS 3.1 wird der Parameter ABDISPCC in der EXEC-Anweisung angeboten, welcher ermöglicht, bei bestimmten aufgetretenen Returncodes die DIOSP-Angabe für ABEND auszuführen.

DISPLAY – ISPF DM Service



Der DISPLAY-Service ruft eine Bildschirmmaske auf. Vor Anzeige werden unter Umständen Variablen und/oder Felder mit Werten gefüllt. Der Benutzer kann die entsprechenden Felder ausfüllen.

PANEL(pname)

ist der Name des anzuzeigenden Panels. Die Bildschirmmaske muss unter diesem Namen Member einer Datei unter ISPLIB sein.

MSG(msg-id)

benennt die Nachricht, die zusammen mit dem Panel gezeigt werden soll. Die Nachricht muss in einem entsprechenden Member unter ISPLIB gefunden werden.

CURSOR(feld)

übergibt das Feld innerhalb der Bildschirmmaske, auf das nach Bildschirmaufbau der Cursor positioniert werden soll.

Wird die Steuervariable `.CURSOR` im Panel kodiert, hat sie Vorrang vor der Angabe im `DISPLAY`-Statement.

CURPOS(cpos)

Byte-Position im Feld, an die der Cursor gesetzt werden soll. Wird die Steuervariable `.CURPOS` im Panel kodiert, hat sie Vorrang vor der Angabe im `DISPLAY`-Statement.

COMMAND(cmd-var)

Erlaubt die Ausführung von Befehlen. Die Variable wird im Function Pool erwartet.

Beispiel

```
:  
MyCMD = "PFSHOW OFF"  
"CONTROL NONDISPLAY ENTER"  
"DISPLAY PANEL(panel) COMMAND(MYCMD) "  
:
```

Führt den Befehl PFSHOW OFF aus.

Return Codes

0	Normales Ende
4	Ein oder mehrere Befehle im Stack konnten nicht in der aktuellen Command Table gefunden werden
8	Das Panel wurde mit END oder RETURN verlassen
12	Panel, Cursorfeld oder Message nicht gefunden oder Syntaxfehler im Panel
16	TRUNC- oder TRANS-Fehler bei der Wertzuweisung im Panel
20	Schwerer Fehler

DISPLAY_COLS – ISPF Edit Macro

```
►►" (var1,var2) = DISPLAY_COLS"◄◄
```

Liefert die linke und rechte Spaltennummer des sichtbaren Bereiches. Spalten, in denen Zeilensequenznummern stehen, werden nicht berücksichtigt. Die Informationen sind erst verfügbar, wenn die Daten am Terminal präsentiert werden. Aus diesem Grund kann die Funktion nicht in einem Initialisierungsmacro benutzt werden.

var1

Äußerste linke sichtbare Spaltennummer (dreistellig numerisch mit führenden Nullen).

var2

Äußerste rechte sichtbare Spaltennummer (dreistellig numerisch mit führenden Nullen).

Beispiele

```
"(li,re) = DISPLAY_COLS"
```

Linke und rechte Spalte des sichtbaren Bereichs werden ermittelt.

Return Codes

0	Normales Ende
12	Falsches Befehlsformat
20	Schwerer Fehler

DISPLAY_LINES – ISPF Edit Macro

```
►►"(var1,var2) = DISPLAY_LINES"◄◄
```

Liefert die relativen Zeilennummern der obersten und untersten sichtbaren Zeile.

var1

Variable, die die relative Zeilennummer der obersten sichtbaren Zeile oder des ausgeblendeten Bereiches enthält (sechstellig numerisch mit führenden Nullen).

var2

Variable, die die relative Zeilennummer der untersten sichtbaren Zeile oder des ausgeblendeten Bereiches enthält (sechstellig numerisch mit führenden Nullen).

Beispiele

```
"(oben,unten) = DISPLAY_LINES"
```

Die Variablen OBEN und UNTEN enthalten die relativen Zeilennummern der obersten und untersten sichtbaren Zeile.

Return Codes

0	Normales Ende
4	Keine sichtbaren Zeilen
8	Keine Datenzeilen vorhanden
20	Schwerer Fehler

DO FOREVER – REXX Instruction

```
DO FOREVER
    instruktion(en)
END
```

FOREVER

Bildet eine Endlosschleife, die durch die Instruktion LEAVE verlassen werden kann. Die Verarbeitung wird hinter dem zu DO FOREVER gehörenden END fortgesetzt.

Beispiel

```
$=0
say 'Eingabeschleife (Verlassen durch Eingabe ENDE) '
do forever
    parse pull ein
    if translate(ein) = 'ENDE' then leave
    $=$+1
    eingabe.$=ein
end
say 'Es wurden' $ 'Eingaben empfangen'
```

DO wert – REXX Instruction

```
DO wert
    instruktion(en)
END
```

wert

Liefert zu Beginn der Schleifenverarbeitung einen Zähler, der von REXX bei jedem Rücksprung an den Schleifenkopf um 1 vermindert wird. Hat der Zähler den Wert 0 erreicht, wird die Schleifenverarbeitung verlassen.

wert kann auch das Ergebnis einer Funktion oder eines arithmetischen Ausdruckes sein.

Beispiel

```
"EXECIO * DISKR ddname (FINIS"
do queued()
    parse pull SATZ
    say SATZ
end
```

DO var = wert TO ende (Iteration) – REXX Instruction

```
DO var = wert TO ende BY ± schrittweite ▶◀  
    instruktion(en)  
END
```

var=wert

Name der Schleifensteuervariablen und ihr Initialisierungswert.

TO ende

Wert, dem sich die Steuervariable schrittweise annähert.

BY ± schrittweite

Die Schrittweite der Zählerveränderung sowie die Richtung (PLUS oder MINUS) kann durch eine Angabe BY gesteuert werden.

Beispiel

```
:  
say 'Achtung, Countdown läuft...'  
DO X = 100 to 1 BY -1  
    say x  
    do y=1 to 1000  
        /* nicht nachahmenswert wg. Performance */  
        /* in den UNIX-System Services gibt es eine */  
        /* SLEEP-Funktion, die keine CPU verbraucht */  
        nop  
    end  
END  
:
```

DO WHILE|UNTIL bedingung – REXX Instruction

```
DO WHILE|UNTIL bedingung  
    instruktion(en)  
END
```

Der primäre Unterschied zwischen der UNTIL- oder WHILE-Schleife liegt in der Prüfung der jeweiligen Bedingung. Zunächst wird in beiden Fällen die Schleife betreten.

WHILE

In Verbindung mit WHILE wird die Bedingung am Schleifenanfang, das heißt, vor Ausführung der in der Schleife liegenden Befehle, geprüft. Man spricht von einer PRE-abweisenden Schleife

UNTIL

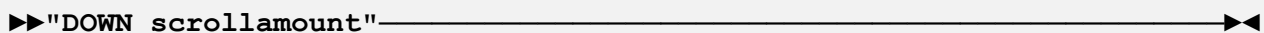
Bei UNTIL wird die Bedingung nach Ausführung der Statements, das heißt am Schleifenende, geprüft. Man spricht von einer POST-abweisenden Schleife.

Beispiel

Grundsätzlich lassen sich mehrere unterschiedliche Bedingungen in einer Verarbeitungsschleife vereinen wie

```
DO $ = 1 to 500 WHILE VAR1 <> "" UNTIL ENDE=JA & DoIt
    verarbeitung
END
```

Eine Verarbeitung in dieser Form ist aber nur REXX-Insidern zu empfehlen (und auch nur, wenn es sein muss – es muss in aller Regel nie sein).

DOWN – ISPF Edit Macro

DOWN wird benutzt, um den Bildschirmausschnitt nach unten zu verschieben.

scrollamount

MAX	Zum Datenende
HALF	um eine halbe Seite
PAGE	um eine ganze Seite
CURSOR	Die Cursor-Zeile wird die oberste.
DATA	Die unterste Zeile wird zur obersten.
Zeilen	Anzahl Zeilen nach unten (0-9999)

Beispiele

"down max"

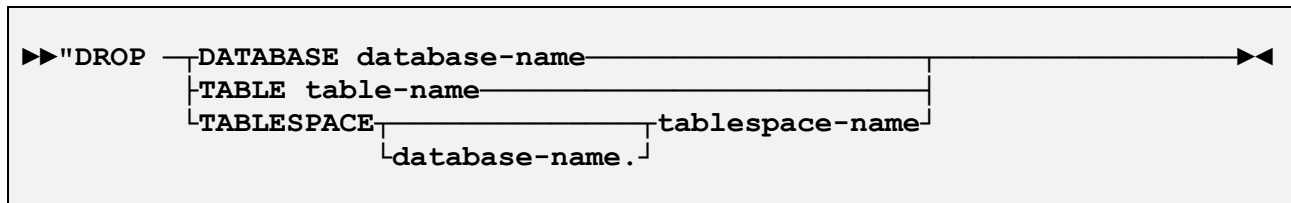
Zeigt das Ende der Datei

Return Codes

0	Normales Ende
2	Down nicht mehr möglich

- 4 Keine sichtbaren Datenzeilen
- 8 Keine Daten anzuzeigen
- 12 Kein Scrollamont angegeben
- 20 Schwerer Fehler

DROP – DB2 Instruction



Drop wird benutzt, um (unter anderem) eine Datenbank, einen Tablespace oder eine Tabelle im DB2 zu löschen.

Die Parameter sind selbsterklärend.

DROP – REXX Instruction



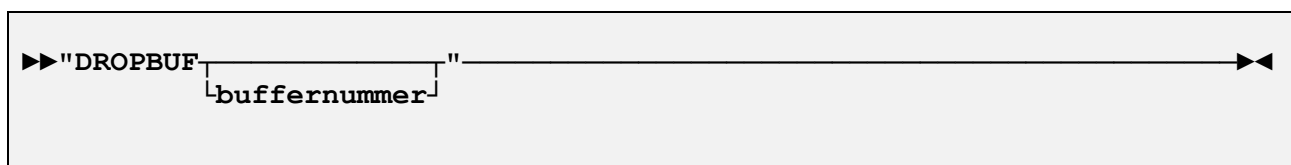
Durch DROP kann einer oder mehreren Variablen, oder auch einem Variablenstamm die Wertzuordnung genommen werden, so dass sie wieder auf ihren Namen referieren.

Beispiele

drop a b table.

Setzt die Inhalte der Variablen A, B und des gesamten Stems TABLE. Auf Grundwert (alles referenziert wieder auf den Eigennamen in Großbuchstaben).

DROPBUF - TSO extended Command



Entfernt einen Puffer aus dem STACK. Wird keine Zahl angegeben, wird der zuletzt erzeugte Puffer entfernt.

buffernummer

Ist eine Zahl angegeben, wird der entsprechende Puffer und alle darunter liegenden entfernt.

Durch die Angabe "DROPBUF 0" werden alle durch MAKEBUF erzeugten Puffer gelöscht.

Return Codes

- 0 Normales Ende
- 1 Falscher Wert angegeben (z.B.: A1)
- 2 Der Puffer existiert nicht. Kommt vor, wenn z.B.: QBUF eine 3 gemeldet hat, aber DROPBUF 4 angegeben wurde.

DSINFO – ISPF DM Service

```

>>"DSINFO DATASET(datei) [VOLUME(volser)]"

```

DSINFO liefert eine Reihe von Informationen, wie sie auch unter ISPF 3.2 vorliegen, in entsprechenden Variablen in Shared Pool. Für DSINFO ist kein LMUNIT notwendig.

DATASET(datei)

Name der Datei nach TSO-Konventionen, über welche die Informationen angezeigt werden sollen

VOLUME(volser)

Name des Datenträgers, auf dem die Datei liegt. Diese Angabe ist nur notwendig, wenn die Datei nicht katalogisiert ist.

Variablen in Shared Pool

VAR-Name	Bedeutung	Länge
ZDSVOL	Erstes oder einziges Volume	6
ZDS#VOLS	Anzahl der Volumes	2
ZDSDEVT	Device Type	8
ZDSORG	Dateiorganisationsformat	8
ZDSRF	Satzformat	6
ZDSLREC	Satzlänge	7
ZDSBLK	Blocksize	6
ZDSSPC	Primary Space Units	8
ZDS1EX	Primary Space Allocation	13
ZDS2SPC	Secondary Space Units	8
ZDS2EX	Secondary Space Allocation	13
ZDSTOTAX	Allocated Space Units (Langformat)	18
ZDSTOTUX	Used Space Units (Langformat)	13
ZSDSNT	Data Set Name Type	8
ZDSSEQ	Compressible (YES/NO)	4

ZDSCDATE	Creation Date (Nationales Format)	10
ZDSXDATE	Expiration Date (Nationales Format)	10
ZDSRDATE	Last Referenced Date (Nationales Format)	10
ZDSTOTA	Allocated Space Units	13
ZDSTOTU	Used Space Units	13
ZDSEXTA	Allocated Extents	7
ZDSEXTU	Used Extents	7
ZDSDIRA	Allocated Directory Blocks	6
ZDSDIRU	User Directory Blocks	8
ZDSDIR	PDSE Directory Blocks	8
ZDS#MEM	Number Of Members	13
ZDSPAGU	Pages Used (PDSE)	13
ZDSPERU	Percent Used (PDSE)	13
ZDSMC	Management Class	8
ZDSSC	Storage Class	8
ZDSDC	Data Class	8
ZDSAPF	APF-Status der Datei (YES NO ERR)	4
ZDSLNK	LNKLST-Status der Datei (YES NO ERR)	4
ZDSCB1	F1-Ettikett Control Block	96
ZDSVTAB	Volume Table (alle VOLSER einer Multi-Volume-Datei)	354
ZDSOVF	Zeigt an, dass ZDSTOTAX und ZDSTOTUX anstelle von ZDSTOTA und ZDSTOTU benutzt werden sollen	3
ZDSDSNV	In Verbindung mit PO/E kann Typ1 oder 2 ermittelt werden	1
ZDSNGEN	Anzahl maximaler Generationen bei PO/E-2	10
ZDSEATR	OPT: Datei kann erweiterte Attribute haben NO: Datei kann keine erweiterten Attribute haben Blank: Der Parameter EATTR wurde nicht angegeben	4
ZDSCJOB	Jobname welcher die Datei erstellt hat (nur EAV-Volumes)	8
ZDSCSTPN	Stepname welcher die Datei erstellt hat (nur EAV-Volumes)	10

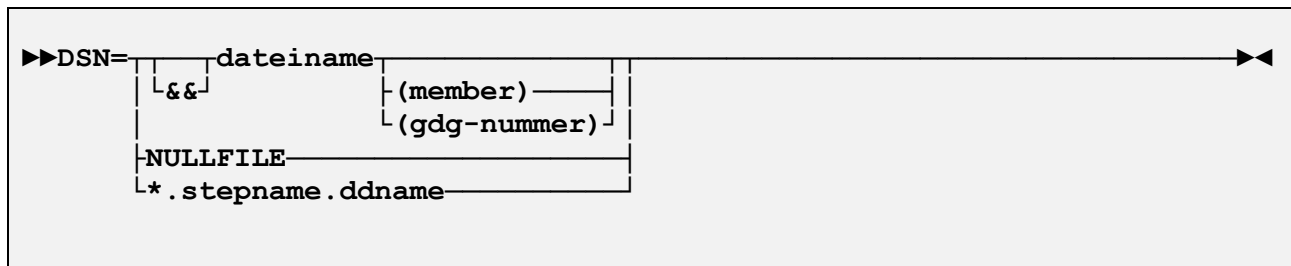
Hinweis

ISPF kann keine Angaben zu BDAM Dateien machen. Aus diesem Grund stehen in allen Variablen für Space-Auslastung in Verbindung mit einer BDAM-Datei Fragezeichen(?).

Return Codes

- 0 Normales Ende
- 8 Datei nicht gefunden
- 8 FAMS Error oder Obtain Error oder Fehler bei Directory-Informationen
- 20 Schwerer Fehler

DSN – JCL DD Parameter



DSN=dateiname | DSN=dateiname(member)

Verbindet den symbolischen Namen im Programm (ddname) mit einem physischen Datenbestand.

Sollen unter einem symbolischen Namen mehrere Dateien (oder Member von PO-Dateien) gelesen werden, können diese miteinander verknüpft werden. Dazu werden mehrere DD-Anweisungen nacheinander kodiert bei denen jede einzelne eine der zu verkettenden Dateien beschreibt, aber nur die erste DD-Anweisung einen DD-Namen enthält. Bei unterschiedlicher Blockgröße muss die Datei mit der größten Blockung als erste genannt werden.

DSN=&&dateiname | DSN=&&dateiname(member)

Bei Verwendung von && vor dem Namen einer Datei wird diese temporär angelegt. Diese Schreibweise wird benutzt, wenn

- ein Membername angegeben werden soll
- die Datei an einen nachfolgenden Step übergeben wird und der Dateiname spezifiziert werden soll.

Ist dies nicht der Fall, kann die Angabe DSN= ganz entfallen.

Für temporäre Dateien wird vom System ein Name generiert, der wie folgt aufgebaut ist:

```

SYSyyddd.Ttttttt.SVxxx.jobname.dateiname
SYSyyddd.Ttttttt.SVxxx.jobname.Rnnnnnnn

```

Dabei bedeuten:

```

yyddd      Jahr und laufender Tag
tttttt     Uhrzeit (hhmmss)
xxx        interne Nummer
dateiname  Information aus DSN=&&dateiname
Rnnnnnnn   Interne Nummer, wenn &&dateiname fehlt

```

Temporäre Dateien werden bei Jobende generell gelöscht.

DSN=gdg-name(relative generationsnummer)

Die sogenannte Datei-Generationen-Gruppe wird vorwiegend in den Fällen eingesetzt, bei denen die Sicherung eines Datenbestandes nicht nach Ablauf einer bestimmten Frist aufgegeben werden soll, sondern Datensicherung abhängig von der Wiederholung einer Verarbeitung betrieben wird. Die Sicherung von Banddateien wird hierbei auch bei Einsatz eines Bandverwaltungssystems wie CA1 über den Systemkatalog durchgeführt.

Um ein GDG zu erzeugen wird das Dienstprogramm IDCAMS benutzt, mit Hilfe dessen im Systemkatalog hinterlegt wird, wie viele Generationen einer Datei festzuhalten sind.

```
//WITTEMA1 JOB (11199002), 'RAUM A11'
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE GDG (NAME(ARTIKEL.BESTAND) LIMIT(5))
```

Der Generations-Dateiname wird intern wie folgt erweitert, damit ein eindeutiger Dateiname entsteht:

```
ARTIKEL.BESTAND.Gmmmm.Vnn
mnnnn = Generationsnummer (0000 bis 9999)
```

Die augenblickliche Generationsnummer wird um den Wert der relativen Generationsnummer erhöht oder vermindert.

```
nn = Versionsnummer (vom System wird 00 eingesetzt)
```

NULLFILE

Bestimmt, dass es sich um ein Dummy-Dataset handelt. Diese Angabe bewirkt das Gleiche wie die Angabe DUMMY in der DD-Anweisung

DSN=*.stepname.ddname (Rückbezugnahme)

Es wird auf eine DSN-Angabe innerhalb einer DD-Anweisung eines vorangegangenen Steps Bezug genommen.

```
//XWITTA1 JOB (1199002)
//*
//IEBGENER EXEC PGM=IEBGENER
//SYSPRINT DD ...
//SYSIN DD ...
//SYSUT1 DD ...
//SYSUT2 DD DSN=SP1.TEST.DATA,DISP=OLD
//*
//SORT EXEC PGM=SORT
//SORTIN DD DSN=*.IEBGENER.SYSUT2,DISP=SHR
//SORTOUT DD ...
//SORTWK01 DD ...
//SYSIN DD ...
//SYSOUT DD ...
```

Hinweise

Wenn der Dateiname für eine Banddatei länger als 17 Stellen ist, werden im Bandkennsatz (HDR1) nur die letzten 17 Stellen (einschließlich der Trennpunkte) eingetragen.

Im Gegensatz zu TSO darf der Dateiname in der JCL im Allgemeinen nicht zwischen Hochkomata gestellt werden.

Die Verwendung von Hochkomata in der JCL erlaubt das Verwenden von Sonderzeichen im Dateinamen. Dies hat zur Folge, dass bei Angabe einer PO-Datei mit Membernamen die runden Klammern, die den Membernamen einschließen, als Dateinamensteil interpretiert werden und die Datei nicht gefunden werden kann.

DSNTYPE – JCL DD Parameter

DSNTYPE=	LIBRARY	
	(LIBRARY,1)	
	(LIBRARY,2)	
	HFS	
	PDS	
	PIPE	
	EXTREQ	
	EXTPREF	
	LARGE	
	BASIC	

Diese Angabe wird nur in Verbindung mit SMS berücksichtigt.

LIBRARY | (LIBRARY,1)

LIBRARY ist ein Dateiformat PDS-E (Partitioned Data Set Extended), welches eine dynamische Verwaltung des Directorys erlaubt und dies in Punkto Mitgliedsleichen selbstreorganisierend ist. Darüber hinaus ist die Anzahl der Extents einer Library nicht auf 16 begrenzt.

(LIBRARY,2)

Eine Type-2-Library ist performanter als ein Type-1 und bietet zudem die Möglichkeit der Versionsführung von Membern. Da dieser Dateityp von den gängigen Editoren (einschließlich PDF) nicht unterstützt wird, empfiehlt sich die Installation eines Tools, welches alles in Verbindung mit Type-2 beherrscht (PDSEGEN von Lionel Dyck – Kostenlos auf www.cbttape.org).

HFS

Bestimmt ein HFS-Dataset und ist nur in Verbindung mit einer DSN-Angabe möglich

PDS

Standard-PO

PIPE

Spezielle FIFO-Datei. Nur in Verbindung mit einer PATH-Angabe

EXTREQ | (EXTREC1) | (EXTREC2)

Nur für VSAM

EXTPREF | (EXTPREF11) | (EXTPREF2)

Nur für sequentielle VSAM-Dateien

DUMMY – JCL DD Parameter

►►DUMMY_____◄◄

Ist eine Blindzuweisung. Wird DUMMY auf eine Eingabedatei zugewiesen, wird es wie eine leere Datei behandelt. Erfolgt die Zuweisung auf eine Ausgabe, wird das Schreiben der Datensätze ignoriert.

DYNAMNBR – JCL EXEC Parameter

►►DYNAMNBR=wert_____◄◄

Legt eine maximale Anzahl von Datasets fest, welche innerhalb eines Programmes dynamisch zugeordnet werden können. z/OS hält den dafür benötigten Platz vor.

Abhängig von der Größe der definierten Task Input Output Table (TIOT; 64KB = MAXWERT 3273 oder 32KB = MAXWERT 1635) kann der angegebene Wert maximal der MAXWERT minus der im JOB definierten DD-Statements sein.

D2C() – Decimal to Character – REXX Built-in-Function

►►D2C(integer_____wert)_____◄◄

Umwandlung von Dezimal nach Character. Wenn wert nicht angegeben ist muss integer eine positive Zahl sein, ansonsten wird ein Fehler erzeugt.

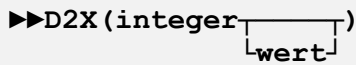
wert

ist die Länge des Ergebnisses in Byte nach der Übersetzung. Hat das Ergebnis in der angegebenen Anzahl Byte nicht Platz, wird entsprechend links abgeschnitten.

Beispiele

```
SAY D2C(7)           '07'x
SAY D2C(129)         '81'x
SAY D2C(129,1)       '81'x
SAY D2C(129,2)       '0081'x
SAY D2C(45,0)        ''
```

D2X() – Decimal to Hex – REXX Built-in-Function



```
►►D2X(integer [wert])
```

Umwandlung von Dezimal nach HEX. Wenn wert nicht angegeben ist muss integer eine positive Zahl sein, ansonsten wird ein Fehler erzeugt.

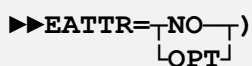
wert

ist die Länge des Ergebnisses in Byte nach der Übersetzung. Hat das Ergebnis in der angegebenen Anzahl Byte nicht Platz, wird entsprechend links abgeschnitten.

Beispiele

```
SAY D2X(7)           '7'  
SAY D2X(129)         '81'  
SAY D2X(129,1)       '1'  
SAY D2X(129,2)       '81'  
SAY D2X(129,4)       '0081'  
SAY D2X(45,0)        ''
```

EATTR – JCL DD Parameter



```
►►EATTR=[NO | OPT]
```

Steuert, ob extended Attributes unterstützt werden sollen, oder nicht (DSCB-8 und DSCB-9 Formate)

Dateien, für die dies vorgesehen ist, können auf EAV-Volumes abgelegt oder in EAS-Adressräumen gehalten werden.

EATTR=NO

Keine erweiterten Attribute erlaubt

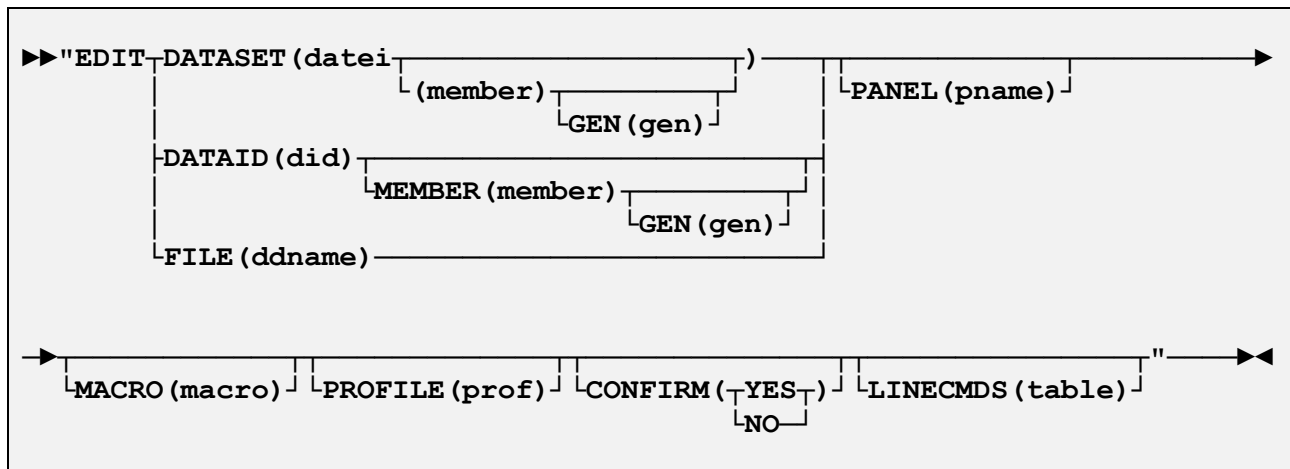
EATTR=OPT

Die erweiterten Attribute werden grundsätzlich unterstützt.

Hinweis

Der Umgang mit EAV-Volumes wird in vielen Unternehmen noch sehr restriktiv behandelt. Haltet gegebenenfalls Rücksprache mit der für Plattenverwaltung zuständigen Stelle im Unternehmen.

EDIT – ISPF DM Service



ruft den ISPF-Editor und präsentiert gleichzeitig die hinter DATASET angegebene Datei. Wird EDIT aus einer Funktion (REXX etc.) aufgerufen, kann nicht wie gewohnt durch =3.4 im PDF-Dialog verzweigt werden. Der Editor kann durch END, RETURN oder CANCEL verlassen werden. Die Ablaufsteuerung wird immer an die Funktion zurückgegeben, die entscheidet, was weiter zu tun ist.

Während der EDIT-Session sind alle Edit-Befehle verfügbar.

Datei | datei(member)

Name der anzuzeigenden Datei. Wird der Name einer sequentiellen Datei angegeben, zeigt BROWSE die Daten. Wird der Name einer PO-Datei ohne Member angegeben, liefert BROWSE das Memberverzeichnis. Wird das Member mit angegeben, werden die Memberdaten gezeigt.

DATAID(did)

DatenID welche per LMINIT zugewiesen wurde

FILE(ddname)

DD-Name, unter dem das Dataset allokiert ist.

GEN(gen)

Generation eines PO/E-Type2-Datasets mit Angabe der relativen Generationsnummer (0, -1, -2, etc.).

PANEL(pname)

Panelname, sofern ein anderes als das Standardpanel der IBM benutzt werden soll.

MACRO(macro)

Name des Initialisierungsmacros.

PROFILE(prof)

Name des zu benutzenden Profils

CONFIRM YES | NO

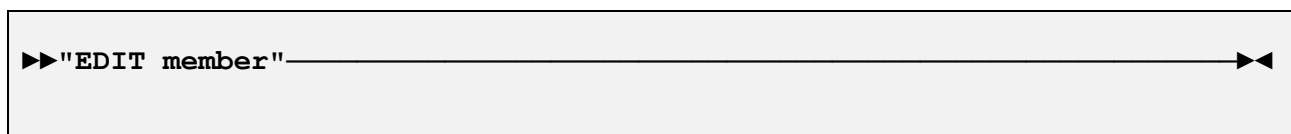
Bestätigung für REPLACE etc.

LINECMDS(table)

Name der Linecommand-Table sofern eine benutzt werden soll.

Return Codes

- 0 Normales Ende. Daten wurden gesichert
- 4 Normales Ende. Daten wurden nicht gesichert. Mögliche Gründe: Daten wurden nicht verändert oder Edit wurde mit CANCEL verlassen oder statt EDIT wurde BROWSE benutzt (wegen Speicherproblemen).
- 10 Generationismember nicht gefunden
- 12 LOCK war auf YES gesetzt
- 14 Datei, Member oder Unix File in USE
- 16 Kein Member matching entsprechend dem Pattern oder keine Member vorhanden
- 18 Es wurde eine VSAM-Datei angegeben und in der ISPF Configuration Table ist dies nicht erlaubt
- 20 Schwerer Fehler

EDIT – ISPF Edit Macro

Ruft den Editor zur Bearbeitung eines anderen Members der gleichen PO-Datei (rekursiver Aufruf). Alternativ könnte Edit auch über Dialog Manager aufgerufen werden. Dies bietet den Vorteil, auch auf andere Datenbestände zugreifen zu können und ein Initialisierungsmacro ausführen zu lassen.

member

Name des zu editierenden Members.

Beispiele

"edit adr2"

Öffnet eine neue EDIT-Session mit den Memberdaten von ADR2.

Return Codes

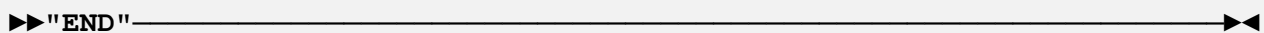
- 0 Normales Ende
- 2 Membername falsch oder Recovery offen
- 12 LOCK für Member auf YES
- 14 Member in USE
- 20 Schwerer Fehler

EDIT – TSO Command

Der Befehl sei hier nur der Vollständigkeit wegen erwähnt. Außerhalb von ISPF zu editieren ist heute niemandem mehr zumutbar.

Wer es wirklich Hardcore braucht, kann die Details im Manual "TSO/E Command Reference" nachlesen.

END – ISPF Edit Macro

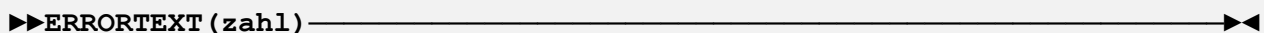


Beendet den Editor. Hinter END kann in einem Macro keine Adressierung mehr an ISREDIT oder ISPEXEC erfolgen. Befehle der jeweiligen Makrosprache (CLIST, REXX etc.) können aber auch nach END noch ausgeführt werden.

Return Codes

0	Normales Ende
4	Neue Daten gesichert
12	END nicht durchgeführt weil AUTOSAVE auf OFF sitzt oder Daten wurden wegen Directory-Überlauf nicht gesichert (warum nicht PO/E benutzen?)
20	Schwerer Fehler

ERRORTXT() – REXX Built-in-Function



Übergibt eine Fehlermeldung mit der in zahl angegebenen Nummer. Die Zahl muss zwischen 0 und 99 liegen, ansonsten wird ein Nullstring übergeben.

Für Details siehe unter "REXX-Reference: Fehlernummern und -nachrichten", oder im Band-1 Kochbuch.

Beispiele

```
SAY ERRORTXT(16)      'Label not found'
SAY ERRORTXT(36)      'Unmatched "(" in Expression'
SAY ERRORTXT(60)      ' '
```


EXCLUDE wirkt nur innerhalb der Boundary-Grenzen, außer, sie werden im Befehl überschrieben.

Wird als Richtung NEXT oder PREV zusammen mit einem Bereich angegeben und der Cursor steht außerhalb des Bereiches, wird der ganze Bereich durchsucht. Ansonsten beginnt die Suche ab Cursorposition.

Wird der String gefunden, ändert sich die Position des Cursors, ansonsten nicht.

Beispiele

```
"exclude all"  
"find do all word"  
"find select all word"  
"find 'end' all word"
```

Hilft bei der Suche nach Fehlern in CLIST oder REXX

Return Codes

0	Normales Ende
4	Zeichenfolge nicht gefunden
8	Keine ausgeblendeten Zeilen
12	Inkonsistente Parameter
20	Schwerer Fehler

EXCLUDE_COUNTS – ISPF Edit Macro

```
►►" (var1,var2) = EXCLUDE_COUNTS"◄◄
```

Übergibt Ergebnisse des EXCLUDE-Befehles an die in der Klammer stehenden Variablen.

var1

Achtstellig numerischer Wert mit führenden Nullen mit der Anzahl gefundener Zeichenfolgen.

var2

Anzahl ausgeschlossener Zeilen, achtstellig numerisch mit führenden Nullen.

Beispiele

```
"exclude all word say"  
" (,says) = exclude_counts"
```

in der Variablen SAYS steht die Anzahl ausgeblendeter Zeilen, die die Zeichenfolge SAY enthalten.

Return Codes

0	Normales Ende
12	Falsches Befehlsformat
20	Schwerer Fehler

EXEC – TSO Command

```

▶▶"EXEC dateiname(member) [parms] [EXEC]"

```

Start von REXX oder Clist. Die Prozedur ist unter dem angegebenen Namen Member der PO-Datei.

Parms

Übergabeparameter

EXEC

Wird der Operand EXEC hinter dem Prozedurnamen angegeben, wird versucht, eine REXX-Prozedur (EXEC) zu starten. Anderenfalls führt der Aufruf zum Starten einer TSO-Clist.

Return Codes

0	Normales Ende
12	Fehler
Others	RC der aufgerufenen REXX

EXECIO - TSO extended Command

```

▶▶"EXECIO [anz] [DISKW] ddname [ ( [STEM name.] [FINIS] [FIFO]
    [DISKR] [LIFO]
    [DISKRU] [SKIP] ) ]"

```

Schreiben und Lesen mit Dateien im TSO. Daten, die gelesen werden, können im STACK abgelegt oder einem Variablenstamm übergeben werden. Daten, die in eine Datei geschrieben werden sollen, können aus dem STACK geholt oder aus einem Variablenstamm ausgelesen werden.

Ein-/Ausgabe des EXECIO kann eine PS-Datei oder ein Member einer PO-Datei sein. Zum Zeitpunkt des EXECIO muss die Datei zugewiesen sein. EXECIO ersetzt nicht die Zuordnung.

Wenn in EXECIO Variable benutzt werden, müssen diese außerhalb der Stringdefinition liegen.

anz

Anzahl der Datensätze, die verarbeitet werden sollen. Wird an Stelle einer Zahl Stern(*) angegeben, werden alle Sätze verarbeitet.

DISKW

Öffnet die Datei (sofern nicht bereits geschehen) mit dem Attribut OUTPUT (zum Schreiben)) und liest die entsprechende Anzahl Datensätze, die geschrieben werden sollen, aus dem STACK-Bereich oder einem Stem.

DISKR

Eröffnet die Datei (sofern dies nicht bereits geschehen ist) mit dem Attribut INPUT (zum Lesen) und liest die entsprechende Anzahl Datensätze, um sie im STACK-Bereich abzulegen, oder an einen Variablenstamm zu übergeben.

DISKRU

Eröffnet mit dem Attribut UPDATE. Dies ist erforderlich, wenn Änderungen in einer Datei durchgeführt werden müssen. Erfolgt bei einem Zugriff mit DISKRU ein "EXECIO DISKW"-Befehl (schreiben), wird der zuletzt gelesene Datensatz (u.U. geändert) zurückgeschrieben.

ddname

Der symbolische Name der Datei, unter dem sie im ALLOCATE-Statement zugewiesen wurde.

FINIS

Schließt die Datei, nachdem EXECIO abgearbeitet ist.

STEM variable

Ein Stem, der die gelesenen oder zu schreibenden Sätze enthält. Um diese Übergabe an indizierte Variable zu vollziehen, sollte hinter den Variablennamen ein Punkt(.) gesetzt werden (Compound-Variablen). Werden 38 Datensätze gelesen stehen diese in den Variablen eing.1 bis eing.38. Die Anzahl gelesener Sätze steht in eing.0.

Endet der Variablenname nicht mit einem Punkt(.), werden zwar ebenfalls aufsteigende Variable gebildet (eing1, eing2...), die aber nicht indiziert angesprochen werden können.

SKIP

Liest die angegebene Anzahl Datensätze, ohne sie im STACK oder in Variablen zu hinterlegen. Wird an Stelle einer Anzahl Datensätze Stern(*) angegeben, wird in Verbindung mit Skip an das Dateiende positioniert.

LIFO

Platziert die gelesenen Datensätze in der Reihenfolge Last In First Out.

FIFO

Übergibt die gelesenen Datensätze in der Folge First In First Out. Wurde weder LIFO, noch FIFO angegeben, wird FIFO angenommen.

Hinweis:

Die Gründe für einen RC 20 bei XECIO lassen sich nicht so ohne weiteres ermitteln. Siehe hierzu auch unter TRAPMSG().

Beispiele

```
"ALLOC DA(EDU.SEQ1) F(EINGABE) SHR REUSE"
"ALLOC DA(EDU.DATA(MEMNEU) F(AUSGABE) SHR"
"NEWSTACK" /* Alter STACK ist geschützt */
"EXECIO * DISKR EINGABE (FINIS"
"EXECIO" queued() "DISKW AUSGABE (FINIS"
"DELSTACK" /* Löschen des erzeugten STACK */
"FREE F(EINGABE AUSGABE)"
:
```

Die sequentielle Datei userid.EDU.SEQ1 werden als Member MEMNEU der Datei userid.EDU.DATA übernommen.

```
ARG zeilen .
"ALLOC DA(TEST.DATA) F(EIN) SHR REUSE"
"EXECIO" zeilen "DISKR EIN (STEM data."
DO $=1 TO DATA.0
  SAY $'. gelesene Zeile:' DATA.$
END
```

Eine am Terminal eingegebene Anzahl Datensätze wird aus einer Datei gelesen in die indizierten Variablen DATA.x gestellt. Die Zahl der gelesenen Datensätze steht in der Variablen DATA.0, die Variablen werden in Schleife ausgegeben.

```
say 'Personalnummer'
pull pers
erg='Keinen passenden Satz gefunden'
do forever
  "execio 1 diskru ddname"
  if rc = 2 then leave
  pull satz
  if substr(satz,1,4)=pers then do
    queue substr(satz,1,4) var2
    erg='Gesuchten Satz geändert'
    "execio 1 diskw ddname (finis"
    leave
  end
end
say erg
```

In einem Datenbestand wird nach einer Personalnummer auf Spalte 1 bis 4 gesucht. Wenn gefunden (Stelle 1 bis 4 ist identisch mit dem Wert der Variablen pers) wird der geänderte Datensatz (Personalnummer unverändert, gefolgt vom Inhalt der Variablen var2) im STACK abgelegt und zurückgeschrieben. Verarbeitungsende ist erreicht, nachdem alle Datensätze gelesen wurden, oder die Änderung durchgeführt worden ist.

Return Codes

- 0 Normales Ende
- 1 String wurde bei DISKW-Operation verkürzt
- 2 Dateiende beim Lesen Erreicht (kann vorkommen, wenn eine Datei nicht als Ganzes eingelesen wird).
- 4 Beim Lesen wurde innerhalb einer Dateiverkettung eine leere Datei gefunden
- 20 Schwerer Fehler, EXECIO wird abgebrochen, eine Fehlermeldung wird ausgegeben (siehe hierzu auch TRAPMSG())

RENAME

mit RENAME kann auf ein Function Package Directory Einfluss genommen werden. Ein Function Package Directory enthält folgende Informationen für jede Function oder Subroutine:

- **function_name**
Name einer externen Funktion oder Subroutine, die in einer EXEC benutzt wird.
- **addr**
Speicheradresse des Entrypoints der Function oder Subroutine.
- **sys_name**
Name des Entrypoints in einem Lademodul, der auf den Code verweist, welcher auszuführen ist.
- **sys_dd**
DD-Name, von dem die Funktion oder Subroutine zu lesen ist.

NAME(function_name)

Name der externen Funktion oder Subroutine. Der Name ist gleichbedeutend mit dem Namen im function_name-Feld im Directory-Eintrag.

SYSNAME(sys_name)

Name des Entrypoints in einem Lademodul, an dem der Code für die Funktion oder Subroutine steht. Fehlt die Angabe, ist das Feld sys_name im Function Package Directory auf Blank gesetzt.

DD(sys_name)

DD-Name von dem das Function Package zu laden ist. Fehlt die Angabe, wird das Feld sys_name im Function Package Directory auf Blank gesetzt.

TS (Trace Start)

Aktiviert den Trace mit der nächsten EXEC, welche aufgerufen wird (und für alle EXECs, die daraus gerufen werden). Der Trace endet

- wenn die originäre EXEC endet
- wenn eine der aufgerufenen EXECs oder CLISTen den TE-Befehl (Trace End) absetzt
- wenn die Eingabe TE im Attention Mode (PA1-Taste gedrückt) erfolgt.

TS wirkt in der Aufrufhierarchie nach unten und oben. Angenommen PROG1 startet PROG2 und PROG2 startet PROG3. Wenn PROG2 einen TS durchführt, erfolgt der Trace ab hier für PROG2, für PROG3 und nach Ende von PROG2 auch für PROG1.

TE (Trace End)

Beendet den Trace. Der TE-Befehl wirkt nur in Folge eines TS-Befehls und NICHT unter TSO native. TE wirkt in einer Aufrufkette in beide Richtungen.

HT (Halt Typing)

Jeglicher Output via SAY wird unterdrückt. Die Verarbeitung läuft aber weiter. HT hat keinen Einfluss auf CLISTen oder TSO-Commands. Halt Typing kann via EXECUTIL RT (Resume Typing) widerrufen werden. Dies kann in einer EXEC oder im Attention Mode erfolgen.

Wird HT in einer CLIST gesetzt, wirkt es nur für unter- oder übergeordnete EXECs.

RT (Resume Typing)

Die Unterdrückung von Terminaloutput wird zurückgenommen. RT ist unter TSO native oder ISPF nicht möglich.

HE (Halt Execution)

Die Ausführung wird unmittelbar beendet-

HI (Halt Interpretation)

Per HI wird die Verarbeitung nicht unmittelbar beendet. Erfolgt HI in einer Subroutine oder Function, wird die gesamte Aufrufhierarchie beendet. HT wird häufig im Attention Mode (PA1-Taste wurde gedrückt) über die Tastatur eingegeben, um im Fehlerfall ein Verarbeitungsende zu erzwingen. Dies wiederum ist nur möglich, wenn die EXEC das Drücken von PA1 nicht über einen Eventhandler (CALL ON HALT) abfängt oder die PA1-Taste nicht als HotKey für andere Anwendungen missbraucht wird (des Öfteren für NetMaster der Fall).

Hinweis

Siehe auch ALTLIB-Kommando

Beispiel

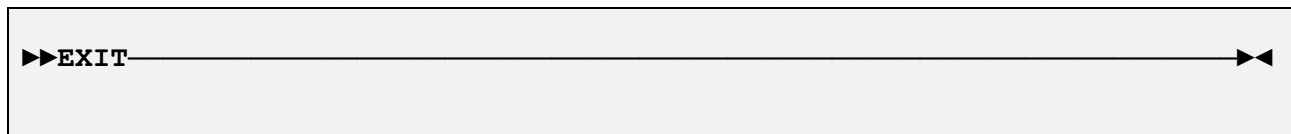
```
/* REXX
*/
:
CALL UPROG1
"EXECUTIL TE"
:
EXIT

UPROG1:
"EXECUTIL TS"
:
:
RETURN
```

Return Codes

- 0 Normales Ende
- 12 Fehler – wird von einer Fehlermeldung begleitet

EXIT – ISPF Panel Instruktion

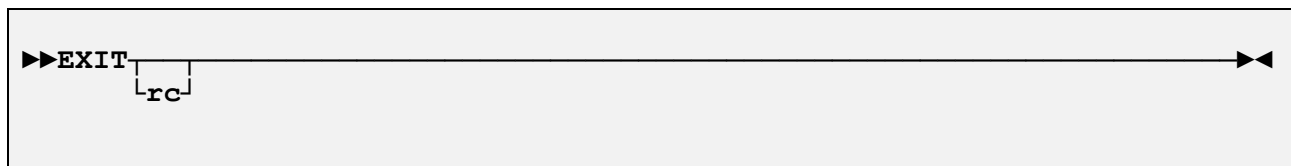


Trifft Dialog Manager während der Verarbeitung einer Panelsection auf EXIT, wird die aktuelle Section sofort verlassen. EXIT in der)INIT-Section beendet diese, führt aber zum Ansprung der)BODY-Section, wogegen EXIT im)PROC-Teil zum Verlassen des Panels führt.

In Verbindung mit GOTO nach einer IF/ELSE-Entscheidung kann durch EXIT die Ablaufsteuerung vereinfacht geschrieben werden.

Wie in allen Sprachen sollte auch hier mit EXIT und GOTO nicht fehlende Denkarbeit in der Steuerung des logischen Ablaufes ausgeglichen werden.

EXIT – REXX Instruction



Wird benutzt, um eine Prozedur unmittelbar zu beenden.

Mit EXIT kann das Ende der Prozedur auch in mitten des Ablaufes gelegt werden. Im Zuge einer ordentlichen Programmierung sollte das aber sparsam benutzt werden.

Hinweis

Ein ordentlich kodierte Programm verfügt maximal über zwei EXITS. Den regulären und einen am Ende einer ABEND-Routine.

Beispiel

```
/* REXX */
do forever
  say 'Druckdatei angeben'
  pull datei
  if sysdsn("'"datei"'") \= 'OK' then exit
  /* instruktionen zur verarbeitung */
end

/* REXX */
:
If sysdsn(datei) <> "OK" then call abend datei "nicht vorhanden"
:
exit
Abend: parse arg zedlmsg;address ispxexec "setmsg msg(isrz000)";exit
```

EXPDT – JCL DD Parameter

```
►►EXPDT=yyddd
           |
           |yyy/ddd|
```

Liefert das Freigabedatum einer Datei. Das Format kann mit 2- oder 4-stelligem Jahr angegeben werden.

Hinweis

In der Regel sinnvoller ist die Verwendung von RETPD (Retention Period). Hier wird eine Anzahl Sicherungstage angegeben und das Freigabedatum wird vom System berechnet.

EXTERNALS() – NonSAA REXX Built-in-Function

```
►►EXTERNALS ( )
```

Externals sei hier nur der Vollständigkeit wegen erwähnt. Der Rückgabewert ist unter TSO immer 0.

Unter VM liefert die Funktion die Anzahl der Elemente im Terminal-Input-Buffer. Unter TSO gibt es hierfür kein Äquivalent.

FETCH – DB2 Instruction

```
►►"FETCH cursor-name INTO :host-variable"
```

Positioniert den Cursor auf eine Zeile in der Ergebnistabelle einer SELECT-Abfrage. Die Instruktion ist sehr vielfältig. Wir werden an dieser Stelle nur den sogenannten "Single Row Fetch" betrachten (und auch diesen nicht vollumfänglich).

Fetch setzt immer auf das Ergebnis eines SELECT-Befehls auf und bedingt, dass der entsprechende Cursor im Vorfeld beschrieben (DECLARE CURSOR), bereitgestellt (PREPARE CURSOR) und geöffnet (OPEN CURSOR) wurde.

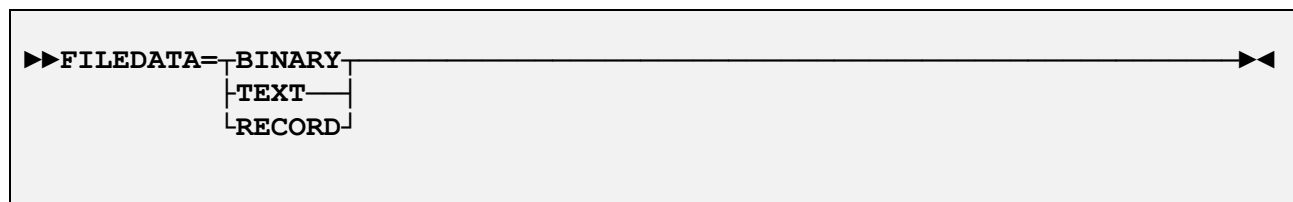
Cursor-name

Name des Cursors aus dem DECLATE-Statement

:host-variable

Name der REXX-Variablen, die den Inhalt des DB2-Feldes aufnehmen soll. Die Reihenfolge der host-variablen muss der Reihenfolge der DB2-Felder im SELECT-Statement entsprechen.

Grundsätzlich könnte das durch dynamic SQL auch ganz anders abgehandelt werden. Dies ist aber in der Mainframe-Umgebung verpönt und ich habe in vielen Jahren keinen Kollegen angetroffen, der dynamic SQL irgendetwas positives abgewinnen kann.

FILEDATA – JCL DD Parameter

Beschreibt den Inhalt eines z/OS UNIX files

BINARY

Bytestrom. Enthält keine Delimiter wie NEWLINE oder vergleichbares.

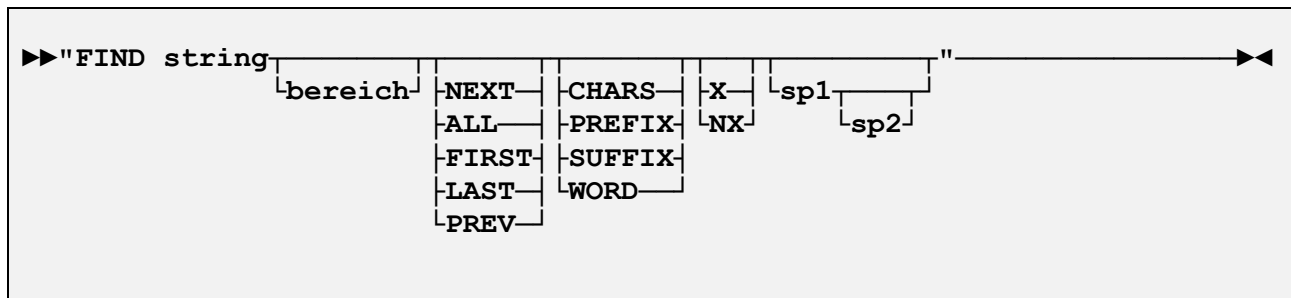
TEXT

Textdaten enthalten Steuerzeichen wie <newline> ('15'x) oder <carriage return> ('0D'x). Nachzulesen unter anderem in "Using REXX and Unix System Services".

RECORD

Ist ein Hinweis darauf, dass dem Datensatz ein Prefix vorangestellt ist. Dies ist beispielsweise bei Sätzen Variabler Länge in Form des Satzlängenfeldes der Fall.

FIND – ISPF Edit Macro



FIND sucht nach einer Zeichenkette. Die Parameter sind die gleichen, wie beim Change-Kommando. Sind Zeilen zum Zeitpunkt des FIND ausgeblendet, die das Suchkriterium enthalten, werden sie wieder eingeblendet.

string

Zeichenkette, nach der gesucht wird.

bereich

Eine von/bis-Angabe in Form zweier Marken (Labels).

Richtung

NEXT	Suche beginnt ab Cursorposition in Richtung Dateiende
ALL	Suche beginnt am Dateianfang in Richtung Ende. Gesucht und geändert werden alle vorkommenden Begriffe
FIRST	Die Suche beginnt am Dateianfang in Richtung Ende. Geändert wird der erste vorkommende Begriff
LAST	Wie FIRST aber vom Ende in Richtung Anfang
PREV	Wie NEXT aber in Richtung Dateianfang

Position

CHARS	Die Zeichen müssen wie angegeben aufeinander folgen, können aber an beliebiger Stelle vorkommen
PREFIX	ZEICHENFOLGE MUSS EIN Wortanfang sein
PREF	
SUFFIX	Zeichenfolge muss ein Wortende sein
SUF	
WORD	Die Zeichenfolge muss ein Wort sein

X

Der Befehl wirkt nur in ausgeblendeten Zeilen (eXcluded lines).

NX

Der Befehl wirkt nur in eingeblendeten Zeilen (Not eXcluded lines)

sp1 sp2

Spaltenangaben, innerhalb derer CHANGE wirken soll (sp1 = links, sp2 = rechts).

Hinweise zum FIND-Befehl

Nicht case-sensitive. Enthält string Sonderzeichen oder Blanks, oder ist es gleichnamig zu Operanden, müssen Apostrophe (') benutzt werden.

FIND wirkt innerhalb der Boundaries.

Wird NEXT/PREV mit einem Bereich angegeben und steht der Cursor außerhalb des Bereiches, wird der ganze Bereich durchsucht. Ansonsten beginnt die Suche ab Cursorposition.

Wird eine Zeichenfolge gefunden, ändert sich die Position des Cursors. Anderenfalls bleibt der Cursor am alten Standort.

Beispiel

```
"find p'$'"
```

Sucht nach dem nächsten vorkommenden Sonderzeichen.

Return Codes

0	Normales Ende
4	Zeichenfolge nicht gefunden
12	Syntax Fehler
20	Schwerer Fehler

FIND() – REXX Built-in-Funktion

►►FIND(string,suchbegriff)◄◄

Durchsucht Zeichenfolgen nach dem angegebenen Suchbegriff (auf Wortbasis im Sinne von REXX) und meldet die Wortposition im String, wenn der Suchbegriff enthalten ist.

Beispiele

```
SAY FIND('Jetzt ist es Zeit','ist es')      2
SAY FIND('Jetzt ist es Zeit','es Zeit')     3
SAY FIND('Jetzt ist es Zeit','Jetzt Zeit')  0
```

FIND_COUNTS – ISPF Edit Macro

```
►►" (var1,var2) = FIND_COUNTS"◄◄
```

Diese Zuweisung liefert die Ergebnisse des letzten FIND- oder RFIND-Befehls.

var1

Name der Variablen, in der die Anzahl der gefundenen Zeichenfolgen hinterlegt werden.

var2

Name der Variablen, in der die Anzahl der Zeilen mit positivem Suchergebnis abgelegt ist.

Beispiel

```
"(fnd,lines) = find_counts"
```

Die Anzahl gefundener Zeichenketten liegt in der Variablen FND, die Anzahl betroffener Zeilen in der Variablen LINES.

Return Codes

0	Normales Ende
12	Falsches Befehlsformat
20	Schwerer Fehler

FLIP – ISPF Edit Macro

```
►►"FLIP"◄◄  
    [bereich]
```

FLIP kehrt den XStatus einer Zeile ins Gegenteil (X nach NX und NX nach X).

Return Codes

0	Normales Ende
20	Schwerer Fehler

FLOW_COUNTS – ISPF Edit Macro

```
"(var1,var2) = FLOW_COUNTS"—————▶◀
```

Diese Zuweisung liefert die Ergebnisse des zuletzt ausgeführten TFLOW-Befehls.

var1

Enthält die Anzahl der Originalzeilen, die vom TFLOW-Befehl betroffen waren. Achtstellig numerisch mit führenden Nullen.

var2

Anzahl Zeilen nach Ausführung von TFLOW. Achtstellig numerisch mit führenden Nullen.

Beispiel

```
"tflow .zcsr" variable  
"(alt,neu) = flow_counts"  
if alt < neu then /*Datei verkürzt */  
else  
    if alt > neu then /* Datei verlängert */  
    else /* Dateigröße unverändert */
```

Return Codes

0	Normales Ende
20	Schwerer Fehler

FORM() – REXX Built-in-Function

```
▶▶FORM()—————▶◀
```

Übergibt den aktuellen Wert aus NUMERIC FORM

FORMAT() – REXX Built-in-Function

```

▶▶FORMAT (zahl [ , vor [ , nach ] ] )

```

Formatiert eine Zahl.

zahl

Ist nur zahl angegeben, wird nach den Regeln von REXX gerundet (NUMERIC DIGITS). Entsprechend der Werte vor/nach werden Vor- bzw. Nachkommastellen definiert.

vor / nach

Ist der Vorkomma-Stellenwert zu klein um zahl aufnehmen zu können, wird Fehler gemeldet. Ist der Vorkommawert größer, wird links mit BLANK aufgefüllt.

Beispiele

SAY FORMAT ('3',4)	' 3'
SAY FORMAT ('3.75',4,0)	' 4'
SAY FORMAT ('3.75',4,3)	' 3.750'
SAY FORMAT ('-.28',4,1)	' -0.3'
SAY FORMAT ('-4.5',,4)	'-4.5000'
SAY FORMAT (' - 6.76')	'-6.76'

FREE – JCL DD Parameter

```

▶▶FREE=END
      CLOSE

```

Regelt den Zeitpunkt der Freigabe einer Datei

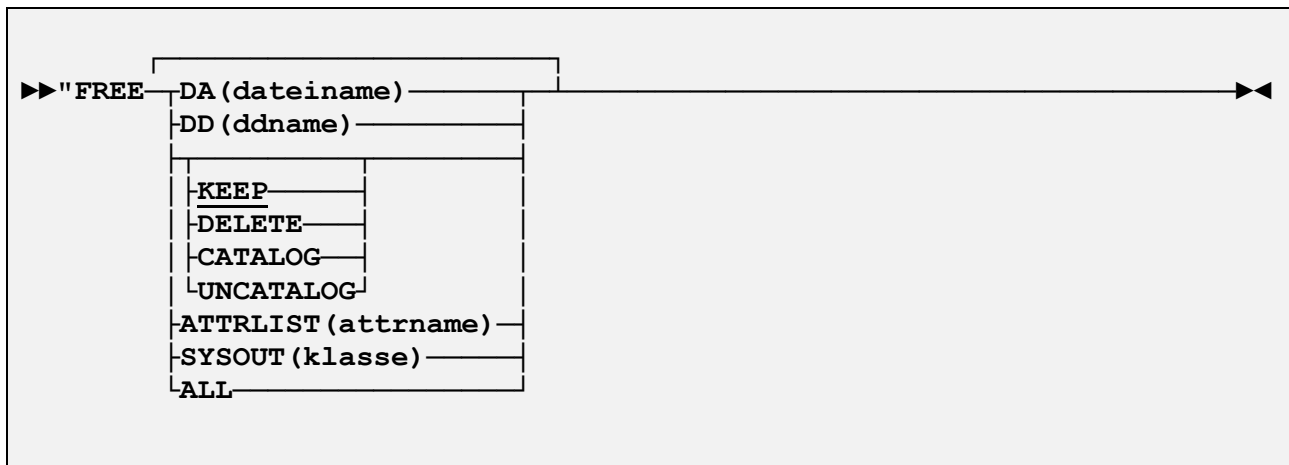
END

Die Freigabe erfolgt am Ende des letzten Steps, der auf diese Datei Referenziert.

CLOSE

Die Freigabe erfolgt beim CLOSE des Programmes auf diese Datei

FREE – TSO Command



Freigabe einer Zuweisung

DA(dateiname)

Die Datei wird freigegeben

DD(ddname)

Der DD-Name wird freigegeben

KEEP | DELETE | CATALOG | UNCATALOG

Überschreibt die Dispositionsangabe aus ALLOC

KEEP	Datei bleibt bestehen. Kein Katalogeintrag
DELETE	Datei wird gelöscht (nur für SYSOUT() erlaubt)
CATALOG	Datei wird im Systemkatalog eingetragen
UNCATALOG	Datei wird im Systemkatalog ausgetragen

SYSOUT(outputclass)

Überschreibt die Outputklasse

Hinweise

FREE ist zeitintensiv. Es sollten möglichst alle Angaben in einem FREE-Kommando kodiert werden. FREE hat aber eine relativ umfangreiche Syntax (siehe TSO Command Reference)

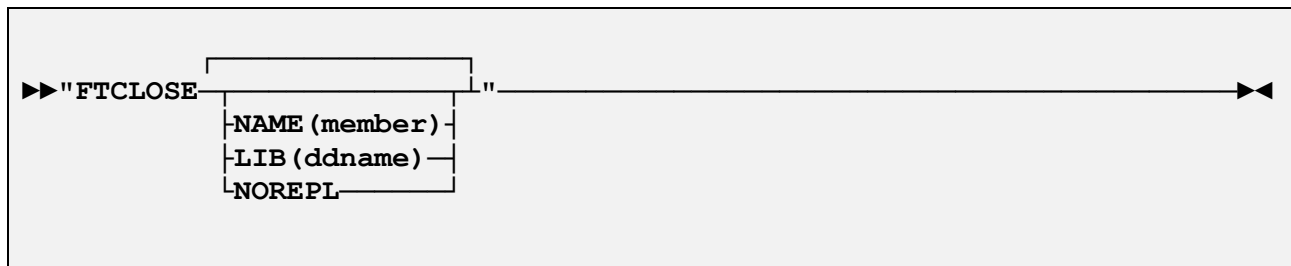
Beispiel

```
"FREE DD(SYSUT1 SYSUT2 SYSIN SYSPRINT)"
```

Return Codes

- 0 Normales Ende
- 12 Fehler, Fehlermeldung wird ausgegeben

FTCLOSE – ISPF DM Instruktion



Der FTCLOSE-Service beendet die File Tailoring Verarbeitung und bestimmt gegebenenfalls deren Ausgabe.

NAME(member)

bestimmt den Namen des Members der Ausgabebibliothek, welches die File Tailoring Ausgabe enthalten soll.

LIB(ddname)

kann abweichend zum standardgemäßen Ausgabe-DD-Namen ISPFIL einen anderen DD-Namen spezifizieren, über den die Schreiboperation erfolgen soll. Die Zuweisung des DD-Namens kann über LIBDEF-Service oder ALLOC-Statement erfolgen.

NOREPL

Falls ein gleichnamiges Member in der Ausgabebibliothek bereits existiert, darf es nicht überschrieben werden.

Return Codes

0	Normales Ende
4	Member vorhanden und NOREPL war angegeben. Das Member wird nicht überschrieben
8	Datei nicht geöffnet. FTOPEN fehlt
12	Ausgabedatei in USE
16	Keine Ausgabedatei zugewiesen
20	Schwerer Fehler

FTERASE – ISPF DM Instruktion

```
►►"FTERASE membername"_____►◄  
      [LIBRARY(ddname)]
```

Löscht ein Member aus einer File Tailoring Ausgabebibliothek. Hat die unter ISPFIL, oder einem anderen für File Tailoring zur Ausgabe vorgesehenen DD-Namen, zugewiesene Datei nicht PO-Format, wird ein schwerer Fehler gemeldet.

LIBRARY(ddname)

beschreibt abweichend vom Standard einen DD-Namen, unter dem die Ausgabebibliothek zugewiesen ist.

Return Codes

0	Normales Ende
8	Member oder Datei existiert nicht
12	Ausgabedatei in USE
16	Alternative Ausgabebibliothek nicht zugewiesen
20	Schwerer Fehler

FTINCL – ISPF DM Instruktion

```
►►"FTINCL skelname"_____►◄  
      [NOFT]
```

benennt das Skelett-File, das genutzt werden soll, um die File Tailoring Ausgabe zu erzeugen. Wird vor FTINCL kein FTOPEN durchgeführt, wird das Öffnen implizit durchgeführt (ohne das Schlüsselwort TEMP).

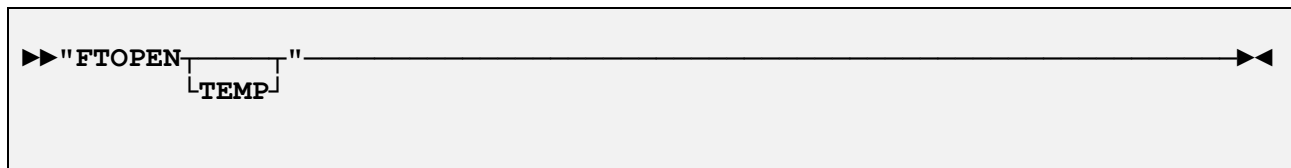
NOFT

legt fest, dass das angegebene Skeleton unmittelbar in die Ausgabe überstellt wird, ohne dass Variable substituiert, oder Steueranweisungen interpretiert werden.

Return Codes

0	Normales Ende
8	Skeleton nicht gefunden
12	Skeleton in USE
16	Daten wurden abgeschnitten oder Skeleton-Eingabe- oder -Ausgabedatei nicht zugewiesen
20	Schwerer Fehler

FTOPEN – ISPF DM Instruktion



ermöglicht dem File Tailoring Service den Zugriff auf alle Bibliotheken, die unter dem Namen ISPS-LIB zugewiesen sind.

TEMP

Ausgabe ist eine Temporärdatei. Ausgabeformat sind Sätze fester Länge von 80 Byte. Der Name der Datei kann vollqualifiziert der Systemvariablen ZTEMPF entnommen werden. In der aufrufenden Funktion ist der Variableninhalt nach VGET verfügbar. Generierte JCL kann somit in einem REXX-Programm durch

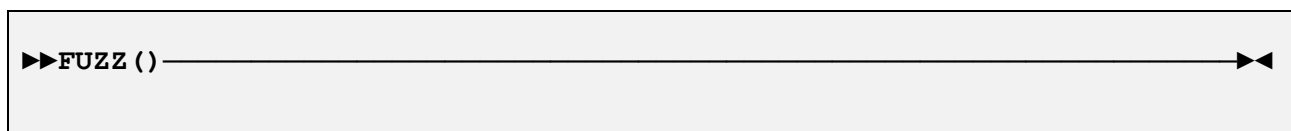
```
ADDRESS TSO "SUBMIT '"ZTEMPF'" "
```

zur Ausführung gebracht werden.

Return Codes

0	Normales Ende
8	FT bereits gestartet
16	Skeleton Ein- oder Ausgabebibliothek nicht zugewiesen
20	Schwerer Fehler

FUZZ() – REXX Built-in-Function



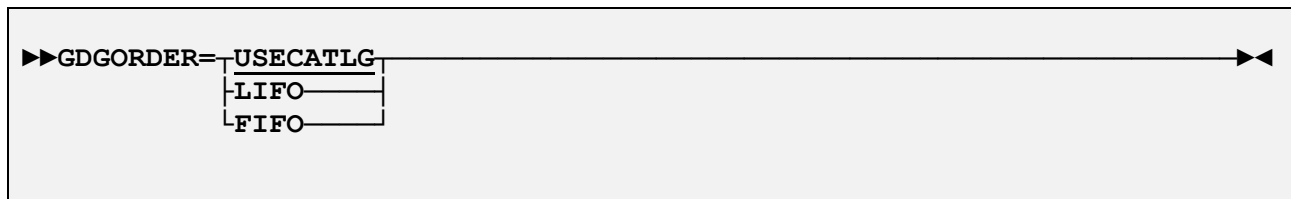
Übergibt den aktuellen Wert aus NUMERIC FUZZ.

Soll dieser Wert für einen Verarbeitungsschritt geändert werden, ist sinnvoll, den aktuellen Wert in einer Variablen abzulegen, um nach der Verarbeitung NUMERIC FUZZ auf den alten Wert zurücksetzen zu können.

Beispiel

```
:  
alt_fuzz=fuzz()  
numeric fuzz 3  
/* Verarbeitung mit dem neuen Wert von FUZZ */  
numeric fuzz alt_fuzz  
:
```

GDGORDER – JCL DD Parameter



Führt für ein GDG eine Gesamtverarbeitung durch

USCATLG

Die Reihenfolge der Verarbeitung entspricht den Einträgen im Katalog

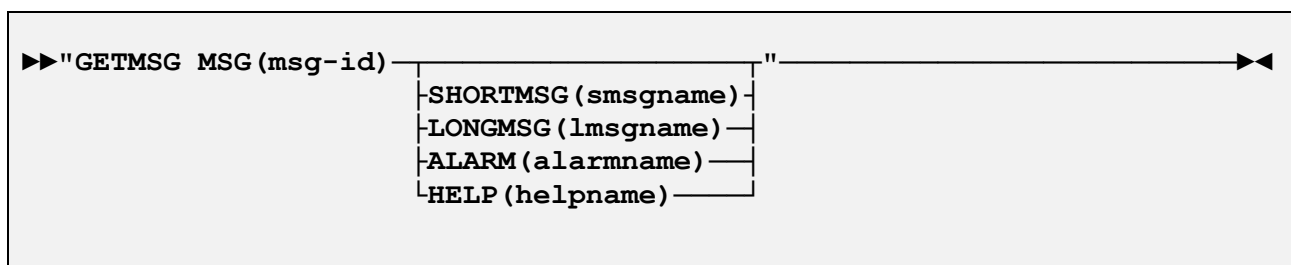
LIFO

Der jüngste Eintrag kommt als erstes, der älteste als letztes

FIFO

Der älteste Eintrag kommt als erstes, der jüngste als letztes.

GETMSG – ISPF DM Instruktion



liest eine Nachricht aus der Nachrichtendatei (ISPMLIB). Die Parameter haben folgende Bedeutung:

MSG(msg-id)

Benennt die Nachricht, die gelesen werden soll.

SHORTMSG(smsgname)

übergibt die Kurznachricht an die Variable in der Klammer.

LONGMSG(lmsgname)

übergibt die lange Meldung in die Variable in der Klammer

ALARM(alarmname)

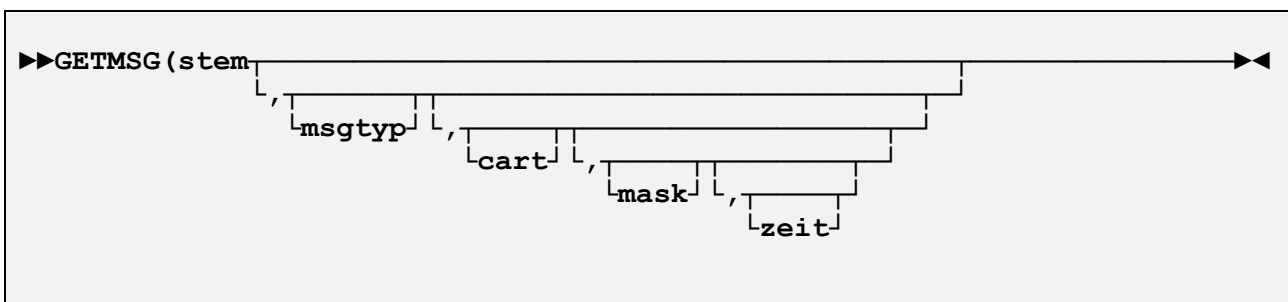
hinterlegt den Indikator für ALARM (YES oder NO) in der Variablen in der Klammer

HELP(helpname)

liefert den Namen des HELP-Panels über die Variable in der Klammer, sofern ein HELP-Panel definiert ist.

Return Codes

- 0 Normales Ende
- 12 Meldung wurde nicht gefunden
- 20 Schwerer Fehler

GETMSG() – TSO external Function

Diese Funktion wird in Verbindung mit dem TSO-CONSOLE-Kommando benutzt. Dieses wird in den meisten Unternehmen in REXX nicht benutzt, da in aller Regel die RACF-Rechte zur Ausführung fehlen. Alternativ hierzu (und mit deutlich einfacherer Syntax) kann ISFSLASH über die SDSF-API benutzt werden

Stem

Name des Stems, welcher die mit GETMSG() gelesenen Informationen aufnimmt.

Msgtyp

Kann einer der folgenden Werte sein:

SOL	Solicited Messages sind Antworten auf MVS-Commands
UNSOL	Unsolicited Messages können beispielsweise Send- oder Broadcast Messages sein
EITHER	Sowohl als auch. Either ist Default, wenn msgtyp nicht angegeben wird

Cart

Kann ein 8Byte String (oder Hex 16Byte) sein. Ist eine sehr exotische Angabe und wird in der Regel nicht gebraucht. Details hierzu sind im REXX Reference Manual unter "Using the Command and response token and mask" beschrieben.

Mask

Siehe CART

Zeit

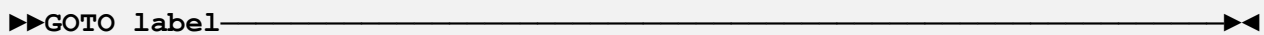
Sekunden, die GETMSG() warten soll, sofern die Information nicht unmittelbar eintrifft

Beispiel

```
GetRC = getmsg("CONSINF.", "SOL")
```

Hinweis

Ein vollständiges Beispiel ist bei der Beschreibung des CONSOLE-Commands zu finden

GOTO – ISPF Panel Instruction

Ein IF/ELSE-Gebilde kann leicht kompliziert und unübersichtlich werden. Mit GOTO (und EXIT) kann Abhilfe geschaffen werden. Erkannte Situationen können zu einem Sprung führen.

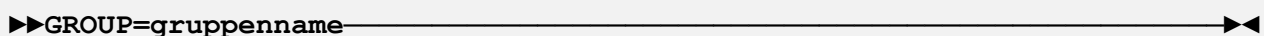
GOTO kann in)INIT-,)REINIT-,)PROC-,)ABCINIT- und)ABCPROC angewendet werden.

Hinweis

Beschränkt die Verwendung von GOTO auf diejenigen Situationen, in denen es Sinn macht. Ich möchte hier nicht den Eindruck vermitteln, dass Sprungbefehle das Mittel der Wahl sind. Ich persönlich habe die vergangenen 30 Jahre keinen kodiert.

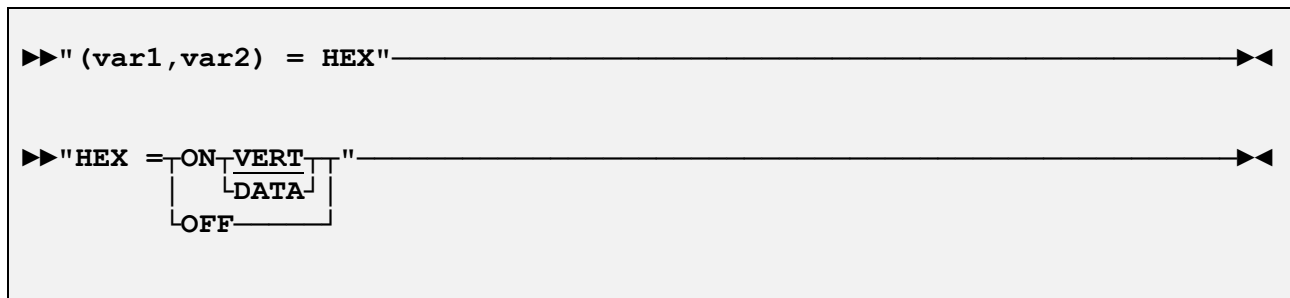
Beispiel

```
)PROC
  if (&KUNDNAM = &Z) GOTO NEUKUNDE
  ver (....
  ver (....
  GOTO WEITER
NEUKUNDE:
/* Hierhin führt der Sprung */
befehle
WEITER:
```

GROUP – JCL JOB Parameter

Identifiziert ein Gruppenmitglied bei Einsatz von RACF.

HEX – ISPF Edit Macro



Der HEX-Modus steuert die Darstellung der Daten im Hex-Format.

=HEX

Hinterlegt in der ersten angegebenen Variablen die Schalterstellung (ON oder OFF). In der zweiten Variablen wird, abhängig von der Schalterstellung, bei ON

- VERT der HEX-Code spaltengleich untereinander, oder
- DATA der HEX-Code als String (zwei HEX-Zeichen für ein Byte)

ausgegeben.

Ist der HEX-Modus ausgeschaltet, wird die zweite Variable auf Blank gesetzt.

HEX=

Der HEX-Modus wird gesetzt. Die möglichen Angaben lauten:

Beispiele

```
" (hexmodus) = HEX"
```

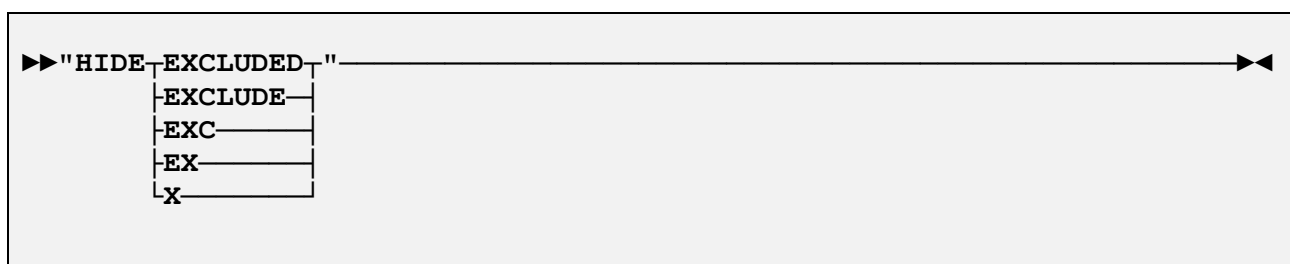
```
if hexmodus = "OFF" then ...
```

Der aktuelle Wert für HEX wird in der Variablen HEXMODUS hinterlegt und abgefragt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

HIDE – ISPF Edit Macro



HIDE unterdrückt die Hinweiszeilen "n Line(s) not displayed" und unterstreicht an Stelle dessen die Zeilennummern der betreffenden Zeilen hinter denen ausgeblendete Zeilen liegen.

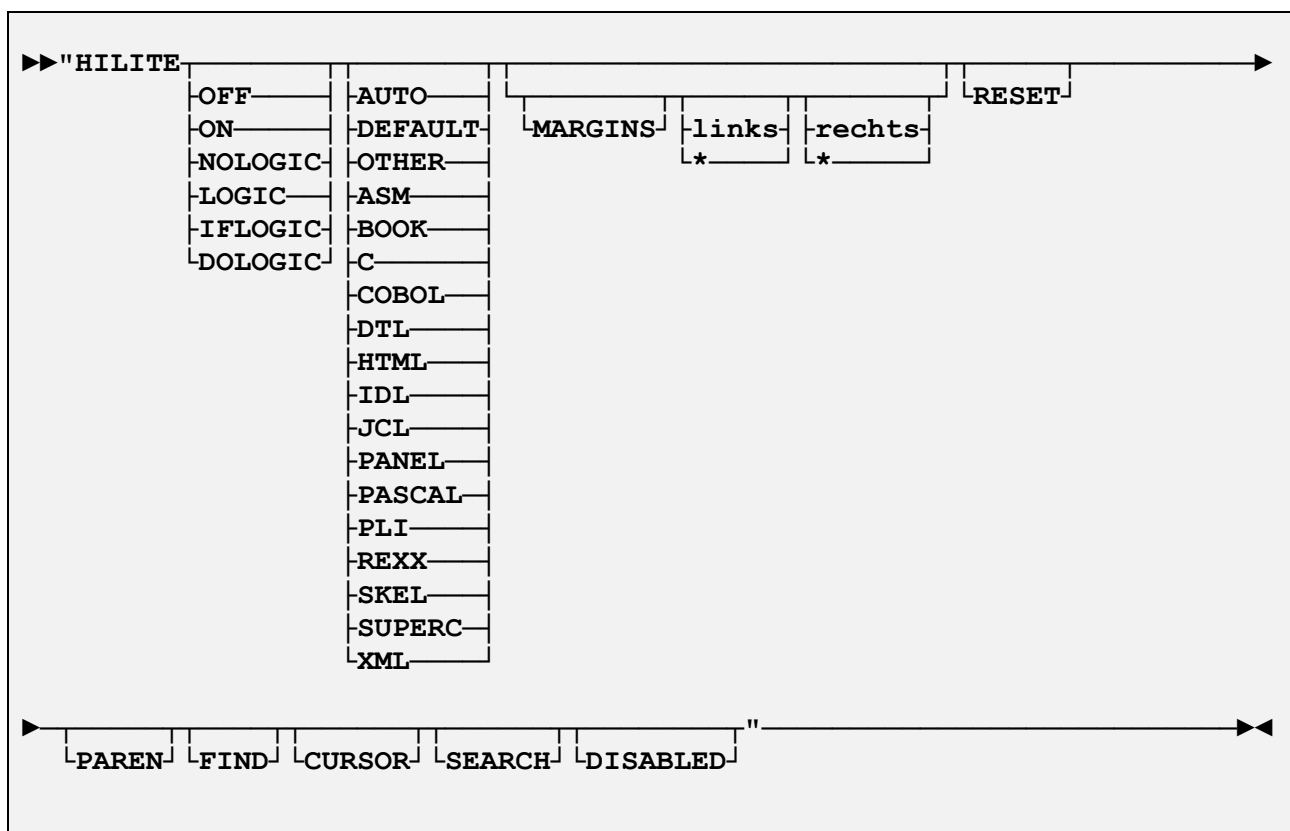
X (und alle anderen Schreibweisen)

Entfernt jede "n Line(s) not displayed"-Zeile.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

HILITE – ISPF Edit Macro



HILITE steuert den Einsatz der Farben im Editor zur Unterstützung beim Editieren von Quellcodes. Der Hilite-Dialog kann im Macro nicht aufgerufen werden.

Weitere erlaubte Schreibweisen sind HI oder HILIGHT.

ON

Farbunterstützung für Programm ON, für Logic OFF

OFF

Schaltet die Farbunterstützung aus (Ausnahme CURSOR-Hiliting)

LOGIC

Stellt sprachabhängig bestimmte Elemente in der gleichen Farbe dar. Bei REXX beispielsweise DO/END, SELECT/END. LOGIC ist gleichbedeutend mit IFLOGIC und DOLOGIC.

IFLOGIC

Unterstützt IF-ELSE-Logik

DOLOGIC

Unterstützt DO-END-Logik.

NOLOGIC

Gleichzusetzen mit HILITE OFF

AUTO

ISPF entscheidet über die Sprache

DEFAULT

Hiliting in einer Farbe.

OTHER

Hilite als Pseudo-PL/I

ASM

Unterstützt Assembler

BOOK

Unterstützt BookManager (ist syntaktisch noch möglich. Bookmanager wurde aber eingestellt)

C

Unterstützt die Programmiersprache C.

COBOL

Unterstützt Cobol.

DTL

Unterstützt Dialog Tag Language.

HTLM

Unterstützt HTML

IDL

Unterstützt IDL.

JCL

Unterstützt Job Control Language.

PANEL

Unterstützt Panels.

PASCAL

Unterstützt Pascal.

PLI

Unterstützt PL/I

REXX

Unterstützt REXX

SKEL

Unterstützt Skeletons

SUPERC

Unterstützt SuperC-Listings

XML

Unterstützt XML.

MARGINS | links | rechts | *

Legt die Spalten fest, innerhalb derer die Farbunterstützung für die jeweilige Sprache erfolgen soll.

RESET

Setzt den Default: HILITE ON AUTO FIND CURSOR

PAREN

Unterstützt paarige Klammern.

FIND

Hebt FIND-Ergebnisse hervor.

CURSOR

Hebt das Wort in dem der CURSOR steht hervor.

SEARCH

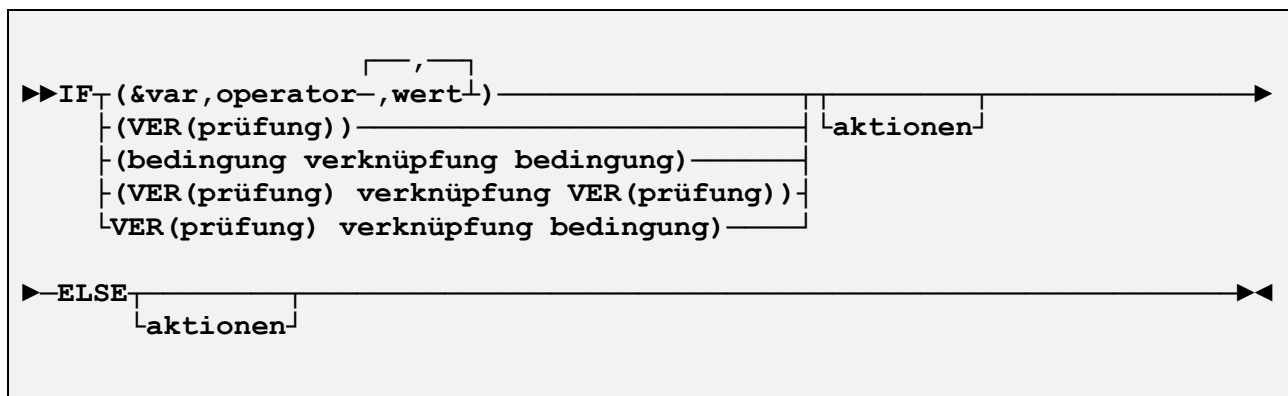
Findet das erste unpassende END, ELSE, }, oder). SEARCH ist nicht erlaubt, wenn DEFAULT gesetzt ist.

DISABLE

Beendet alle HILITE-Features und entfernt alle Action-Bars. Erfolgt ein HILITE DISABLE in einem Macro, kann das gleiche Macro den alten Zustand durch ein HILITE RESTORE nicht wieder herstellen.

Return Codes

- 0 Normales Ende
- 8 LOGIC oder SEARCH wird im aktuellen Environment nicht unterstützt oder falsche Sprache
- 12 HILITE-Dialog aus einem Edit Macro gerufen (geht nicht) oder HILITE nicht verfügbar (wegen der Installations-Defaults) oder Das benutzte Edit Panel unterstützt keine Shaddow Variablen
- 20 Schwerer Fehler

IF – ELSE – ISPF Panel Instruction

Erlaubt Ablaufsteuerung innerhalb eines Panels. Die Kodierung ist spaltensensitiv. Die an den JA-/NEIN-Zweig geknüpften Aktionen müssen eingerückt geschrieben werden. Der jeweilige Zweig wird als beendet betrachtet, sobald eine Instruktion spaltengleich, oder links von IF oder ELSE, kodiert wird.

Um zu einem logischen TRUE oder FALSE der Abfrage zu kommen können Bedingungen aufgestellt oder Verifikationen durchgeführt werden. Beides kann mehrfach erfolgen und miteinander kombiniert werden.

Mehrfachbedingungen oder -prüfungen werden durch Boolesche Operanden miteinander verbunden.

Operator

Grafikzeichen	Buchstaben	Bedeutung
=	EQ	Gleich
^=	NE	Ungleich
<	LT	Kleiner als
<=	LE	Kleiner oder gleich
^<	NL	Nicht kleiner als
>	GT	Größer als
>=	GE	Größer oder gleich
^>	NG	Nicht größer als

wert

kann eine Konstante (Zeichenfolge) oder eine Variable sein.

Verknüpfung

Um Bedingungen oder Verifys miteinander zu verknüpfen stehen die Booleschen Operanden zur Verfügung:

Grafikzeichen	Buchstaben	Bedeutung
&	AND	UND-Verknüpfung
!	OR	ODER-Verknüpfung

Beispiele

```
IF (VER(&var1,NB,NUM) AND VER(&var2,NB,NUM))
:
ELSE
  IF (VER(&var3,NB,NUM) OR &var4 = 100)
  :
```

Die Bedingung des ersten IF-Befehles wird als erfüllt angesehen, wenn Variable1 und Variable2 numerische Inhalte haben. Für die Rückmeldung TRUE im zweiten IF-Befehl genügt es, wenn Variable3 numerisch ist, oder der Inhalt der Variablen4 100 ist.

```
IF (VER(&var1,NB,NUM) & VER(&var2,NB,NUM) &
    &var3,lt,6 ! &var4,=,9 ! &var5,GT,&var6)
:
ELSE
  .MSG=ERR001A
```

Die Bedingung ist wahr, wenn die ersten drei Ausdrücke alle zutreffend sind, oder die vierte oder fünfte Bedingung stimmt.

Nicht alles was sich syntaktisch kodieren lässt ist unbedingt empfehlenswert.

IF – THEN – ELSE – JCL Special Statement

```
//label IF ausdruck THEN  
//*statements  
//label ELSE  
//*statements  
//label ENDIF
```

Die IF-THEN-ELSE-Konstruktion erlaubt in weit komfortablerer Art die Ablaufsteuerung in einer Job Control. Alternativ zu den Möglichkeiten der CONDition-Steuerung kann hier auf das Auftreten bestimmter Return Codes oder auch auf Abnormal End Bezug genommen werden.

Der Vorteil der IF-Konstruktion ist, dass hier sowohl eine Laufbedingung (THEN), als auch eine Negation (ELSE) kodiert werden kann. Diese Form ist sehr viel sprechender als eine vergleichbare COND-Bedingung.

label

Die Labels müssen nicht kodiert werden, verbessern gegebenenfalls aber die Lesbarkeit. IF, ELSE und ENDIF dürfen aber nicht auf Stelle drei stehen.

THEN

bestimmt den JA-Zweig der Abfrage. Die folgenden Anweisungen kommen zur Ausführung, wenn die aufgestellte Bedingung TRUE ist.

ELSE

bestimmt den NEIN-Zweig der Abfrage. Die folgenden Anweisungen werden ausgeführt, wenn die Bedingung FALSE ist. Eine ELSE-Klausel ist formal nicht zwingend.

ENDIF

Schließt die IF-Konstruktion ab.

ausdruck

Beschreibt einen logischen Ausdruck, über welchen ein vorangegangenes ABEND-Ereignis und/oder ein vorausgegangenes RC-Ergebnis abgeprüft werden kann. Wird auf den Return Code des letzten vorausgegangenen Steps geprüft, kann dieser unmittelbar über die Variable RC untersucht werden. Soll auf einen RC eines weiter zurückliegenden Steps Bezug hergestellt werden, ist zu kodieren stepname.RC.

Bei einfachen Ausdrücken können die Klammern entfallen. Verknüpfte Ausdrücke müssen in Klammern eingeschlossen werden.

Soll ein Ausdruck über mehrere Zeilen kodiert werden, sind die fortzusetzenden Zeilen mit BLANK abzuschließen und die Fortsetzungszeilen zwischen Spalte 4 und 16 weiter zu kodieren. In fortzusetzenden Zeilen dürfen keine Kommentare kodiert werden (ist nur in der letzten Zeile der Anweisung möglich).

Bei Überprüfung eines Step-RC wird zunächst überprüft, ob der Step ausgeführt wurde. Ist dies nicht der Fall, wird der entsprechende Ausdruck als FALSE interpretiert.

Folgende Operatoren für Ausdrücke stehen zur Verfügung (die Zeichen für NICHT und ODER können abhängig von der Emulation anders sein):

Koding	Bedeutung	Priorität
NOT oder ¬	Negation (NICHT)	Erste
GT oder >	Größer als	Zweite
LT oder <	Kleiner als	Zweite
NG oder ¬>	Nicht größer	Zweite
NL oder ¬<	Nicht kleiner	Zweite
EQ oder =	Gleich	Zweite
NE oder ¬=	Ungleich	Zweite
GE oder >=	Größer oder gleich	Zweite
LE oder <=	Kleiner oder gleich	Zweite
AND oder &	UND	Dritte
OR oder	ODER	Dritte

NOT Operator

NOT wird benutzt, um einen Ausdruck in sein Gegenteil zu kehren. Die Statements NOT1 und NOT2 des folgenden Beispiels prüfen dasselbe. Die Bedingung ist TRUE, wenn der Return Code zwischen 0 und 8 liegt:

```
//NOT1 IF ¬(RC > 8) THEN
//NOT2 IF (RC <= 8) THEN
```

Die Statements NOT3 und NOT4 liefern TRUE, wenn der RC einen der folgenden Werte aufweist: 0,1,2,3,4,5,6,7,9,10,...

```
//NOT3 IF ¬(RC = 8 | RC > 10) THEN
//NOT4 IF (RC ¬= 8 & RC <= 10) THEN
```

Die Nutzung des NOT-Operators hat zur Folge, dass sowohl die Logik als auch der Ausdruck umgekehrt werden. Zwischen NOT und dem Ausdruck wird kein BLANK kodiert.

Schlüsselworte in Ausdrücken

Schlüsselwort	Bedeutung
RC	Zeigt den Return Code an
ABEND	Zeigt eine ABEND-Bedingung
¬ABEND	Zeige KEINE ABEND-Bedingung an
ABENDCC	Zeigt auf einen System- oder User-ABEND-Code
RUN	Zeigt, dass die Verarbeitung angelaufen ist
¬RUN	Zeigt, dass die Verarbeitung NICHT angelaufen ist
***	Kommentare oder JES-Anweisungen

RC

Der Ausdruck vergleicht einen Return Code.

stepname.RC

Der Ausdruck vergleicht den RC eines bestimmten Steps.

stepname.procstepname.RC

Der Ausdruck vergleicht den RC eines Steps innerhalb einer Prozedur.

ABEND oder ABEND=TRUE**stepname.ABEND****stepname.ABEND=TRUE****stepname.procstepname.ABEND****stepname.procstepname.ABEND=TRUE**

Ohne Angabe eines Stepnamens wird auf eine im gesamten Vorlauf des Jobs aufgetretene ABEND-Situation geprüft. Mit Step-Angabe wird das ABEND eines bestimmten Steps oder Prozedur-Steps geprüft.

¬ABEND oder ABEND=FALSE**¬stepname.ABEND****stepname.ABEND=FALSE****¬stepname.procstepname.ABEND****stepname.procstepname.ABEND=FALSE**

Ohne Angabe eines Stepnamens wird geprüft, ob nirgends im gesamten Vorlauf des Jobs eine ABEND-Situation aufgetreten ist. Mit Step-Angabe wird das NICHT-ABEND eines bestimmten Steps oder Prozedur-Steps geprüft.

ABENDCC=Sxxx**ABENDCC=Uxxxx****stepname.ABENDCC=Sxxx****stepname.ABENDCC=Uxxxx****stepname.procstepname.ABENDCC=Sxxx****stepname.procstepname.ABENDCC=Uxxxx**

Zeigt an, dass ein ABEND-System oder -Usercode zu prüfen ist. Sxxx (dreistelliger Hexwert) zeigt auf einen Systemcode, Uxxxx (vierstelliger Dezimalwert) zeigt auf einen Usercode. Über ABENDCC=S0C4 könnte beispielsweise ein Adressierungsfehler eines Programmes abgeprüft werden.

stepname.RUN**stepname.RUN=TRUE****stepname.procstepname.RUN oder****stepname.procstepname.RUN=TRUE**

Lässt erkennen, dass die Verarbeitung eines Steps angelaufen ist. Auf diese Weise kann erkannt werden, ob ein Step unter Umständen aufgrund einer COND-Bedingung oder IF-Abfrage nicht zur Ausführung kam. In Verbindung mit einer RUN-Abfrage muss immer eine Step-Angabe erfolgen.

¬stepname.RUN**stepname.RUN=FALSE****¬stepname.procstepname.RUN****stepname.procstepname.RUN=FALSE**

Auf diese Weise lässt sich erkennen, dass die Verarbeitung eines Steps nicht angelaufen ist. In Verbindung mit einer RUN-Abfrage muss immer eine Step-Angabe erfolgen.

Angabe der Klammern in Ausdrücken

Durch Klammerung von Ausdrücken kann die Reihenfolge der Auflösung abweichend vom Standard geregelt werden. Das folgende Beispiel erkennt einen RC 0, 1, 2 oder 3

```
//TEST IF (RC LT 4 & (RC LT 12 | RC = 16)) THEN
```

Wird dieser Ausdruck nun geklammert, würden die RCs 0, 1, 2, 3, und 16 erkannt werden:

```
//TEST IF ((RC LT 4 & RC LT 12) | RC = 16) THEN
```

Kommentarfelder

Kommentare können hinter THEN, ELSE und ENDIF im Anschluss an ein BLANK kodiert werden..

Standort in der JCL

Eine IF/THEN/ELSE/ENDIF-Konstruktion kann an beliebiger Stelle nach der ersten EXEC-Anweisung kodiert werden.

Hinweis

Wird innerhalb einer IF-THEN-ELSE-Konstruktion eine EXEC-Anweisung mit COND-Bedingung kodiert, muss für eine Verarbeitung des betreffenden Steps die IF-Bedingung wahr und die COND-Bedingung unwahr sein.

IF-THEN-ELSE wird nicht ausgeführt, wenn ein ABEND aufgetreten ist und der Ausdruck kein ABEND, ABENDCC, oder -ABEND enthält.

Wenn im FALSE-Fall der Abfrage keine spezielle Verarbeitung gewünscht ist, kann auf die Angabe der ELSE-Anweisung verzichtet werden.

Im THEN- und ELSE-Zweig muss mindestens eine EXEC-Anweisung vorkommen.

Folgende Anweisungen sind innerhalb IF-THEN-ELSE nicht erlaubt:

- JOB Statement
- JCLLIB Statement
- JOBCAT und JOBLIB Statement
- SYSCHK (job level) Statement
- XMIT JCL Statement

Beispiele

```
//JOBA JOB ...  
//STEP1 EXEC PGM=PROG1  
:  
//IFBAD IF (ABEND OR STEP1.RC > 8) THEN  
//FEHLER EXEC PROC=ERROR  
//IFBADEND ENDIF  
//NEXTSTEP EXEC PROC=CONTINUE
```

Das IF-Statement IFBAD führt im Falle TRUE zum Aufruf der Prozedur ERROR, sofern ein ABEND vorausging oder der RC aus STEP1 größer 8 war. Anderenfalls wird der Step FEHLER übersprungen und die Verarbeitung wird mit NEXTSTEP fortgesetzt.

```
//JOBA JOB ...  
//STEP1 EXEC PGM=PROG1  
:  
//IFBAD IF (ABEND OR STEP1.RC > 8) THEN  
//TRUE EXEC PROC=ERROR  
// ELSE  
//IFBAD ENDIF  
//NEXTSTEP EXEC PROC=CONTINUE
```

Die gleiche Verarbeitung mit einem kodierten ELSE-Statement, welches zwar nicht zwingend erforderlich ist, aber die Ablauflogik besser verdeutlicht und gegebenenfalls die Wartbarkeit verbessert.

IF – THEN – ELSE – REXX Instruction

```
IF ausdruck THEN befehl  
      ELSE befehl
```

IF erlaubt eine Bedingung zu überprüfen. Trifft diese zu, wird der Befehl hinter THEN ausgeführt. Sollen mehrere Befehle ausgeführt werden, sind diese zwischen DO und END anzugeben.

Soll im Falle der Nichterfüllung der Bedingung eine gesonderte Verarbeitung durchlaufen werden, kann diese unter ELSE beschrieben werden.

Beispiel

```
if zahl >= 20 & zahl <= 40 then say 'ZAHL ist OK'  
else do  
    say 'ZAHL falsch, 30 wird gesetzt'  
    zahl=30  
end
```

IMACRO – ISPF Edit Macro

```
►►" (variable) = IMACRO"-----►►  
  
►►"IMACRO = macroname"-----►►
```

Erfragt oder bestimmt ein Initialisierungsmacro für das Profile.

=IMACRO

Überstellt den Namen des gültigen Initialisierungsmakros der angegebenen Variablen.

IMACRO=

Legt ein Initialisierungsmacro fest und trägt den Namen in das aktuelle Profil ein.

Beispiel

```
"imacro = none"
```

Es soll kein Initialisierungsmacro ausgeführt werden.

Return Codes

0	Normales Ende
4	Profil gesperrt, kein Eintrag
12	Falscher Macroname
20	Schwerer Fehler

INCLUDE – JCL Special Statement

```
// [label] INCLUDE MEMBER=name [kommentar]
```

Die INCLUDE-Anweisung wird benutzt, um vorbereitete JCL analog zu den Möglichkeiten des File-Tailorings unter TSO in eine andere JCL zu implementieren. Im Unterschied zu katalogisierten Prozeduren können hier beliebige JCL-Fragmente zusammengemischt werden.

Name

Name des Members der JCL-Prozedur-Bibliothek, welches an die Stelle der INCLUDE-Anweisung tritt.

Beispiel

```
//DEMO5 EXEC PGM=PROG1
//      INCLUDE MEMBER=ISPFLIBS          ALLOC ISPF-LIBs
```

Hinweise:

INCLUDEs werden aus der Standard-PROCLIB gezogen.

Sofern es ein JCLLIB-Statement gibt, müssen die INCLUDE-Statements nach dem JCLLIB-Statement kodiert werden.

In INCLUDEs dürfen keine JOB-, PROC-, PEND-, JCLLIB-, DD-DATA-Statements vorkommen.

INDEX() – NonSAA REXX Built-in-Function

```
►►INDEX(string,suchbegriff[ ,startpos])◄◄
```

Sucht nach einer Teilzeichenfolge in einem String. Wenn suchbeginn nicht angegeben wird, wird der betreffende String ab Position 1 durchsucht. Rückgemeldet wird die Position im String, an der der Suchbegriff gefunden wurde.

INDEX() gehört nicht zum SAA-Befehlssatz von REXX und ist auf anderen Plattformen nicht verfügbar. Sinnvoller wäre die Nutzung der Funktion POS()

Beispiele

```
SAY INDEX('Kniescheibe','nie')      2
SAY INDEX('Syntaxaufbau','au')      7
SAY INDEX('Syntaxaufbau','au',10)   11
SAY INDEX('Syntaxaufbau','abc')      0
```

```
:
say 'Zu durchsuchende Zeichenfolge'
pull str1
say 'Suchkriterium'
pull str2
if index(str1,str2) > 0 then
    say str2 'gefunden an Stelle' index(str1,str2)
    else say str2 'in' str1 'nicht vorhanden'
:
```

Hinweis

Die Funktion POS() ist der INDEX-Funktion vorzuziehen. Sie liefert das gleiche Ergebnis, gehört aber entgegen INDEX zum SAA-Befehlssatz.

INSERT – DB2 Instruction

```
►►"INSERT INTO [schema.]table ([tab-feld])VALUES ([host-var])"◄◄
```

Wird benutzt, um eine Zeile in eine Tabelle einzufügen.

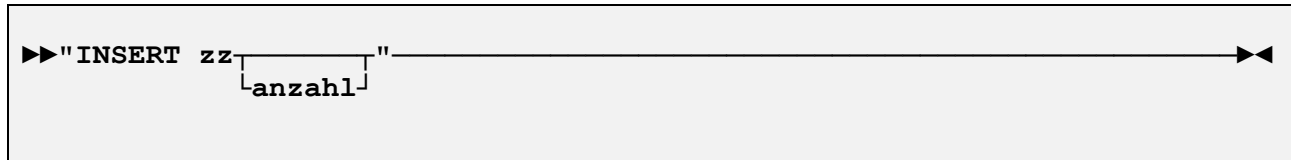
(tab-feld)

Liste von Tabellenfeldern, die inserted werden sollen.

VALUES(host-var)

Liste von Programmvariablen, deren Inhalte in der Tabelle landen sollen.

Die Liste der Hostvariablen muss von der Reihenfolge her zur Liste der Tabellenfeldern passen.

INSERT – ISPF Edit Macro

Liefert eine oder mehrere Leerzeilen entsprechend der Maskenzeile, die der Anwender beschreiben kann. Einfügezeilen sind keine Datenzeilen und können von einem Macro nicht adressiert werden. Werden keine Daten eingegeben, werden diese Zeilen bei ENTER gelöscht.

Es sollte sichergestellt sein, dass die Zeile, auf die sich der INSERT-Befehl bezieht, sichtbar ist, da anderenfalls die Leerzeile für den Anwender nicht sichtbar sein wird.

Um durch ein Macro neue Zeilen in eine Datei einzuarbeiten, müssen die Zuweisungsbefehle `LINE_BEFORE` und `LINE_AFTER` benutzt werden.

zz

Zeilenzeiger (Label oder relative Zeilennummer) auf den sich der INSERT-Befehl bezieht.

anzahl

Anzahl Leerzeilen, die eingefügt werden sollen.

Beispiele

`"insert .anfang"`

Es wird eine Leerzeile hinter der Zeile mit dem Label `.ANFANG` eingefügt.

`"insert .zfirst"`

Am Anfang der Datei wird eine Leerzeile eingefügt.

`"insert .zlast 5"`

An das Ende der Datei werden fünf Leerzeilen angefügt.

Return Codes

0	Normales Ende
12	Falsche Zeilennummer
20	Schwerer Fehler

INSERT() – REXX Built-in-Function

```

>>INSERT(string1,string2,
           ,pos,
           ,länge,füll)

```

Fügt eine Zeichenfolge (string1) in eine bestehende Zeichenfolge (string2) an der angegebenen Position ein. Ist die Längenangabe größer als die einzufügende Zeichenfolge, wird mit dem angegebenen Füllzeichen aufgefüllt (Standard ist BLANK).

Beispiele

SAY INSERT('ist ','Hans doof',6)	'Hans ist doof'
SAY INSERT(' doof','Hans ist')	'Hans ist doof'
SAY INSERT('123','abc',5)	'abc 123'
SAY INSERT('123','abc',5,6)	'abc 123 '
SAY INSERT('123','abc',5,6,'-')	'abc--123---'
SAY INSERT('123','abc',5,, '-')	'abc--123'

INTERPRET – REXX Instruction

```

>>INTERPRET ausdrück

```

Interpretiert einen Ausdruck als Befehl und führt ihn aus.

Beispiel

```

/* REXX *****/
do forever
  say "REXX-Befehl eingeben (ENTER = Ende)"
  pull cmd
  if length(cmd)=0 then leave
  interpret cmd
end
exit

```

```

/* REXX *****/
Games = "Knobeln BlakJack Poker"
Say "1 => Knobeln"
Say "2 => Black Jack"
Say "3 => Pokern"
Say "Was willst Du spielen?"
Parse external game
If wordpos(game,"1 2 3") = 0 then exit

```

```
Interpret "call" word(games,game)
```

Hinweis

Interpret sollte mit Bedacht benutzt werden. Die Anwendung von INTERPRET wird in den meisten Unternehmen ähnlich kritisch betrachtet wie Dynamic SQL. In 98% der Anwendungsfälle genügt die Value-Funktion. Das Beispiel zeigt eine von ganz wenigen Ausnahmen. Ein dynamischer Call ist mit VALUE() nicht möglich.

ISFACT – SDSF-API-Aufruf

```
►►"ISFACT cmd TOKEN(token) PARM(NP lcmd) "◄◄
```

Mit ISFACT können Linecommands ausgeführt werden. Wir wollen hier nur die Ausführung von Linecommands betrachten, Generell wären auch Dinge wie das Überschreiben von Variablen mit beliebigen Werten durchzuführen. Bemüht für "EXOTISCHE" Dinge das Manual "z/OS SDSF User's Guide".

cmd

Als cmd muss der gleiche Befehl (das gleiche Panel) aufgerufen werden, wie in einer vorangegangenen ISFEXEC-Anweisung.

Token

Identifiziert das Element im SDSF und wird vom vorangegangenen ISFEXEC-Aufruf bereitgestellt.

PARMS(NP lcmd)

Ist die Ausführung eines Linecommands auf das entsprechende SDSF-Element. Die gängigsten über NP ausführbaren Linecommands für Informationen aus den Status Panel sind folgende:

Linecommand	Bedeutung
SA	Allocate Spool-Datasets
SJ	Browse JCL
XD, XDC	Druckausgabe eines LOGs
XF, XFC	Druckausgabe in ein File
P	Purge

Return Codes

- 0 Normales Ende
- 4 Die Anforderung wurde ausgeführt aber nicht mit allen Funktionen
- 8 Falsche Parameter
- 12 Syntaxfehler
- 16 Der Benutzer hat keine Berechtigung SDSF aufzurufen
- 29 Environment Fehler
- 24 Speicherüberlauf

ISFBROWSE – SDSF-API-Aufruf

```
►►"ISFBROWSE cmd TOKEN(token) [options]"◄◄
```

Für die Anzeige von SDSF-Elementen. Generell gäbe es hier verschiedene Methoden, aber ISFBROWSE ist die komfortabelste. Nach dem Aufruf von ISFBROWSE steht der Stem ISFLINE. Zur weiteren Verarbeitung zur Verfügung.

(options)

Es sind einige Optionen möglich.

Option	Bedeutung
JCL	Allocate Spool-Datasets
NOCLOSE	Schließt die Dateien nicht, um bei mehrfachen Zugriffen den Overhead durch OPEN und CLOSE gering zu halten.
VERBOSE	Befüllt den Stem ISFMSG2. Mit einer Vielzahl zusätzlicher Informationen. Diese können bei der Fehleranalyse hilfreich sein.

Return Codes

- 0 Normales Ende
- 4 Die Anforderung wurde ausgeführt aber nicht mit allen Funktionen
- 8 Falsche Parameter
- 12 Syntaxfehler
- 16 Der Benutzer hat keine Berechtigung SDSF aufzurufen
- 29 Environment Fehler
- 24 Speicherüberlauf

ISFEXEC – SDSF-API-Aufruf

```
►►"ISFEXEC panelname [command]"◄◄
```

Mit ISFEXEC kann ein Panelaufruf erfolgen. Abhängig vom Panel werden eine Reihe paralleler Stems aufgebaut, welche den Spaltenüberschriften im Panel entsprechen. Welche Felder gefüllt wurden, ist in der Variablen ISFCOLS hinterlegt.

Wird ISFCOLS vor dem Aufruf von ISFEXEC befüllt, werden nur diejenigen Stems aufgebaut, welche in ISFCOLS hinterlegt sind.

Die Möglichkeit, mit ISFEXEC ZOS- oder JES-Befehle absetzen zu können, wird hier nicht weiter beschrieben. Dafür kann seit einiger Zeit ISFSLASH benutzt werden (bitte hier nachlesen).

Beispiel

```
DO $=1 TO 10
  IF $ // 3 = 0 THEN ITERATE
  SAY $          /* Zeigt 1, 2, 4, 5, 7, 8, und 10*/
END
```

ISFGET – SDSF-API-Aufruf

```
►►"ISFGET parameter"◄◄
```

Die Idee wäre, einen bestimmten Eintrag für ein SDSF-Element nachzulesen (vergleichbar mit dem zeitgleichen Umgang mehrerer ISPF-Tabellen, welche zumindest in Teilen gleichnamige Tabellenfelder haben. Hier würde es zu einem Überschreiben im Funktion Pool kommen, sodass gegebenenfalls eine Zeile einer bestimmten Tabelle mit TBGET nachgelesen werden muss.)

Darauf soll hier nicht weiter eingegangen werden.

ISFLOG – SDSF-API-Aufruf

```
►►"ISFLOG [ALLOC | READ] TYPE (SYSLOG | OPERLOG) [(option)]◄◄
```

Mit ISFLOG kann sowohl auf das SYSLOG des aktuellen MVS als auch das plexweite Operlog zugegriffen werden.

ALLOC

Das Syslog wird für die Verarbeitung via EXECIO zugewiesen. Die Variablen ISFDDNAME und ISFDSNAME werden gefüllt

READ

Das LOG wird im Stern ISFULOG. Abgelegt. ISFULOG.0 enthält die Anzahl der gebildeten ISFULOG.-Variablen. Folgende Variablen können das Ergebnis beeinflussen:

Variable	Bedeutung
ISFLINELIM	Maximale Anzahl der Output-Lines
ISFLOGSTARTTIME	Uhrzeit ab der das LOG gelesen werden soll
ISFLOGSTARTDATE	Datum ab dem das LOG gelesen werden soll
ISFLOGSTOPTIME	Endezeit
ISFLOGSTOPDATE	Endedatum

TYPE(SYSLOG)

Das SYSLOG eines Single MVS soll gelesen werden. Über die Variable ISFSYSID kann das gewünschte MVS benannt werden.

TYPE(OPERLOG)

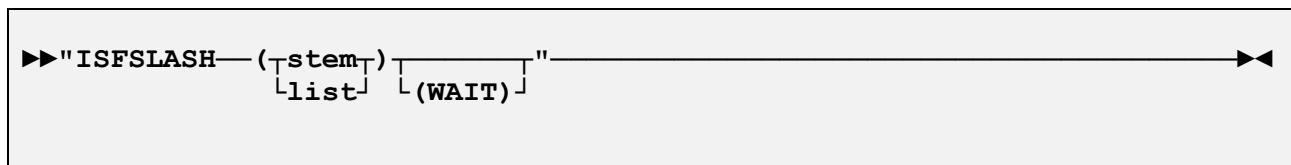
Gelesen wird das plexweite OPERLOG

Hinweis

Es werden jede Menge weiterer Variablen gefüllt, welche ausgewertet werden können. Dies sind beispielsweise ISFCOLOR, ISFDATE (Datumsformat) und viele mehr.

Return Codes

0	Normales Ende
4	Die Anforderung wurde ausgeführt aber nicht mit allen Funktionen
8	Falsche Parameter
12	Syntaxfehler
16	Der Benutzer hat keine Berechtigung SDSF aufzurufen
29	Environment Fehler
24	Speicherüberlauf

ISFSLASH – SDSF-API-Aufruf

Mit ISFSLASH können beliebige Befehle an das MVS zur Ausführung übergeben werden. Egal, wie es sonst noch ginge, ob über ISFEXEC ULOG oder die Consol-Schnittstelle im TSO, benutzt am Sinnvollsten nur noch diese Art.

Die auszuführenden Befehle können entweder in einem Stem vorbereitet oder als Liste übergeben werden. Bei der Stem-Variante ist darauf zu achten, dass die Anzahl der Stem-Inkremente immer in der .0-Variablen definiert sein muss. Dennoch ist das in der Praxis die bessere Lösung, weil sie übersichtlicher und pflegeleichter ist.

WAIT

Es soll eine bestimmte Zeit gewartet werden, bevor der Command Output eingefangen wird. Der Parameter sollte unbedingt angegeben werden. Die gewünschte Wartezeit kann über die Variable ISFDELAY gesetzt werden.

Special Variables

Im Zusammenhang mit ISFSLASH können einige Variablen hilfreich sein:

Variable	Bedeutung
ISFDELAY	Wartezeit vor Einholung des Command Outputs
ISFCMDLIM	Maximale Anzahl der an ISFSLASH zu übergebenden Befehle. 0 ist ohne Limit. Der Maximalwert ist 99.999.999 (bravo).
ISFCONMODE	Analog zu SET CONMODE in der SDSF-ISPFL-Oberfläche. Beim Zugriff via REXX sollte der Consol Mode niemals disabled sein.
ISFCONS	Wenn benutzt muss es einen UNIQUE Consolnamen enthalten
ISFULOG	Stem mit dem Ergebnis. In ISFULOG.0 steht die Größe des Stems.

Beispiel

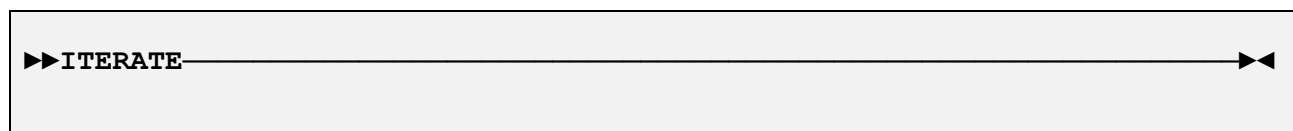
```

:
CMD.==1
CMD.1 = "D A,L"
"ISFSLASH ("cmd.") (WAIT)"
DO $=1 TO ISFOLOG.0
  /* Was auch immer mit dem Command Response passieren soll */
END

```

Return Codes

- 0 Normales Ende
- 4 Die Anforderung wurde ausgeführt aber nicht mit allen Funktionen
- 8 Falsche Parameter
- 12 Syntaxfehler
- 16 Der Benutzer hat keine Berechtigung SDSF aufzurufen
- 29 Environment Fehler
- 24 Speicherüberlauf

ITERATE – REXX Instruction

Steuert den Fluss in einer Verarbeitungsschleife durch DO und führt zum Rücksprung an den Beginn der DO-Schleife, ohne die nachfolgenden Instruktionen auszuführen.

Beispiel

```
DO $=1 TO 10
```

```
IF $ // 3 = 0 THEN ITERATE
SAY $          /* Zeigt 1, 2, 4, 5, 7, 8, und 10*/
END
```

JCLLIB – JCL Special Statement

```
// [label] JCLLIB ORDER=( [library] )
```

Mit JCLLIB können private JCL-Bibliotheken zur Nutzung von Prozeduraufrufen oder INCLUDEs benutzt werden. Die Suchreihenfolge entspricht der Schreibweise der Libraries von links nach rechts.

JCLLIB ORDER=(...

Wird nur eine Bibliothek angegeben, können die Klammern entfallen.

Hinweise:

JCLLIB liegt zwischen JOB und der ersten EXEC-Anweisung, vor INCLUDEs und Prozeduraufrufen, welche die Information aus JCLLIB benötigen.

Es gibt nur EIN JCLLIB-Statement je JOB.

JCLLIB darf nicht in INCLUDEs vorkommen.

Beispiele

```
//MYLIB JCLLIB ORDER=MY.PROC1
```

Die Datei MY.PROC1 wird vor den Standard-PROCLIBs berücksichtigt.

```
//MYLIB JCLLIB ORDER=( 'MY.PROC1' , 'MY.PROC2' , 'MY
// .PROC3' , 'MY.PROC4' , 'MY.PROC5' )
```

Sind mehrere private Bibliotheken zu verknüpfen, kann das Statement bis Spalte 71 geschrieben und in der Folgezeile nach zwei führenden Slashes(//) zwischen Spalte 4 und 16 fortgeführt werden.

JOBLIB – JCL Special DD Statement

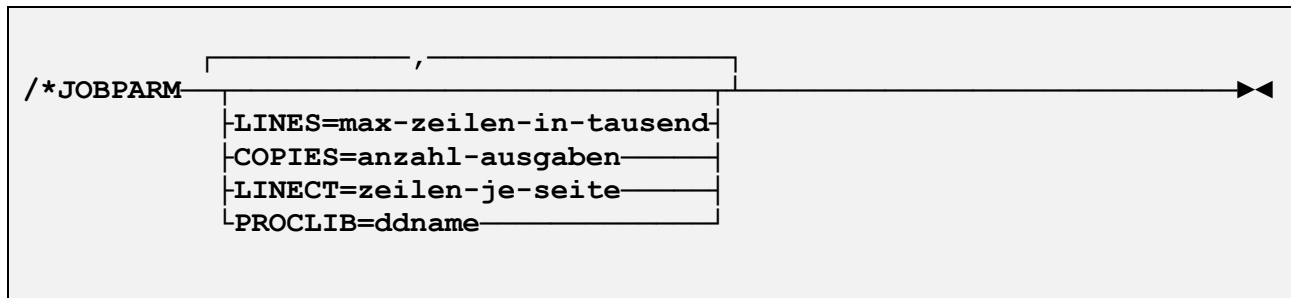
```
//JOBLIB DD parameter
```

Die JOBLIB-DD-Anweisung zeigt auf eine oder mehrere Lademodul-Bibliotheken, aus denen Programme aufzurufen sind. Die Definition ist für den gesamten JOB gültig. Sie wird gebraucht, wenn

die Programme über die LINKLIST nicht gefunden werden, oder wenn sie gezielt aus einer bestimmten Bibliothek geladen werden sollen (z.B. beim Testen).

JOBLIB muss vor der ersten EXEC-Anweisung liegen.

JOBPARM – JCL JES Instruction



Mit `/*JOBPARM` werden Anweisungen direkt an JES2 übergeben. Die Anweisung ist einzeilig. Bei Bedarf müssen mehrere Anweisungen kodiert werden.

LINEs=max-zeilen-in-tausend

Bestimmt die Maximalgröße des Outputs in 1000 Zeilen (maximal 6-stellig). Ist der Wert erreicht, übergibt JES2 die Kontrolle an einen definierten USER-Exit (der den Job beendet oder auch nicht).

COPIES=anz-ausgaben

Anzahl der Gesamtdruckausgaben (maximal 255, bzw. die entsprechend als Maximum hinterlegte Anzahl in den Spool-Ladeparametern). Die Angabe wird ignoriert, wenn eins der folgenden zutrifft:

- `FREE=CLOSE` ist in der Output-DD-Anweisung kodiert.
- `HOLD=YES` wurde in einem DD-Statement angegeben.
- Die Outputklasse ist eine HELD-Queue

LINECT=zeilen-je-seite

Anzahl Zeilen je Seite, bevor ein Seitenwechsel erfolgt. Wird `LINECT=0` kodiert, wird von JES kein Seitenwechsel durchgeführt. Wurde eine `LINECT`-Angabe in einem `OUTPUT`-Statement angegeben, hat diese Vorrang.

PROCLIB=ddname

Weist eine Datei, welche über das angegebene DD-Statement zugewiesen wurde, als Prozedurbibliothek aus. Üblicherweise werden diese DD-Namen mit `PROCxx` bezeichnet (`XX = 01 bis 99`). JES2 liest katalogisierte Prozeduren von diesem DD-Namen.

Hinweis

Das `JOBPARM`-Statement sollte immer unmittelbar hinter `JOB` liegen. Wenn `JOBPARM` vor einer `/*ROUTE=XEQ`-Anweisung liegt, wird diese vor Übergabe auf dem Sendesystem überprüft, andernfalls auf dem Zielsystem.

JUSTIFY() – REXX Built-in-Function

```
►►JUSTIFY(string,länge[,füllzeichen])◄◄
```

Formatiert Worte innerhalb eines Strings, die durch BLANK voneinander getrennt sind, in dem die aktuelle Stringlänge auf die angegebene Länge geändert wird. Standardfüllzeichen ist BLANK.

Beispiele

```
SAY JUSTIFY('Hans ist doof',15)      'Hans  ist  doof'
SAY JUSTIFY('Hans ist doof',9)       'Hans  ist'
SAY JUSTIFY('Hans ist doof',8)       'Hans ist'
SAY JUSTIFY('Hans ist doof',15,'-')  'Hans--ist--doof'
```

LABEL – ISPF Edit Macro

```
►►"(var1,var2) = LABEL zz"◄◄
►►"LABEL zz = labelname[ebene]"◄◄
```

Setzt ein Label, oder übergibt den Labelnamen der Zeile.

=LABEL zz

Labelnamen der Zeile und der LEVEL, in der das Label gesetzt wurde.

LABEL zz =

der entsprechenden Zeile wird ein Labelname zugewiesen.

labelname

Beginnt mit Punkt. Besteht aus Alphazeichen. Kein führendes Z. Soll ein Label gelöscht werden, wird es auf Blank gesetzt. Über LINENUM kann festgestellt werden, ob eine Zeile ein Label hat (siehe auch dort).

ebene

Höchste Ebene ist LEVEL 0 (Anwender). Level 1 wird dem ersten Macro zugewiesen. Ruft ein Macro ein anderes auf, wird LEVEL hochgezählt.

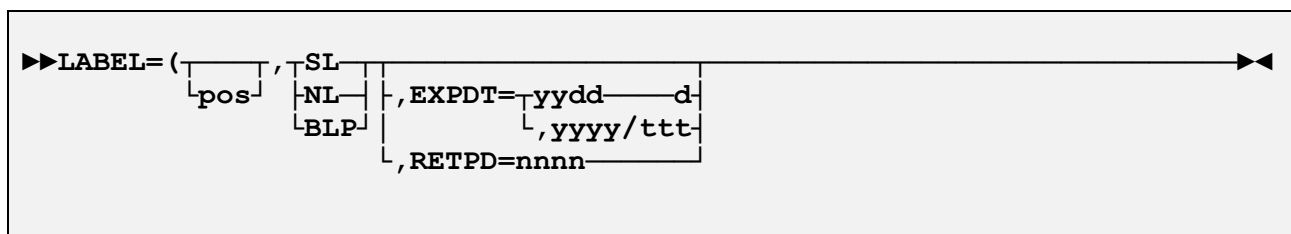
Beispiel

```
"find" suchvar  
if rc = 0 then "(lname)= label .zcsr"  
if lname = "" then "label .zcsr = .found 0"
```

War der FIND-Befehl erfolgreich, wird geprüft, ob die gefundene Zeile ein Label hat. Wenn nicht, wird das Label .FOUND auf Ebene 0 gesetzt.

Return Codes

- 0 Normales Ende
- 4 Kein Label ermittelt
- 8 Label gesetzt. Existentes Label auf gleicher Stufe gelöscht
- 12 Zeilennummer größer als .ZLAST
- 20 Schwerer Fehler

LABEL – JCL DD Parameter

Kennsatzbeschreibung auf einem Magnetband oder einer Kassette.

pos

Relative Position der Datei auf dem Datenträger. Sie wird durch entsprechende TapeMarks definiert. Bei BLP-Verarbeitung ist als Position 2 anzugeben (die erste logische Datei nach den Kennsätzen).

SL

Standard-Label. Die standardgemäßen Kennsätze VOLUME, HEADER1 und HEADER2 sind vorhanden und werden geprüft, oder werden geschrieben.

NL

No-Label. Es sind keine Kennsätze vorhanden. Kommt heute in der Regel nicht mehr vor und war früher oft in Verbindung mit Fremdbändern nötig.

BLP

Bypass-Label-Processing. Die Kennsatzverarbeitung wird umgangen.

EXPDT=yyddd

Das Verfalldatum (Freigabedatum - ExpirationDate).

RETPD=nnnn

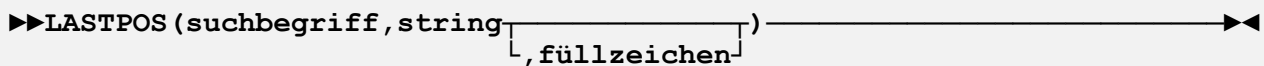
Eine Sicherungsfrist (RetentionPeriod) in Tagen wird angegeben. Das System errechnet daraus das Verfalldatum selbst.

Besonderheiten in Verbindung mit:

EXPDT=99000 Sperrfrist über Systemkatalog. Spezialität bei GDGs.

EXPDT=99365 Keine Freigabe (permanente Sicherung)

EXPDT=98000 Band liegt außerhalb des Verwaltungs-Kataloges. Kommt bei Fremdmaterial, oder erstmals zu initialisierenden Datenträgern vor.

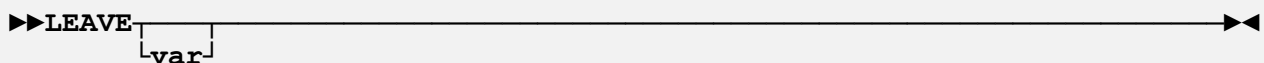
LASTPOS() – REXX Built-in-Function

```
>>LASTPOS(suchbegriff,string[,füllzeichen])
```

Von der Funktionsweise her identisch mit POS(), wobei der betreffende String von rechts nach links durchsucht wird (rückwärts).

Beispiele

```
SAY LASTPOS(' ','Hans ist doof')      9
SAY LASTPOS(' ','Hansistdoof')      0
SAY LASTPOS(' ','Hans ist doof',8)   5
```

LEAVE – REXX Instruction

```
>>LEAVE[var]
```

LEAVE führt zum unmittelbaren Verlassen einer Schleife. Die Verarbeitung innerhalb der Prozedur wird hinter dem zu DO gehörenden END fortgesetzt.

Wird eine Iteration durch LEAVE vorzeitig verlassen, enthält diese Steuervariable außerhalb der Schleife den Wert, der ihr zum Zeitpunkt des LEAVE zugeordnet war.

Beispiele

```
do $=1 to 5
    say $'. Ausgabezeile in der Schleife'
    if $=3 then leave
end
```

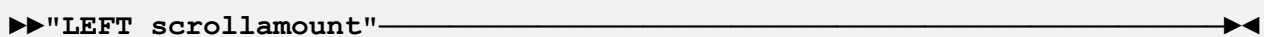
/ Zeigt folgendes am Terminal */*

1. Ausgabezeile in der Schleife
2. Ausgabezeile in der Schleife
3. Ausgabezeile in der Schleife

Kein MUSS, aber möglich:

```
do _1=1 to b
    do _2=1 to y
        do forever
            do while var1 < var2
                /*
                 Ereignis, das die Rückkehr in die
                 äußerste Schleife bewirken soll
                */
                if ??? then leave _2
            end
        end
    end
end
```

LEFT – ISPF Edit Macro



LEFT wird benutzt, um den Bildschirmausschnitt nach links zu verschieben.

scrollamount

MAX	Zum Datenende
HALF	um eine halbe Seite
PAGE	um eine ganze Seite
CURSOR	Die Cursor-Position steht ganz links
spalten	Anzahl Spalten nach links

Beispiel

```
"left max"
```

Zeigt den linken Rand der Datei.

Return Codes

0	Normales Ende
4	Keine sichtbaren Datenzeilen
8	Keine Daten anzuzeigen
12	Kein Scrollamount angegeben
20	Schwerer Fehler

LEFT() – REXX Built-in-Function

►►LEFT(string,länge) —————►◄
 füllzeichen

Übergibt eine Zeichenfolge in der angegebenen Länge. Situationsabhängig wird nach der angegebenen Anzahl Byte auf der rechten Seite abgeschnitten oder mit dem angegebenen Füllzeichen aufgefüllt (Standard ist BLANK).

Beispiele

```
Say LEFT('Hans ist doof',15)           'Hans ist doof  '
Say LEFT('Hans ist doof',15,'*') 'Hans ist doof**'
Say LEFT('Hans ist doof',4)           'Hans'
```

```
:
if datatype(left(var,1),"M")=0 then
    say 'Stelle 1 von' var 'ist kein Buchstabe'
:
```

Hinweise:

LEFT(string,5) ist gleichzusetzen mit SUBSTR(string,1,5).

LENGTH() – REXX Built-in-Function

►►LENGTH(string) —————►◄

Übergibt die Länge eines Strings in Anzahl Byte.

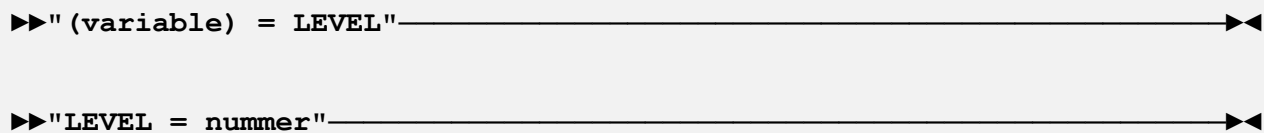
Beispiele

```
SAY LENGTH('Hans ist doof')           13
SAY LENGTH('abcdefghijklm')           13
```

```
SAY LENGTH(' ') 0
```

```
:  
say 'Dateinamen angeben (ENTER = Ende) '  
do forever  
    pull datei  
    if length(datei)=0 then exit  
:  
end  
:
```

LEVEL – Edit



```
»»"(variable) = LEVEL"_____<<<<  
»»"LEVEL = nummer"_____<<<<
```

Setzt oder liefert den Modifikationslevel einer Datei.

=LEVEL

Aktueller Modifikationslevel, zweistellig numerisch mit führenden Nullen.

LEVEL=

Der Modifikationslevel wird neu gesetzt (möglich 00 bis 99).

Beispiele

```
"level = 1"
```

Der Modifikationslevel wird auf 1 gesetzt.

Return Codes

0	Normales Ende
4	STATS-Mode ausgeschaltet, Befehl ignoriert
12	Falsche Wertangabe
20	Schwerer Fehler

LIBDEF – ISPF DM Instruction

```
►►"LIBDEF ddname DATASET ID(dateiliste) [COND | UNCOND | STACK | STKADD] "◄◄
```

Erlaubt die dynamische Zuweisung von Bibliotheken.

Da der LIBDEF-Service eine sehr große Anzahl unterschiedlichster Möglichkeiten bietet soll hier nur auf die in der Praxis am häufigsten angewandten eingegangen werden.

ddname

Die relevanten DD-Namen zeigt die folgende Tabelle:

DD-Name	Nutzung
ISPM LIB	Nachrichtenbibliothek
ISPP LIB	Panelbibliothek
ISPS LIB	File Tailoring Eingabe
ISPF FILE	File Tailoring Ausgabe
ISPT LIB	Tabelleneingabe
ISPTABL	Tabellenausgabe
ISPLLIB	Lademodule
ISPILIB	Image Bibliothek (benötigt GDDM)
xxxxUSR	Gleichbedeutend mit den oben genannten auf User-Ebene

DATASET

steuert, dass die unter ID in der Klammer stehende Angabe eine katalogisierte Datei sein muss.

Dateiliste

eine oder mehrere katalogisierte Dateien. Werden mehrere angegeben, sind die einzelnen Angaben durch Komma voneinander zu trennen.

COND

sagt aus, dass die Zuweisung nur erfolgt, wenn im aktuellen Dialog noch keine Zuweisung gleichen Namens durchgeführt ist.

UNCOND

Dynamische Zuweisung. Die angegebenen Dateien werden in der Suchreihenfolge vorne eingereiht.

STACK

Die Zuweisungen werden gestacked. Ein Reset der Libdef-Definitionen führt zur vorausgegangenen Zuweisung.

STKADD

Ergänzendes Stacking der Libdef-Definitionen.

xxxxUSR

Sollen die Zuweisungen über User-spezifische Namen erfolgen, setzt dies voraus, dass diese DD-Namen im Vorfeld per Allocate zugewiesen wurden.

Hinweis

Häufig sind Default-Parameter in der ISPF Configuration Table gesetzt, die heute mehr Ärger als Nutzen erzeugen. Am sinnvollsten ist die Angabe aller benötigten Parameter. Gegebenenfalls sind die Defaults bei der für das ISPF zuständigen Stelle in der Systemprogrammierung zu hinterfragen.

Return Codes

0	Normales Ende
4	Für den angegebenen Typ existiert keine Bibliothek
8	Zuweisung besteht bereits (in Verbindung mit COND)
12	Bei ISPPROF-Zuweisung falscher Librarytyp angegeben (in Verbindung mit EXCLDATA oder EXCLLIBR)
16	Kein Bibliotheksname angegeben oder ein Dateiname ist nicht zOS-konform
20	Schwerer Fehler

LIKE – JCL DD Parameter

```
▶▶LIKE=dateiname▶▶
```

Diese Angabe wird nur in Verbindung mit SMS berücksichtigt. Für SMS-gemanagte Dateien können auf diese Weise Informationen wie Satzformat, Dateiorganisation und SPACE-Angaben übernommen werden.

Achtung:

Werden Informationen per LIKE ermittelt, haben diese Vorrang gegenüber Angaben aus DATACLASS oder STORAGECLASS.

LINE – ISPF Edit Macro

```
▶▶"(variable) = LINE zz"▶▶
```

```
▶▶"LINE zz = daten"▶▶
```

Auslesen oder Überschreiben einer Datenzeile.

=LINE zz

Inhalt der Datenzeile wird in einer Variablen abgelegt.

LINE zz =

Die Zeile, auf die der Zeilenzeiger zeigt, wird inhaltlich überschrieben.

daten

- einfache oder eingegrenzte Zeichenketten, oder Variable
- Schablonen (< spalte,zeichenfolge >)
- Mischformate (string1 + string2,schlüsselwort + ...)
- Schlüsselworte
 - LINE [zz] Daten der aktuellen Zeile (oder Ziele zz)
 - MASKLINE Daten der Maskenzeile
 - TABSLINE Daten der TABS-Zeile

Beispiele

```
"(aktdata) = line .zcsr"
```

Die Zeile, auf die der Cursor zeigt, wird nach AKTDATA kopiert.

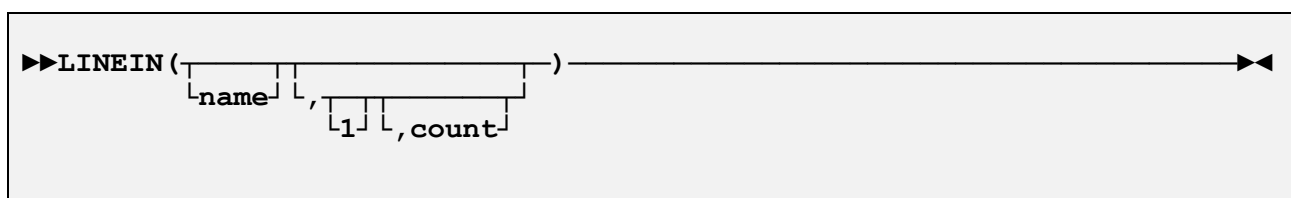
```
"line .zcsr = line + //"
```

Überlagert Spalte 1 und 2 der aktuellen Zeile mit //.

Return Codes

0	Normales Ende
4	Daten abgeschnitten, Zeile zu kurz
12	Falsche Zeilennummer
16	Variablendaten abgeschnitten
20	Schwerer Fehler

LINEIN() – UNIX Function



Liefert eine oder keine Zeile aus dem Input-Stream. Dabei fungiert das Steuerzeichen ESC_N (NewLine) als Delimiter.

Name

Der Name eines Streams. Fehlt die Angabe, wird STDIN angenommen.

1

Sofern gewollt, kann damit der erste Datensatz gelesen werden. Diese Angabe ist ausschließlich für persistente Streams erlaubt.

Cont

Kann 0 oder 1 enthalten. Bei 0 werden keine Daten gelesen und die Funktion gibt einen Nullstring zurück und setzt, sofern als 2. Parameter eine 1 gesetzt ist, den Lesepointer auf Byte 1 des Streams zurück.

Bei Angabe 1 wird 1 Datensatz gelesen, sofern es noch etwas zu lesen gibt. Ist dies nicht der Fall, wird ein entsprechender Return Code zurückgegeben.

Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Es gibt eine Reihe von Fluchtsequenzen (ESC_?). ESC_N erzeugt eine neue Zeile. Siehe hierzu auch das IBM Manual "Using REXX and Unix System Services"

Beispiel

```
/* REXX
*/
File = 'unix_file_name'
Satz = linein(file)
Say "Inhalt des gelesenen Satzes:" Satz
```

LINENUM – ISPF Edit Macro

```
>>"variable = LINENUM label"<<
```

Ermittelt die relative Zeilennummer einer Datenzeile. Das Ergebnis ist nicht zuverlässig, wenn Zeilen eingefügt oder gelöscht wurden, die noch keine Zeilennummer vom Edit erhalten haben.

variable

Enthält die relative Zeilennummer

label

Labelname, von dessen Zeile die Zeilennummer benötigt wird.

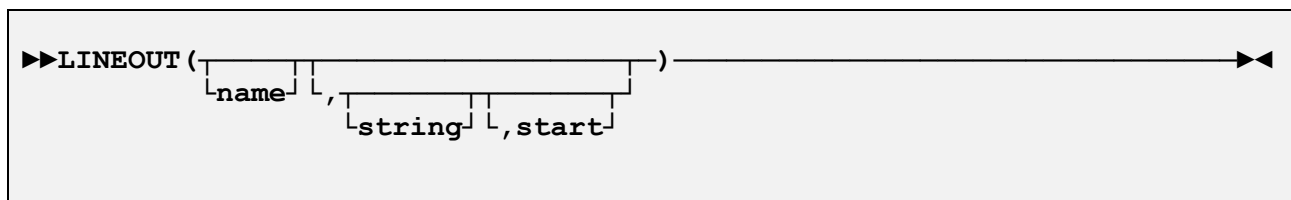
Beispiel

```
"(lastline) linenum .zlast"
```

Die relative Zeilennummer der letzten Zeile wird ermittelt. Sind keine Zeilen vorhanden, enthält die Variable den Wert 0.

Return Codes

0	Normales Ende
4	Zeile 0 angegeben
8	Label nicht angegeben, Zeile enthält 0
12	Falsche Zeilennummer
20	Schwerer Fehler

LINEOUT() – UNIX Function

Schreibt eine Zeile in den Datenstrom. Ein NewLine-Steuerzeichen (ESC_N) wird an die Daten angehängt.

Name

Der Name eines Streams. Fehlt die Angabe, wird nach STDOUT geschrieben.

string

Daten, die geschrieben werden sollen. Sind keine Daten definiert, erfolgt kein Schreiben. Der Satz-Posotionszeiger wird auf den Wert von start gesetzt und als RC wird 0 zurückgegeben Ist start auch nicht angegeben, wird der Positionszeiger auf den Anfang des Files gesetzt.

Start

Nur im Falle persistenter Datenströme könnte hier eine 1 für den ersten zu schreibenden Satz angegeben werden.

Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Es gibt eine Reihe von Fluchtsequenzen (ESC_?). ESC_N erzeugt eine neue Zeile. Siehe hierzu auch das IBM Manual "Using REXX and Unix System Services"

Beispiel

```
/* REXX */  
call lineout , 'I am the very best'
```

LINES() – UNIX Function

►►LINES (name) ◀◀

Liefert eine 1, wenn noch Sätze zur Verarbeitung vorhanden sind. Anderenfalls eine 0.

Name

Der Name eines Streams. Fehlt die Angabe, wird STDIN angenommen.

Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Beispiel

```
/* REXX
*/
File = 'unix_file_name'
do forever
  if ^lines(file) then leave
  data = linein(file)
  if pos(var,data) > 0 then say "Treffer in" data
end
Say "alle Sätze verarbeitet"
exit
```

LINESIZE() – NonSAA-Built-in-Function

►►LINESIZE () ◀◀

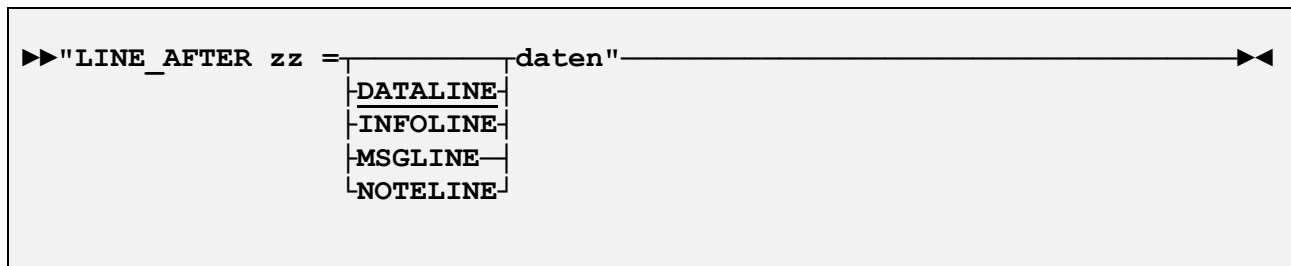
Übergibt in einem Batch-Adressraum immer den Wert 131.

Unter TSO Foreground wird der Wert der Bildschirmbreite – 1 ausgegeben.

Hinweis

Oftmals sinnvoller ist die Nutzung der ISPF-Systemvariablen ZSCREENW.

LINE_AFTER – ISPF Edit Macro



Fügt hinter der angegebenen Zeile eine neue Zeile in die Datei ein. Der Befehl wird im Macro genutzt, wenn eine neue Zeile mit bereits vorbereiteten Daten in die Datei eingefügt werden soll. INSERT erlaubt dem Anwender das Füllen der Zeile.

zz

Zeilenzeiger (Label oder relative Zeilennummer). Hinter dieser Zeile wird die neue Zeile eingefügt.

DATALINE

Die Zeile ist eine Datenzeile.

INFOLINE

Temporäre Nachrichtenzeile. Diese führt im Zeilennummernfeld =====. Wie Hinweiszeilen haben Infozeilen unabhängig von der Datensatzlänge eine Länge von 72 Byte. Auch auf Nachrichtenzeilen kann im weiteren Verlauf des Makros nicht Bezug genommen werden.

MSGLINE

Temporäre Nachrichtenzeile (führt im Zeilennummernfeld ==MSG>). Wie Hinweiszeilen haben Nachrichtenzeilen unabhängig von der Datensatzlänge eine Länge von 72 Byte. Auch auf Nachrichtenzeilen kann im weiteren Verlauf des Makros nicht Bezug genommen werden.

NOTELINE

Es handelt sich um eine temporäre Hinweiszeile. Im Zeilenbefehlsfeld der Zeile wird =NOTE= intensiv dargestellt eingetragen, die Zeile selbst wird normal dargestellt. Hinweiszeilen haben, unabhängig von der Satzlänge der Datei, eine Datenlänge von 72 Byte. Auf Hinweiszeilen kann im Makros nicht Bezug genommen werden.

daten

Für daten können unterschiedliche Angaben gemacht werden:

- einfache oder eingegrenzte Zeichenketten, oder Variable
- Schablonen (< spalte,zeichenfolge >)
- Mischformate (string1 + string2,schlüsselwort + ...)
- Schlüsselworte
 - LINE Daten der aktuellen Zeile
 - LINE zz Daten der Zeile zz
 - MASKLINE Daten der Maskenzeile
 - TABSLINE Daten der TABS-Zeile

Beispiele

```
"line_after 0 = 'Neue Zeile1'"
```

Das Einfügen hinter die relative Zeile 0 erzeugt eine neue Zeile1.

```
"line_after .zlast = .zcsr"
```

Die Zeile, in der der Cursor steht, wird an das Ende der Datei kopiert.

Return Codes

0	Normales Ende
4	Daten abgeschnitten
12	Falsche Zeilennummer
20	Schwerer Fehler

LINE_BEFORE – ISPF Edit Macro



Fügt vor der angegebenen Zeile eine neue Zeile in die Datei ein. Der Befehl wird im Macro genutzt, wenn eine neue Zeile mit bereits vorbereiteten Daten in die Datei eingefügt werden soll. INSERT erlaubt dem Anwender das Füllen der Zeile.

zz

Zeilenzeiger (Label oder relative Zeilennummer). Hinter dieser Zeile wird die neue Zeile eingefügt.

DATALINE

Die Zeile ist eine Datenzeile.

INFOLINE

Temporäre Nachrichtenzeile. Diese führen im Zeilennummernfeld ===== in intensiver Darstellung. Der Infotext wird normal dargestellt. Wie Hinweiszeilen haben Infozeilen unabhängig von der Datensatzlänge eine Länge von 72 Byte. Auch auf Nachrichtenzeilen kann im weiteren Verlauf des Makros nicht Bezug genommen werden.

MSGLINE

Temporäre Nachrichtenzeile. Diese führen im Zeilennummernfeld ==MSG> in intensiver Darstellung. Der Nachrichtentext wird normal dargestellt. Wie Hinweiszeilen haben Nachrichtenzeilen unabhängig von der Datensatzlänge eine Länge von 72 Byte. Auch auf Nachrichtenzeilen kann im weiteren Verlauf des Makros nicht Bezug genommen werden.

NOTELINE

Temporäre Hinweiszeile. Im Zeilenbefehlsfeld der Zeile wird =NOTE= eingetragen. Hinweiszeilen haben, unabhängig von der Satzlänge der Datei, eine Datenlänge von 72 Byte. Auf Hinweiszeilen kann im Makros nicht Bezug genommen werden.

daten

Für `daten` können unterschiedliche Angaben gemacht werden:

- einfache oder eingegrenzte Zeichenketten, oder Variable
- Schablonen (< spalte,zeichenfolge >)
- Mischformate (string1 + string2,schlüsselwort + ...)
- Schlüsselworte
 - `LINE` Daten der aktuellen Zeile
 - `LINE zz` Daten der Zeile `zz`
 - `MASKLINE` Daten der Maskenzeile
 - `TABSLINE` Daten der TABS-Zeile

Beispiele

```
"line_before 1 = 'Neue Zeile 1'"
```

Das Einfügen hinter die relative Zeile 0 erzeugt eine neue Zeile1.

```
"line_before .zcsr = line .start"
```

```
"delete .start"
```

Die Zeile mit dem Label `.START` wird vor die Zeile in der der Cursor steht übertragen.

Return Codes

0	Normales Ende
4	Daten abgeschnitten
12	Falsche Zeilennummer
20	Schwerer Fehler

LINE_STATUS – ISPF Edit Macro

```
►►" (variable) = LINE_STATUS_zeilennummer"◄◄  
                        label
```

`LINE_STATUS` liefert die Source/Change-Informationen einer Zeile

variable

enthält einen String von 32 Byte Länge, dessen einzelne Stellen jeweils 1 oder 0 sind. Die Spaltenpositionen haben folgende Bedeutung:

Spalte 1 bis 7 sind Source-Informationen	
Spalte	Bedeutung, wenn 1
1	Originalzeile. Existierte zu Beginn der Edit Session.
2	Zeile wurde durch MOVE-Zeilenkommando erstellt
3	Erstellt durch COPY- oder REPEAT-Zeilenkommando
4	Erstellt durch MOVE Primary Command oder Macro
5	Erstellt durch MOVE Primary Command oder Macro
6	Erstellt durch TE Line Command
7	Erstellt durch INSERT Line Command
Spalte 8 bis 14 sind Change-Informationen	
8	Zeile wurde geändert
9	Zeile wurde überschrieben
10	Änderung durch CHANGE Primary Command oder OVERLAY Line Command
11	Änderung durch Column Shifting
12	Änderung durch Data Shifting
13	Änderung durch TE, TF oder TS Line Command
14	Zeile wurde neu nummeriert
15-32	Für zukünftige Benutzung

zeilennummer

Nummer der Zeile, deren LINE_STATUS-Informationen gewünscht werden.

label

Label, das die betreffende Zeile identifiziert

Beispiel

```
"(linestat)" LINE_STATUS" lnum
if linestat(1) = 1 then say "Zeile wurde geändert"
else say "Zeile ist noch unverändert"
```

Return Codes

- 0 Normales Ende
- 12 Zeilennummer falsch
- 20 Schwerer Fehler

LISTALC – TSO Command

```
►►"LISTA[STATUS][HISTORY][MEMBERS][SYSNAMES]"◄◄
```

Listet alle zugeordneten Dateien auf. Wird das Kommando im Verlauf einer TSO-Sitzung mehrmals eingegeben, wird sich der Umfang der Meldungsausgabe verändern.

Die Zusätze ST(atus) und H(istory) M(embers) und SYSNAMES liefern erweiterte Informationen.

STATUS | ST

DD-Name und Disposition

HISTORY | H

CREATION-/EXPIRATION Date und die Dateiorganisationsform.

MEMBER | M

Zeigt bei einer PO-Datei die Membernamen, welche als Update-Eintrag die HLQ führen (ist die Datei anderweitig im Zugriff führt dies zum Fehler).

SYSNAMES

Liefert eine Liste aller zugewiesener Dateien (auch temporärer).

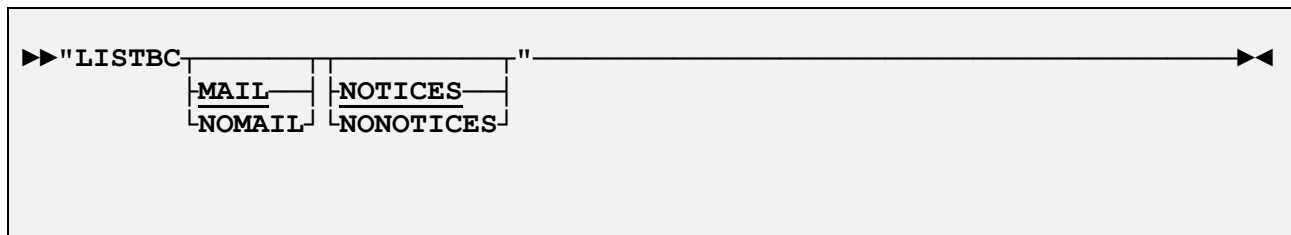
Beispiel LISTA ST

```
:
XWITT.LIB.COB
    GENCOB    KEEP
SE01.LIB.DUMMY
    SYSPROC   KEEP
XWITT.LIB.CLIST
    KEEP
SE01.TOOL.CLIST
    KEEP
:
```

LIST – ISPF DM Service

Sei nur der Vollständigkeit wegen erwähnt. Der LIST-Service schreibt einen Eintrag in das ISPF-List-Dataset., das heute in der Regel niemand mehr benutzt (aber wer weiß...).

LISTBC – TSO Command



Zeigt die Abschnitte NOTICES und/oder MAIL aus der SYS1.BROADCAST-Datei für den jeweiligen User an

MAIL | NOMAIL

Anzeige oder nicht aus dem Bereich MAIL.

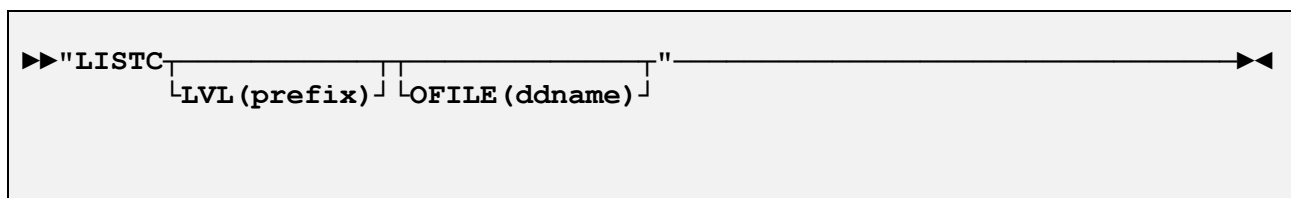
NOTICES | NONOTICES

Anzeige oder nicht aus dem Bereich NOTICES.

Hinweis

Kommt heute in aller Regel nicht mehr häufig vor. Mittlerweile hat man die Vorzüge von Anwendungen wie Notes oder Outlook entdeckt 😊

LISTCAT – TSO Command



Gibt eine Liste aller katalogisierten Dateien der eigenen Benutzerkennung bzw. mit dem angegebenen Prefix aus.

Insgesamt kann sehr vieles mit dem LISTCAT-Befehl ermittelt werden, was in der täglichen Praxis aber weniger häufig vorkommt. Im Bedarfsfall gibt das Manual TSO Command Reference Auskunft.

Prefix

Prefixangabe, welche bei der Ermittlung der Katalogeinträge berücksichtigt wird.

OFFILE(ddname)

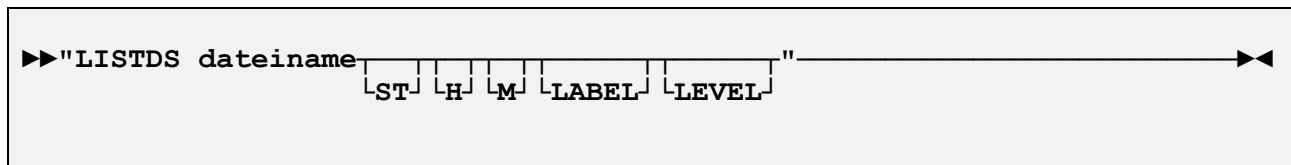
Ist zur Laufzeit der DD-Name allokiert, erfolgt die Befehlsausgabe dorthin.

Beispiel LISTC LVL(SP1)

```
:  
NONVSAM —— SP1.AUSB.CNTL  
      IN -CAT — USER01.UCAT  
NONVSAM —— SP1.BACKUP.BANKLIST  
      IN CAT — USER01.UCAT  
NONVASM —— SP1.LIB.CNTL  
      IN CAT — USER01.UCAT  
NONVSAM —— SP1.MVS.CLIST  
      IN CAT — USER01.UCAT  
:
```

Return Codes

- 0 Normales Ende
- 4 Verarbeitung OK aber ein Warnhinweis wurde ausgegeben
- 8 Grundsätzlich OK, aber Details wurden ausgelassen
- 12 Keine Ausführung
- 20 Schwerer Fehler

LISTDS – TSO-Kommando

```
>>>'LISTDS dateiname [ST] [H] [M] [LABEL] [LEVEL]'
```

Gibt eine Liste der Kenngrößen der angegebenen Datei aus. Wenn ein Dateiname inklusive Membername angegeben wird, werden Informationen über das Member aus dem Directory ausgegeben.

STATUS | ST

Liefert den Status einer Datei

HISTORY | H

Informationen über Erstellung-/Freigabedatum und RACF-Schutz.

MEMBER | M

Zeigt im Falle einer PO-Datei auch die Membernamen an.

LABEL

Der gesamte DSCB wird angezeigt

LEVEL

Bestimmt, dass die Dateiangabe nur ein HLQ ist und alle Dateien, welche darunter im Katalog gefunden werden, sind betroffen.

Beispiel LISTDS PDF.DATA STATUS

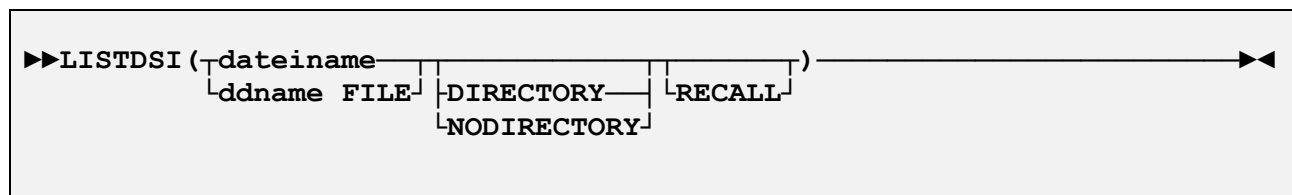
```
WITTEMA.PDF.DATA132
-RECFM-LRECL-BLKSIZE-DSORG-DDNAME-DISP
  FB      132      15444    PO      SYS00043 KEEP
-VOLUMES-
  USER01
```

Beispiel LISTDS PDF.DATA STATUS MEMBER

```
WITTEMA.PDF.DATA132
-RECFM-LRECL-BLKSIZE-DSORG-DDNAME-DISP
  FB      132      15444    PO      SYS00043 KEEP
-VOLUMES-
  USER01
-MEMBERS-
ADR1
ADR2
:
```

Return Codes

- 0 Normales Ende
- 12 Fehler, Meldung wurde ausgegeben.

LISTDSI() – TSO extended Function

Füllt eine Vielzahl von Variablen über die Datei. Generell können mit LISTDSI nur Informationen von Dateien ermittelt werden, die auf einer Platte (DASD) liegen. Wird die LISTDSI-Funktion für eine VSAM-Datei genutzt, werden nur Plattenname, Gerätegruppe und Dateiorganisationsform ausgegeben. Alle anderen Variablen werden mit einem Fragezeichen gefüllt.

DIRECTORY

führt bei einer PO-Datei zur Übergabe der Directory.

RECALL

veranlasst das Rückspeichern einer migrierten Datei, um an die Dateiinformatoren zu gelangen. Ein Rückholen von TAPE wird nicht veranlasst.

Folgende Variable werden durch das LISTDSI gefüllt (unvollständig):

Variable	Inhalt
SYSDSNAME	Dateiname vollqualifiziert

SYSVOLUME	Name des Datenträgers
SYSDSORG	Datei-Organisationsformat
SYSRECFM	Satzformat
SYSLRECL	Satzlänge
SYSBLKSIZE	Blockgröße
SYSALLOC	Zugeordneter Bereich in SYSUNITS
SYSUSED	Benutzter Bereich in SYSUNITS
SYSPRIMARY	Primärzuweisungsgröße
SYSSECONDS	Sekundärzuweisungsgröße
SYSUNITS	Größenangabe (Blöcke, Spuren, Cylinder)
SYSEXTENTS	Anzahl der Extents
SYSCREATE	Erstellungsdatum im Format jjjj/ttt
SYSREFDATE	Datum des letzten Zugriffs
SYSEXDATE	Verfalldatum
SYSRACFA	RACF-Informationen
SYSADIRBLK	Zugewiesene Directory-Blocks
SYSUDIRBLK	Benutzte Directory-Blocks
SYSMEMBERS	Anzahl der Member
SYSREASON	Return Code des LISTDSI
SYSMSGVL1	First Level Message bei Fehlern
SYSMSGVL2	Second Level Message bei Fehlern
SYSMAXGENS	Anzahl Generations für Type-2-Libraries
SYSDSVERSION	In Verbindung mit PO/E-Dateien: Type 1 oder Type 2

Hinweis

In Verbindung mit SMS (Storage Management Subsystem) werden eine Vielzahl weiterer Variabler gefüllt. Details hierzu können im REXX Reference Manual nachgelesen werden.

Neben dem RC von LISTDSI kann auch der Reason Code von Bedeutung sein. Ein Reason Code 5 hieße beispielsweise "Dataset not cataloged"

Return Codes

- 0 Normales Ende
- 4 Einige Informationen nicht verfügbar
- 16 Schwerer Fehler. Keine Informationen

LMCLOSE – ISPF LM Service

```
►►"LMCLOSE DATAID (did) "-----◄◄
```

LMCLOSE schließt die Datei, die mit der angegebenen Daten-ID verbunden ist.

DATAID(did)

Liefert die Daten-ID der zu schließenden Datei.

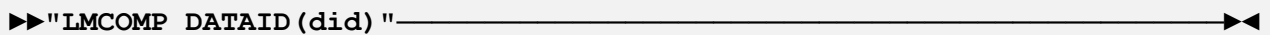
Beispiel

```
address ispxexec "lmclose dataid(&did) "
```

Die Datei, die über die Daten-ID DID geöffnet worden war, wird geschlossen

Return Codes

0	Normales Ende
8	Datei war nicht geöffnet
10	Daten-ID unbekannt
20	Schwerer Fehler

LMCOMP – ISPF LM Service

Mit LMCOMP kann ein Compress auf ein PO-Dataset durchgeführt werden. Bei Installation wird ein EXIT angeboten. Wird dieser nicht benutzt, wird IEBCOPY aufgerufen.

DATAID(did)

Liefert die Daten-ID der zu schließenden Datei.

Beispiel

```
address ispxexec "lmcomp dataid(&did) "
```

Die Datei, die über die Daten-ID DID zugewiesen ist, wird komprimiert.

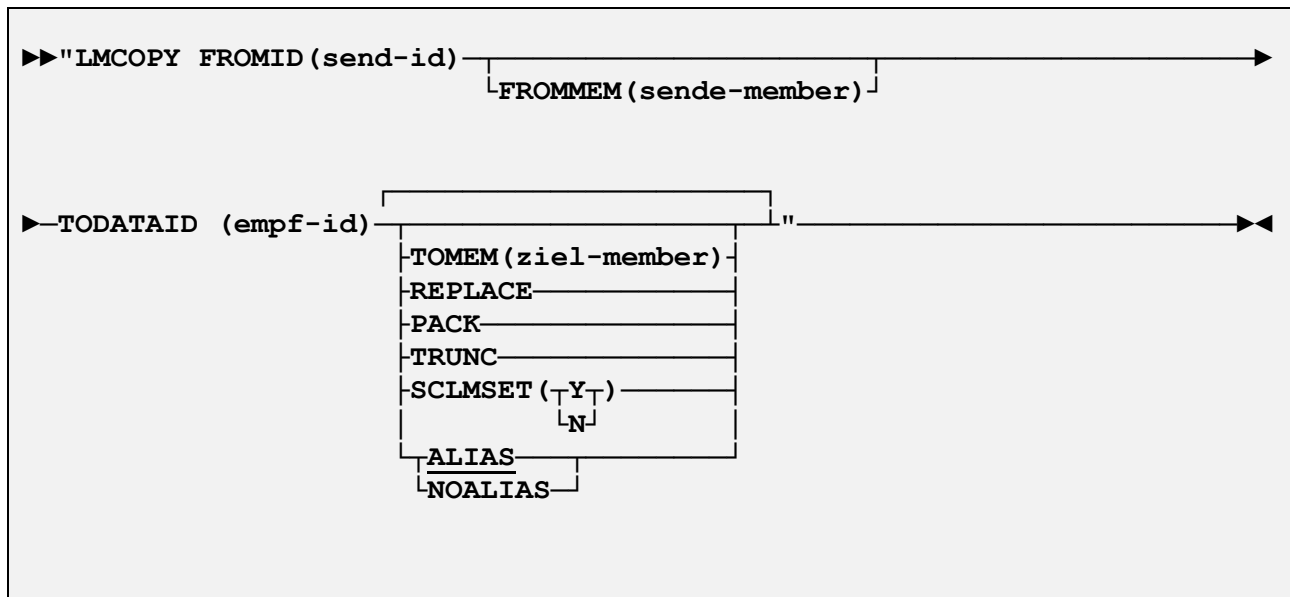
Hinweis

Heute ist der Umgang mit dem alten PO-Format fragwürdig. PO/E erzeugt keine Leichen, muss dementsprechend auch nicht compressed werden. PO/E-Type2 erlaubt Generationsführung von Membern und macht ein "UNDELETE" überflüssig. Um damit vernünftig arbeiten zu können, braucht ihr ein Tool, welches mit der Generationsführung umgehen kann. Das meiner Meinung nach derzeit Beste ist PDSEGEN von Lionel Dyck, kostenlos als Sourcecode unter www.cbttape.org erhältlich.

Return Codes

0	Normales Ende
8	Dateityp ist PDS/E (kann nicht compressed werden)
12	Einer der folgenden Gründe: Datei nicht partitioned Datei nicht zugewiesen Datei bereits geöffnet Datei nicht movable Datei muss mit LMINIT exklusiv geöffnet werden Concatenated Datasets nicht erlaubt
20	Schwerer Fehler

LMCOPY – ISPF LM Service



LMCOPY kopiert ein einzelnes Member aus einer PO-Datei oder eine sequentielle Datei.

Nur Satzformate F und V können gepackt werden.

FROMID und TODATAID können auf das gleiche Dataset zeigen, aber nicht die gleiche ID haben.

Dateien mit dem Attribut UNMOVEABLE können nicht kopiert werden.

FROMID(send-id)

DatenID der Sendedatei

FROMMEM(sende-member)

Benennt das zu kopierende Member. Generische Angaben sind möglich.

TODATAID(empf-id)

DatenID der Zieldatei

TOMEM(ziel-member)

Benennt das Zielfmember

REPLACE

Gleichnamig in der Zieldatei vorhandene Member werden überschrieben

PACK

Wenn angegeben, werden die Daten in der Zieldatei im gepackten Format gespeichert.

TRUNC

Abschneiden der Datensätze, wenn die Satzlänge des Senders größer als des Empfängers ist.

SCLMSET(Y|N)

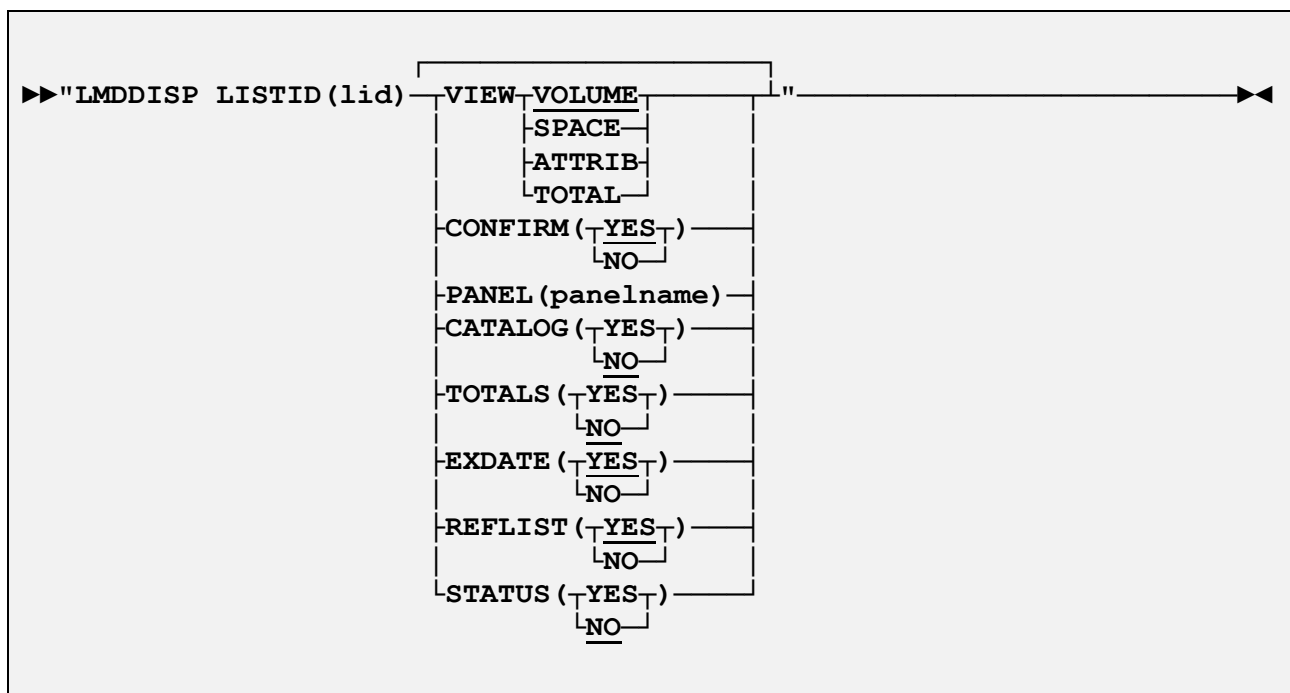
ISPF pflegt ein Bit im Directory, wenn SCLM eingesetzt wird.

ALIAS | NOALIAS

Bestimmt, ob Alias-Einträge mitkopiert werden, oder nicht.

Return Codes

- 0 Normales Ende
- 4 Member nicht verfügbar. Entweder ist die Sendedatei leer oder kein Member erfüllt die FROM-Angabe
- 8 FROMMEM nicht gefunden oder FROMMEM und TOMEM sowie FROMID und TOID sind gleich
- 10 Es gibt keine Dateiliste passend zur DataID
- 12 Einer der folgenden Gründe:
Member im Ziel vorhanden und REPLACE nicht angegeben
Ein oder mehrere Member in Zieldatei in USE
Falsche DS-Org.
Dateiattribute falsch
OPEN-Fehler
- 16 Truncation Fehler
- 20 Schwerer Fehler

LMDDISP – ISPF LM Service

LMDDISP erlaubt das Schreiben eines eigenen DSLIST-Dialogs vergleichbar zu 3.4 in PDF.

LISTID(lid)

Ist die Listidentifikationsnummer (LID), welche über dem LMDINIT-Service bereitgestellt wurde.

VIEW(VOLUME|SPACE|ATTRIB|TOTAL)

Steuert die initiale Anzeige der Zusatzinformationen zu den Dateinamen. Defaultwert ist VOLUME.

Wie in 3.4 kann zwischen den verschiedenen Anzeigeformen mit der Taste <PF-11> gewechselt werden

CONFIRM(YES|NO)

Steuert, ob das Confirm-Delete-Panel beim Löschen von Dateien angezeigt werden soll

PANEL(panelname)

Name des Panels, mit dem die Anzeige erfolgen soll.

CATALOG(YES|NO)

Steuert, ob der Catalog-Name in der VIEW-Anzeige erscheinen soll

TOTALS(YES|NO)

Beeinflusst die VIEW mit TOTALS: Anzeige Gesamtspuren aller Dateien, aller non-excluded Dateien, Gesamtanzahl der Dateien und Anzahl der non-excluded Dateien können in einer separaten Titelzeile ausgegeben werden.

Die Defaults für SPACE und TOTALS sind auf NO gesetzt, da dies deutlich performanter ist. Ist STATUS(ON) gesetzt, wird der Fortschritt der Datenermittlung von SPACE und TOTALS in einem PopUp-Panel angezeigt.

EXDATE(YES|NO)

Steuert, ob bei der TOTEL-Ansicht, das Freigabedatum ebenfalls angezeigt werden soll.

REFLIST(YES|NO)

Bestimmt, ob der unter der angegebenen ListID eingetragene DS-Name Level in die ISPF Reference List eingetragen werden soll.

STATUS(YES|NO)

Steuert, ob der Fortschritt der Datenermittlung für SPACE und TOTALS in einem PopUp-Panel angezeigt wird.

Return Codes

0	Normales Ende
8	Fehler beim Aufbau der Dateiliste. Der Grund ist in den ISPF-Systemvariablen ersichtlich
10	Keine zur ListID passende Dateiliste gefunden

- 12 Falsche Parameter
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMDFREE – ISPF LM Service

```
►►"LMDFREE LISTID(lid)"_____◄◄
```

Freigabe der ListID aus LMDINIT

LISTID(lid)

Name der dynamisch zugewiesenen ListID

Return Codes

- 0 Normales Ende
- 8 Fehler bei Freigabe der ListID (siehe ISPF System Dialogvariablen)
- 10 ListID nicht vorhanden (meist hat der LMDINIT nicht funktioniert)
- 20 Schwerer Fehler

LMDINIT – ISPF LM Service

```
►►"LMDINIT LISTID(lid) [LEVEL(ds-level)] [VOLUME(volume)]"_____◄◄
```

LMDINIT generiert eine ListID, für den Aufbau einer Dateiliste. Diese ID benötigen die Services LMDDISP und LMDLIST.

LISTID(lid)

Ist die Listidentifikationsnummer (LID), welche über dem LMDINIT-Service bereitgestellt wurde.

LEVEL(ds-level)

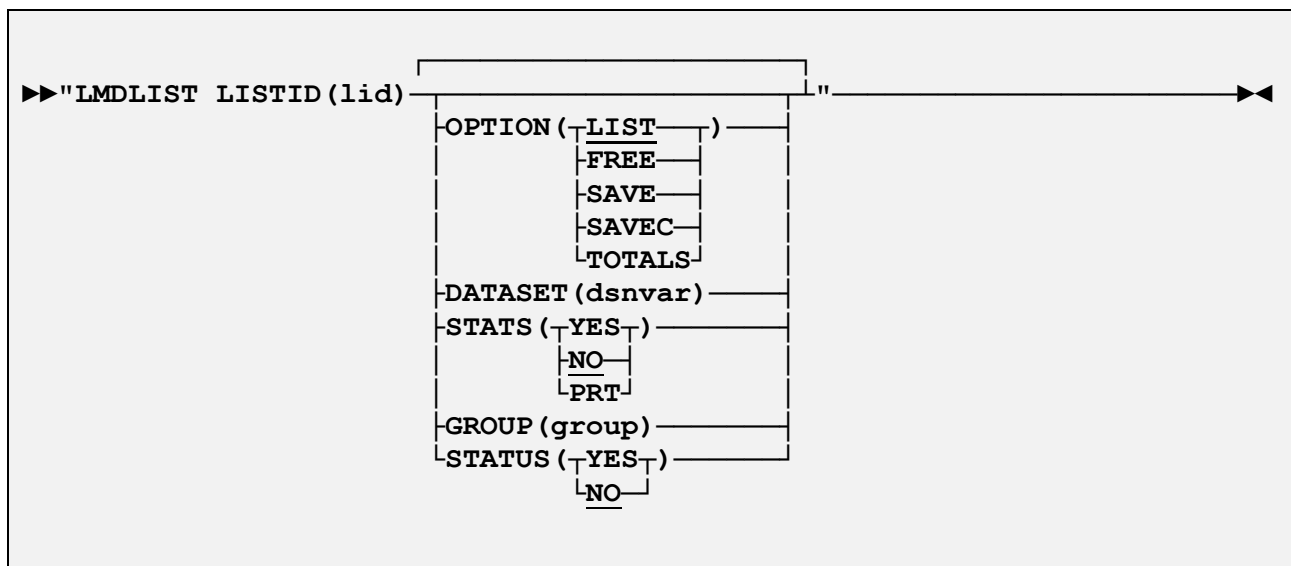
Ein oder mehrere Levels von Dateinamen, die angezeigt werden sollen. Wenn angegeben, ist dies ein String von Dateinamen (nach TSO-Konventionen, teil- oder vollqualifiziert)

VOLUME(volume)

Name des Volumes, dessen VTOC für die Generierung der Dateiliste benutzt werden soll.

Return Codes

- 0 Normales Ende
- 8 Fehler beim Aufbau der Dateiliste (siehe ISPF System Dialog Variablen)
- 12 Falsche Parameter
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMDLIST – ISPF LM Service

Generiert eine Liste von Dateien und erlaubt die Angabe der in ISPF üblichen Wildcards.

LISTID(lid)

Ist die Listidentifikationsnummer (LID), welche über dem LMDINIT-Service bereitgestellt wurde.

OPTION(LIST|FREE|SAVE|SAVEC)

Steuert, wohin der Service seine Informationen liefert.

LIST

Aufbau einer Namensliste, welche im Programm abgearbeitet werden kann.

FREE

Gibt die Liste frei. Wird der Service mehrfach hintereinander mit LIST gerufen, kann es erforderlich sein, dass dazwischen eine Freigabe erfolgt.

SAVE

Schreibt die Liste von Datasets in eine Datei. Die SAVE-Option kann nicht im Anschluss an eine LIST-Option benutzt werden, ohne dazwischen die Option FREE aufzurufen.

SAVEC

Wie SAVE, jedoch wird der Catalogname für jede Datei mit ausgegeben.

TOTALS

es werden zwei zusätzliche Variablen gefüllt: ZDLSIZET: Gesamtspuren aller Dateien und ZDL DST: Gesamtanzahl der Dateien in der Liste

DATASET(dsnvar)

Ist diese Variable gefüllt, beginnt die Liste der ermittelten Dateien erst bei der angegebenen. Wird diese Datei nicht gefunden, wird die nächste ausgegeben.

STATS(YES|NO|PRT)

Wird für die Optionen LIST und SAVE berücksichtigt. Wird PRT zusammen mit SAVE oder SAVEC angegeben, werden zusätzliche Informationen in der Headerzeile ausgegeben.

STATS(YES) liefert nachfolgende Variablen mit entsprechenden Werten:

Variable	Inhalt
ZDLVOL	Volume Serial des Datenträgers auf dem die Datei liegt
ZDLDEV	Device Type
ZDLDSORG	Dateiorganisation
ZDLRECFM	Satzformat
ZDLLRECL	Satzlänge
ZDLBLKSZ	Blockgröße
ZDLSIZE	Dateigröße in Spuren
ZDLSIZEEX	Dateigröße in Spuren (12 Byte langes Format)
ZDLUSED	Prozentuale Belegung der Spuren (oder Pages bei PDSE)
ZDLEXT	Anzahl benutzter Extents
ZDLEXTX	Anzahl benutzter Extents (5 Byte langes Format)
ZDLCDATE	Creation Date
ZDLEDATE	Expiration Date
ZDLRDATE	Date Last Referenced
ZDLMIGR	Information, ob die Datei migriert ist (YES oder NO)
ZDLDSNTP	Dateityp: PDS, LIBRARY oder Blank
ZDLSPACU	Space Units
ZDLMVOL	Multi Volume Dataset (Y oder N)
ZDLCATNM	Katalogname, in dem die Datei steht
ZDLOVF	Space Overflow Indicator (YES oder NO)

GROUP(group)

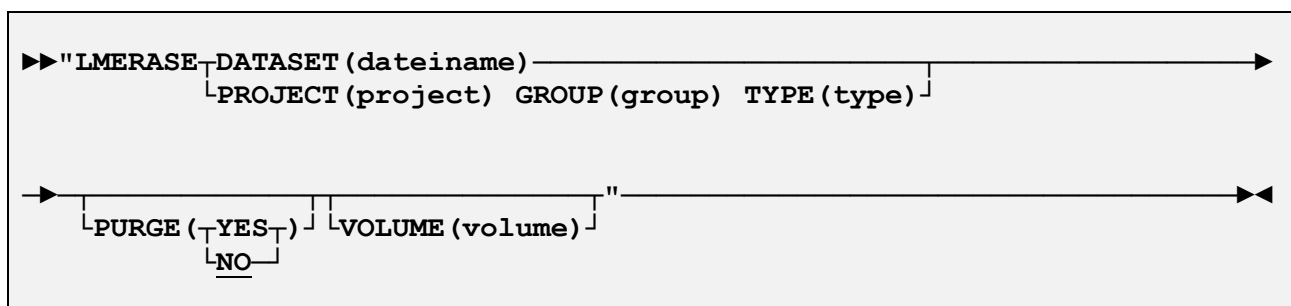
Teilqualifier der Ausgabedatei, in welche der Service die Dateiliste abspeichert. Der vollständige Name ist UserID.group.DATASETS.

STATUS(YES|NO)

Steuert, ob der Fortschritt der Datenermittlung für SPACE und TOTALS in einem PopUp-Panel angezeigt wird.

Return Codes

- 0 Normales Ende aus einem der folgenden Gründe
 - LIST: Der nächste Dateiname wurde übergeben
 - FREE: Der Speicherplatz wurde freigegeben
 - SAVE: Dateiliste erfolgreich in Datei geschrieben
- 4 Keine Daten gefunden
- 8 Ende der Dateiliste
- 10 Es existiert keine Dateiliste passend zur ListID
- 12 Falsche Parameter
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMERASE – ISPF LM Service

Löscht eine katalogisierte Datei.

DATASET(dateiname)

Name der zu löschenden Datei.

PROJECT(project) GROUP(group) TYPE(type)

First, Second und Third Qualifier der Datei.

PURGE(YES | NO)

Im Falle Yes wird auch gelöscht, wenn das Freigabedatum noch nicht erreicht ist.

VOLUME(volume)

Muss angegeben werden, wenn die Datei nicht katalogisiert ist.

Hinweis

Im Gegensatz zum TSO-Delete LMERASE deutlich bessere Meldungen, wenn es nicht funktioniert.

Return Codes

- 0 Normales Ende
- 8 Einer der folgenden Gründe:
 - Datei nicht katalogisiert

	Fehler beim Löschen
	Freigabedatum noch nicht erreicht
20	Schwerer Fehler

LMFREE – ISPF LM Service

```
►►"LMFREE DATAID (did) "_____◄◄
```

LMFREE gibt eine zugewiesene Daten-ID wieder frei.

DATAID(did)

Liefert den Namen der freizugebenden Daten-ID.

Beispiel

Unter REXX kann Variablensubstitution vom Interpreter oder Dialog Manager übernommen werden:

```
address ispxec "lmfree dataid("did")"
address ispxec "lmfree dataid(&did)"
```

In beiden Fällen wird die in der Variablen DID enthaltene Daten-ID wieder freigegeben.

Return Codes

0	Normales Ende
8	DataID nicht freigegeben (siehe ISPF Systemvariablen)
10	DataID unbekannt (hat LMINIT nicht funktioniert?)
20	Schwerer Fehler

LMGET – ISPF LM Service

```
►►"LMGET DATAID (did) MODE (INVAR) DATALOC (locvar) DATALEN (lenvar)_____◄◄
→►MAXLEN (maxl) " _____◄◄
```

LMGET liest einen Datensatz. Bevor LMGET eingesetzt werden kann, müssen LMINIT und LMO-PEN erfolgreich beendet sein. In einer sequentiellen Datei führt jeder LMGET zum Lesen eines Datensatzes. Bei PO-Dateien greift LMGET auf das Member des letzten LMMFIND-Befehls. Erfolgt zwischen zwei LMGETs ein LMMFIND, liest der zweite LMGET ggfls. ein anderes Member.

DATAID(did)

Daten-ID der Datei, aus der gelesen werden soll.

MODE(INVAR)

MODE() kennt mehr Parameter. In CLIST oder REXX, ist nur die Angabe INVAR möglich. Der Satz liegt in der Variablen aus DATALOC().

DATALOC(locvar)

Variable, in der der Inhalt des gelesenen Datensatzes hinterlegt wird.

DATALEN(lenvar)

In der Variablen wird die Länge des gelesenen Datensatzes abgelegt.

MAXLEN(maxl)

Maximale Satzlänge (die in der Variablen hinterlegt werden können).

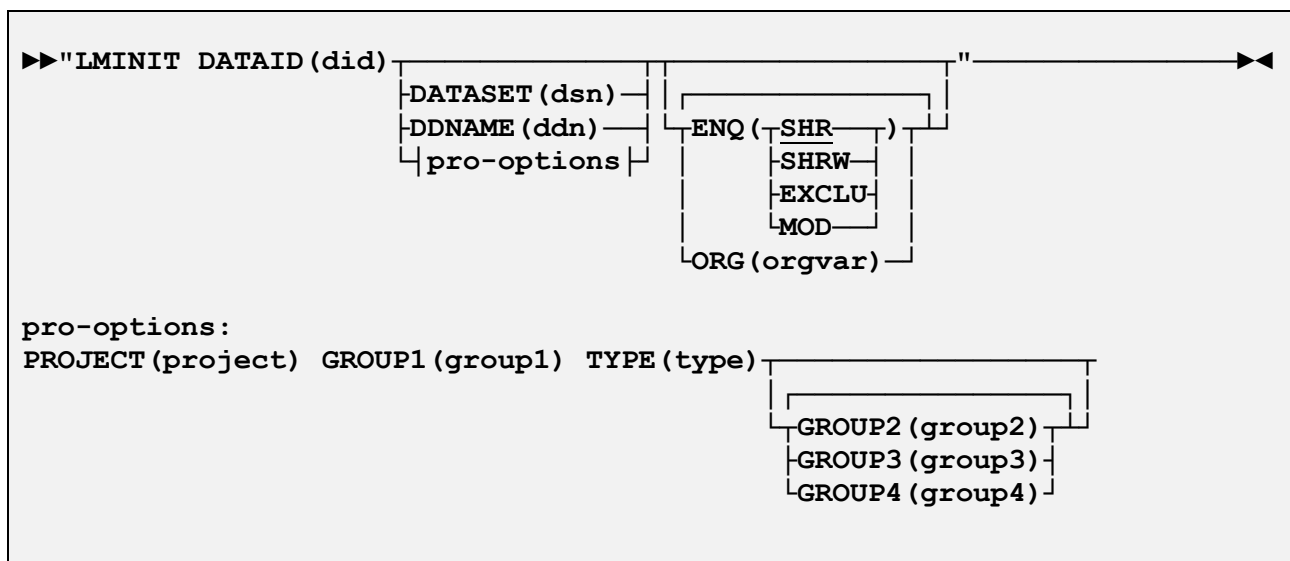
Beispiel

```
"lminit dataid(&did) dataset(seq.data)"
"lmopen dataid(&did) option(input)"
"lmget dataid(&did) mode(invar) dataloc(rec) datalen(recl) maxlen(132)"
do while rc=0
    /* verarbeitung */
    "lmget dataid(&did) ..."
end
```

Liest einen Satz und stellt ihn nach REC. War das Lesen möglich, wird Return Code 0 gesetzt.

Return Codes

- 0 Normales Ende
- 8 EOF erkannt
- 20 Schwerer Fehler

LMINIT – ISPF LM Service

LMINIT weist einer Datei eine Identitätskennung zu. Über diese können andere Services in ISPF (Browse, Edit, LM...) darauf zugreifen. Existiert die Datei ohne zugewiesen zu sein, erfolgt die Zuweisung über LMINIT (expliziter ALLOCATE-Befehl ist nicht erforderlich). Eine Zuweisung durch LMINIT kann durch LMFREE wieder aufgelöst werden.

did

ist der Name der Variablen, in der die Datenidentifikationsnummer (DID) hinterlegt wird. Diese ID wird von den meisten Services als Input benötigt.

DATASET(dsn)

PS- oder PO-Datei. Memberangaben sind nicht möglich. Werden keine Apostrophe benutzt, wird gemäß des TSO-Benutzerprofiles (PREFIX() oder NOPREFIX) verfahren.

DDNAME(ddn)

Synonym unter dem das betreffende Dataset zugewiesen ist

Pro-opts

Angaben von PROJECT-GROUP-TYPE, wobei unter GROUP bis zu vier Angaben gemacht werden können (wie im Edit Entry Panel).

ENQ(...)

Art des Zugriffes. Wurde die Zuweisung über ddname durchgeführt, wird ENQ() ignoriert (Art der Zuweisung wurde bereits im ALLOCATE-Befehl geregelt). Folgende Angaben sind möglich:

SHR

Mehrfachzugriffe sind erlaubt (Standard).

SHRW

SHARE/WRITE nutzt beispielsweise der Editor, wenn er ein Member einer PO-Datei bearbeitet.

EXCLU

Die Datei wird exklusiv belegt.

MOD

Ist nur möglich bei sequentiellen Dateien und fügt beim Schreiben Datensätze an das Ende der Datei an.

ORG(orgvar)

Bestimmt die Variable, in der die Dateiorganisationsform (PS oder PO) abgelegt werden soll.

Beispiel

```
"lminit dataid(did) dataset(edu.data) "
```

Füllt die Variable DID mit der für die Datei user.EDU.DATA vergebenen ID. Der Wert kann in vielen LM-Befehlen (LMOPEN, LMCLOSE, etc.) benutzt werden.

Return Codes

- 0 Normales Ende
- 8 DataID nicht erzeugt (siehe ISPF Systemvariablen)
- 12 Falsche Parameter
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMMADD – ISPF LM Service

```

▶▶"LMMADD DATAID (did) MEMBER (mbr) [STATS (YES)]"
                                     [NO]

```

Fügt ein Member einer PO-Datei hinzu. Voraussetzungen sind die Zuweisung der Datei mit OUTPUT und ein oder mehrere LMPUT-Befehle.

DATAID(did)

Daten-ID der Datei, in der der Directory-Update vollzogen werden soll.

MEMBER(mbr)

Name des Members, das hinzugefügt werden soll.

STATS()

Entscheidet, ob Memberstatistiken mitgeführt werden, oder nicht. Im Falle YES müssen die einzu-tragenden Statistikwerte in folgenden Variablen bereitgestellt werden:

Variable	Inhalt
ZLVERS	Versionsnummer (1-99)
ZLMOD	Modifikationslevel (0-99)
ZLCDATE	Erstellungstag. Achtung: Es gibt nationale Besonderheiten.
ZLMDATE	Datum Last Update (nationale Besonderheiten)
ZLMTIME	Uhrzeit der letzten Änderung (hh:mm)
ZLMSEC	Sekunden der letzten Änderung (2-stellig)
ZLCNORC	Aktuelle Anzahl Datensätze (0-65535)
ZLINORC	Anzahl der Datensätze nach dem ersten SAVE (0-65535)
ZLMNORC	Anzahl modifizierter Sätze (0-65535)
ZLUSER	Benutzerkennung des letzten Veränders
ZLC4DATE	Creationdate mit 4-stelligem Jahr
ZLM4DATE	Datum last modified mit 4-stelligem Jahr
ZLUSER8	8-stellige UserID, sofern dies im System erlaubt ist

Hinweis

Vor z/OS 2.3 gab es noch einen Parameter EXT (für die Extended Statistics). Dieser wird heute nicht mehr unterstützt.

Return Codes

- 0 Normales Ende
- 4 Member in Directory bereits vorhanden
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Eins der folgenden:
 - Falscher Parameter
 - Falsche Dateiorganisation
 - Falsche Angaben für Memberstatistiken
- 14 Kein Satz für das neue Member geschrieben
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMMDEL – ISPF LM Service

```
►►"LMMDEL DATAID (did) MEMBER (mbr) "_____◄◄
```

Löscht ein Member aus einer PO-Datei. Die Datei muss mit dem Attribut OUTPUT geöffnet worden sein.

DATAID(did)

Daten-ID der Datei, aus der ein Member gelöscht werden soll.

MEMBER(mbr)

Name des zu löschenden Members.

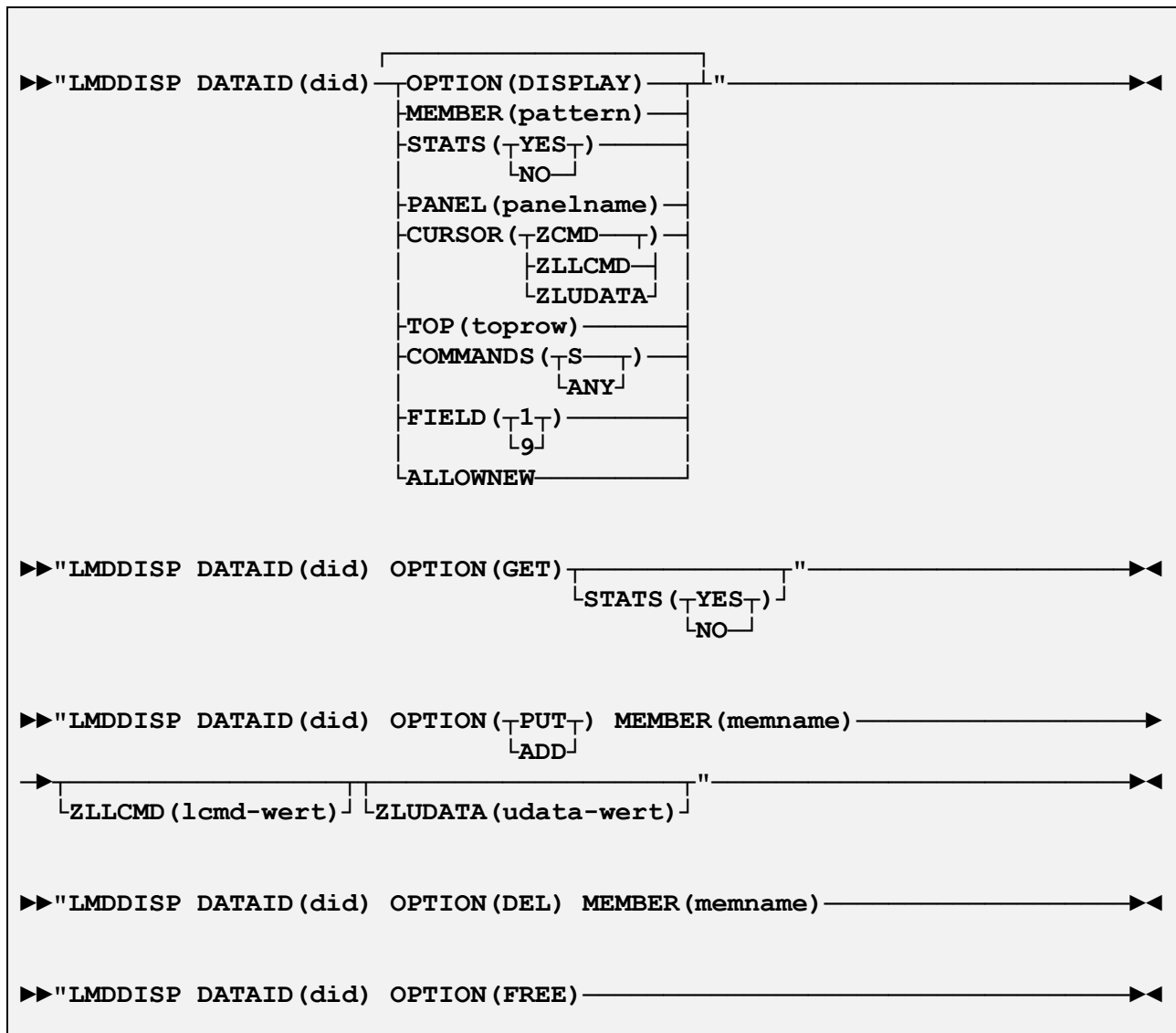
Hinweis

Im Gegensatz zum TSO-Kommando DELETE wird hier die Datei nicht exklusiv belegt.

Return Codes

- 0 Normales Ende
- 4 Member nicht gefunden
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Eins der folgenden:
 - Falsche Parameter
 - Datei nicht mit OUTPUT geöffnet
 - Falsche Dateiorganisation
- 20 Schwerer Fehler

LMMDISP – ISPF LM Service



LMMDISP erlaubt das Einbinden von Memberlist-Bearbeitung in eigene Dialoge auf hohem Komfortniveau. LMINIT und LMOOPEN müssen vorausgehen. Der Service erlaubt grundsätzlich sechs Funktionen:

Display Anzeige des Members. Ein nicht-existierendes Member kann nur angezeigt werden, wenn ALLOWNEW gesetzt ist.

Get Übergibt den Namen des ausgewählten Members

Put Sichert Informationen im Line Command Feld und im User Data Feld der Memberliste

Add Fügt ein Member zur Liste hinzu

Delete Löscht ein Member aus der Liste

FREE Gibt die Liste frei

Nachfolgende Tabelle zeigt, welche Variablen im Shared Pool gesichert werden. Die Spalte Rückgabe zeigt an, wann dies passiert. Steht hier beispielsweise STATS(YES) wird die Variable gesichert, wenn der Service mit STATS(YES) aufgerufen wurde.

Variable	Rückgabe	Inhalt
ZLAC	STATS(YES)	2Byte mit dem Autorisierungscode des Members
ZLALIAS	STATS(YES)	8Byte, Name des Originären Members bei einem Alias
ZLAMODE	STATS(YES)	3Byte AMODE des Members
ZLATTR	STATS(YES)	20Byte für LOAD-Attribute
ZLC4DATE	STATS(YES)	Creation Date (Jahr 4-stellig)
ZLCDATE	STATS(YES)	Creation Date
ZLCNORC	STATS(YES)	Current Number of Records
ZLGENS	STATS(YES)	Wenn YES liegt das Member in einer PO-E/2 Bibliothek mit Versionsführung
ZLGMAX	STATS(YES)	Wenn ZLGENS=YES maximale Anzahl Generations
ZLGNEW	STATS(YES)	Wenn ZLGENS=YES absolute Nummer der neuesten Vorversion
ZLGOLD	STATS(YES)	Wenn ZLGENS=YES absolute Nummer der ältesten Vorversion
ZLGSAB	STATS(YES)	Wenn ZLGENS=YES Anzahl Generationen für dieses Member
ZLINORC	STATS(YES)	Initial Number of Records
ZLLCMD	Immer	Benutztes Zeilenkommando
ZLLIB	STATS(YES)	Zahl von 1 bis 16 bei concatenated Libraries
ZLM4DATE	STATS(YES)	Date Last Modified (Jahr 4-stellig)
ZLMDATE	STATS(YES)	Date Last Modified
ZLMEMBER	Immer	Name des ausgewählten Members
ZLMOD	STATS(YES)	Anzahl modifizierter Sätze
ZLMSEC	STATS(YES)	Sekundenangabe der letzten Änderung
ZLMTIME	STATS(YES)	Uhrzeit der letzten Änderung
ZLMTOP	Immer	Das oberste sichtbare Member
ZLPDSUDA	STATS(YES)	Wert der PDS Directory User Data Area
ZLRMODE	STATS(YES)	3Byte RMODE des Members
ZLSIZE	STATS(YES)	8Byte LOAD-Modul-Größe in Hex
ZLSSI	STATS(YES)	8Byte der SSI-Informationen für LOADs
ZLTTR	STATS(YES)	6Byte TTR des Members (nur LOADs)
ZLUDATA	STATS(YES)	UserID Last Update
ZLVERS	STATS(YES)	PDF Versionsnummer
ZSCLM	STATS(YES)	Ast Modified durch SCLM oder ISPF

DATAID(did)

Ist die Datenidentifikationsnummer (DID), welche angezeigt werden soll.

OPTION()

DISPLAY	Eine Memberliste wird erstellt und angezeigt, sofern keine existiert
GET	Infos über das nächste und die folgenden Member, wenn nach einer Option(Display) ein oder mehrere Member selektiert wurden
PUT	Sichert Informationen des bearbeiteten Members im Linecommand- und UserData-Feld
ADD	Fügt einen Membernamen in die Liste ein. Das Member darf weder in der Liste, noch physisch in der Dateiverkettung vorhanden sein
DEL	Entfernt einen Membernamen aus der Liste (kein physischen Löschen aus Datei)

FREE Lösche die Memberliste aus dem Speicher

MEMBER(pattern)

Steuert die Namen der Member, welche in der Liste angezeigt werden sollen. Wildcards sind erlaubt.

STATS(YES|NO)

Bestimmt, ob die Memberstatistiken in Variablen geliefert werden sollen.

PANEL(panelname)

Name des Panels, mit dem die Anzeige erfolgen soll.

CURSOR(ZCMD|ZLLCMD|ZLUDATA)

Cursorposition. ZLLCMD und ZLUDATA bewirken, dass der Cursor in das entsprechende Feld des obersten sichtbaren Members positioniert wird.

TOP(top-member)

Das Member, welches als oberstes sichtbar sein soll. Ist die Memberliste nach Namen sortiert und das Member nicht vorhanden, wird das davorliegende angezeigt.

COMMAND(S|ANY)

Ist "S" angegeben, kann als Command nur S benutzt werden, um ein Member zu selektieren.

Bei ANY ist jedes Zeichen oder jeder String erlaubt.

FIELD(1|9)

Regelt, ob das Zeilenkommandofeld der Memberanzeige 1Byte oder 9Byte lang ist.

ALLOWNEW

Nicht existierende Member können auch selektiert werden.

Return Codes OPTION(DISPLAY)

- 0 Ein oder mehrere Member wurden durch Zeilenkommandos ausgewählt und/oder ein LMMDISP unbekannter Primärbefehl wurde eingegeben
- 4 Die angegebene Datei ist leer oder es wurde kein Member entsprechend der Kriterien ausgewählt
- 8 END oder RETURN wurde eingegeben
- 10 Es existiert keine Dateiliste zur DataID (Probleme beim LMINIT?)
- 12 Eins der folgenden:
 - Datei nicht geöffnet
 - Datei nicht partitioned
 - Falscher Parameter
 - Falsche DS-Org
 - Syntaxfehler
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

Zusätzlich für OPTION(GET)

- 0 Normales Ende
- 8 Keine weiteren Member ausgewählt

Zusätzlich für OPTION(PUT)

- 0 Normales Ende

8 Das angegebene Member existiert nicht in der Liste

Zusätzlich für OPTION(ADD)

0 Normales Ende

8 Member bereits in der Liste vorhanden

Zusätzlich für OPTION(DEL)

0 Normales Ende

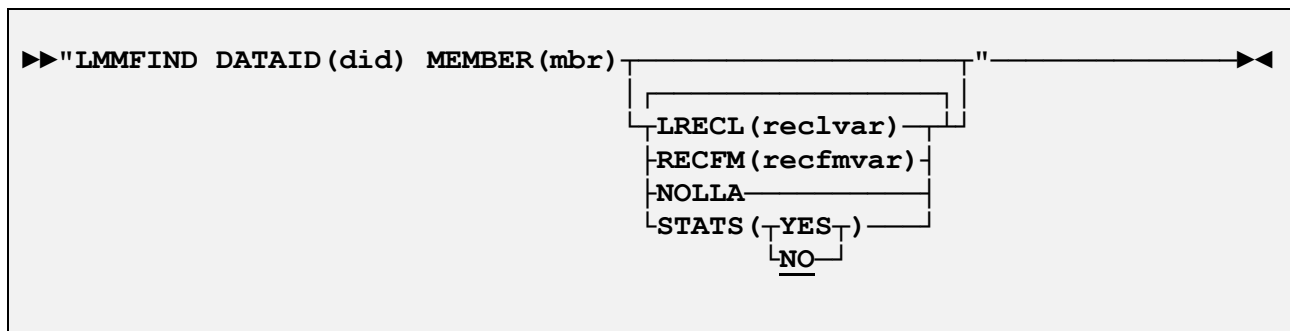
8 Das angegebene Member existiert nicht in der Liste

Zusätzlich für OPTION(FREE)

0 Normales Ende

8 Die angegebene DataID enthält keine Liste

LMMFIND – ISPF LM Service



LMMFIND sucht ein Member in einer PO-Datei und macht die Memberstatistiken eines Members verfügbar. LMINIT und LMOOPEN müssen vorausgehen.

DATAID(did)

Daten-ID der Datei, in der gesucht werden soll.

MEMBER(mbr)

Member, das in der Datei gesucht werden soll.

LRECL(reclvar)

Variable, in der die Satzlänge abgelegt wird.

RECFM(recfmvar)

In der Variablen wird das Satzformat hinterlegt.

NOLLA

Falls die betreffende Datei LLA-managed ist, sollte der Parameter angegeben werden, damit bei Abweichungen nicht die Startadresse des Members aus dem LLA-Adressraum benutzt wird.

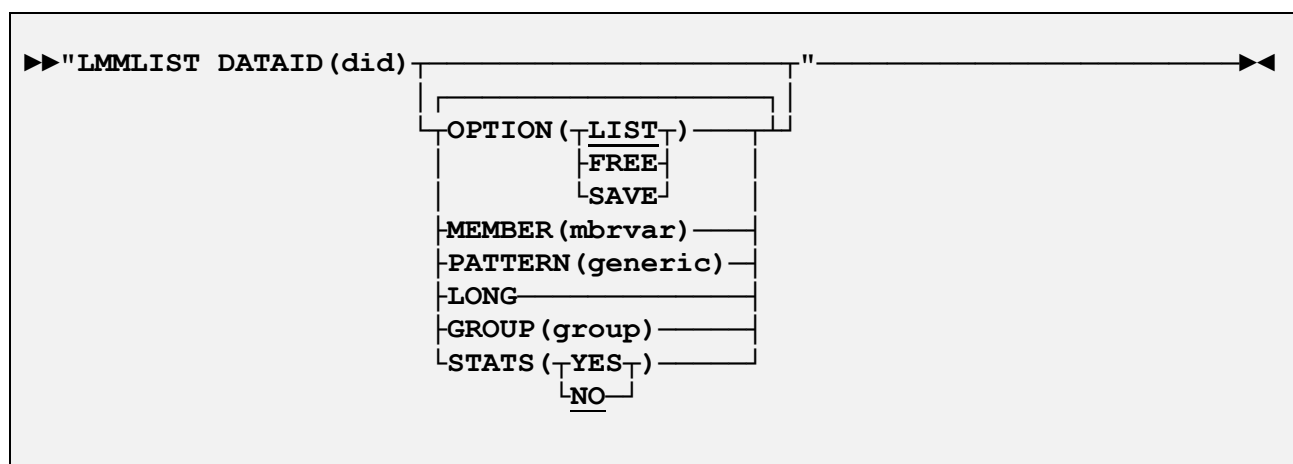
STATS()

Im Falle YES werden folgende Variablen bereitgestellt:

Variable	Inhalt
ZLVERS	Versionsnummer (1-99)
ZLMOD	Modifikationslevel (0-99)
ZLCDATE	Erstellungstag. Achtung: Es gibt nationale Besonderheiten.
ZLGENS	Wenn YES liegt das Member in einer PO-E/2 Bibliothek mit Versionsführung
ZLGMAX	Wenn ZLGENS=YES maximale Anzahl Generations
ZLGNEW	Wenn ZLGENS=YES absolute Nummer der neuesten Vorversion
ZLGOLD	Wenn ZLGENS=YES absolute Nummer der ältesten Vorversion
ZLGSAV	Wenn ZLGENS=YES Anzahl Generationen für dieses Member
ZLMDATE	Datum Last Update (nationale Besonderheiten)
ZLMTIME	Uhrzeit der letzten Änderung (hh:mm)
ZLMSEC	Sekunden der letzten Änderung (2-stellig)
ZLCNORC	Aktuelle Anzahl Datensätze (0-65535)
ZLINORC	Anzahl der Datensätze nach dem ersten SAVE (0-65535)
ZLMNORC	Anzahl modifizierter Sätze (0-65535)
ZLUSER	Benutzerkennung des letzten Veränders
ZLC4DATE	Creationdate mit 4-stelligem Jahr
ZLM4DATE	Datum last modified mit 4-stelligem Jahr
ZLUSER8	8-stellige UserID, sofern dies im System erlaubt ist

Return Codes

- 0 Normales Ende
- 4 Member nicht verfügbar
- 8 Member nicht gefunden
- 10 DataID unbekannt (LMINIT-Problem?)
- 12 Falscher Parameter oder Datei nicht geöffnet
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMMLIST – ISPF LM Service

LMMLIST erzeugt eine Memberliste im Hintergrund und erlaubt, alle Member einer PO-Datei zu ermitteln.

DATAID(did)

Daten-ID der Datei, deren Member ermittelt werden sollen.

OPTION()

Steuert, ob der ermittelte Membername in die Memberliste eingetragen (LIST) oder freigegeben werden soll (FREE).

Mit SAVE kann eine Memberliste auf Datei ausgegeben werden

MEMBER(mbrvar)

Variablen, in der der Name eines Members hinterlegt werden soll.

PATTERN(generic)

Erlaubt eine generische Angabe für Membernamen

GROUP(group)

Wird nur benötigt, wenn mit OPTION(SAVE) gearbeitet wird. In diesem Fall wird die Datei, in welcher das Ergebnis abgelegt wird, heißen: <prefix>.<group>.MEMBERS. Wird zusammen mit SAVE keine GROUP-Angabe kodiert, wird das Ergebnis im ISPF-List-Dataset abgelegt (das will für gewöhnlich NIEMAND).

LONG

Wenn mit OPTION(SAVE) gearbeitet wird, bestimmt LONG, dass alle Datumsvariablen mit 4-stelligem Jahr ausgegeben werden.

STATS()

Im Falle YES liegen die Informationen in folgenden Variablen:

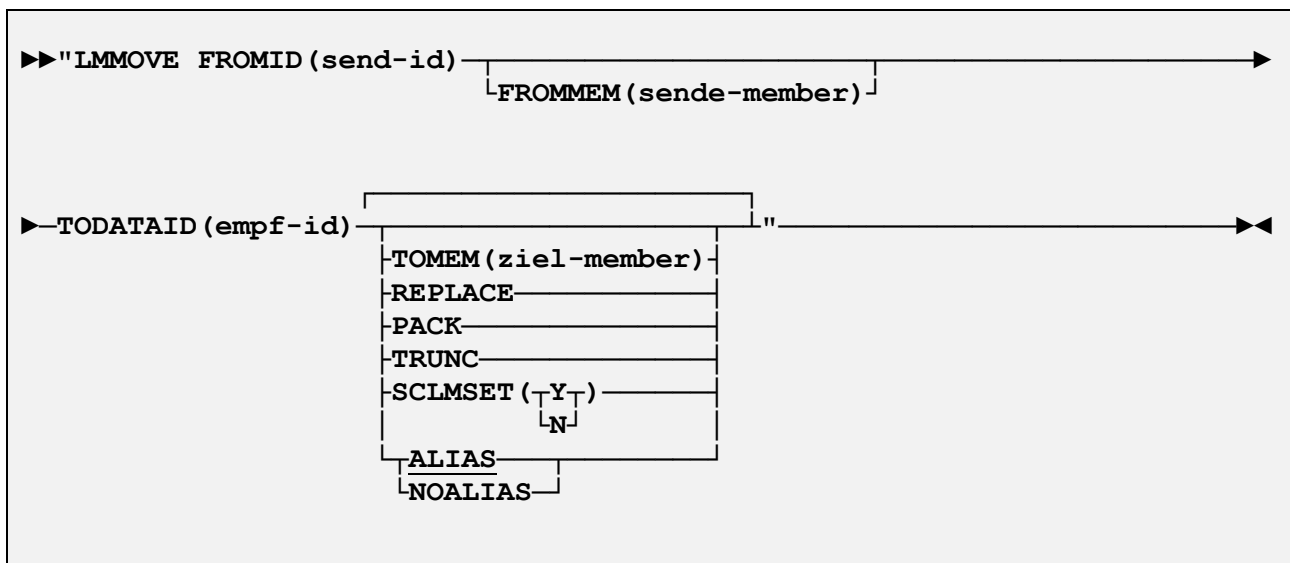
Variable	Inhalt
ZLVERS	Versionsnummer (1-99)
ZLMOD	Modifikationslevel (0-99)
ZLCDATE	Erstellungstag. Achtung: Es gibt nationale Besonderheiten.
ZLGENS	Wenn YES liegt das Member in einer PO-E/2 Bibliothek mit Versionsführung
ZLGMAX	Wenn ZLGENS=YES maximale Anzahl Generations
ZLGNEW	Wenn ZLGENS=YES absolute Nummer der neuesten Vorversion
ZLGOLD	Wenn ZLGENS=YES absolute Nummer der ältesten Vorversion
ZLGSAV	Wenn ZLGENS=YES Anzahl Generationen für dieses Member
ZLMDATE	Datum Last Update (nationale Besonderheiten)
ZLMTIME	Uhrzeit der letzten Änderung (hh:mm)
ZLMSEC	Sekunden der letzten Änderung (2-stellig)
ZLCNORC	Aktuelle Anzahl Datensätze (0-65535)
ZLINORC	Anzahl der Datensätze nach dem ersten SAVE (0-65535)
ZLMNORC	Anzahl modifizierter Sätze (0-65535)
ZLUSER	Benutzerkennung des letzten Veränders

ZLC4DATE	Creationdate mit 4-stelligem Jahr
ZLM4DATE	Datum last modified mit 4-stelligem Jahr
ZLUSER8	8-stellige UserID, sofern dies im System erlaubt ist

Return Codes

- 0 Normales Ende
 LIST: Memberliste aufgebaut, Membervariable erstmals gefüllt
 FREE: Die Memberliste wurde freigegeben
- 4 Memberliste leer
- 8 LIST: Listenende
 FREE: Liste existiert nicht
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Datei nicht geöffnet oder kein PO-Format
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMMOVE – ISPF LM Service



LMMOVE überträgt ein einzelnes Member aus einer PO-Datei oder eine sequentielle Datei. Nach der Übertragung wird der Sender gelöscht.

Nur Satzformate F und V können gepackt werden.

FROMID und TODATAID können auf das gleiche Dataset zeigen, aber nicht die gleiche ID haben.

Dateien mit dem Attribut UNMOVEABLE können nicht übertragen werden.

FROMID(send-id)

DatenID der Sendedatei

FROMMEM(sende-member)

Benennt das zu übertragende Member. Generische Angaben sind möglich.

TODATAID(empf-id)

DatenID der Zielfeile

TOMEM(ziel-member)

Benennt das Zielfeile

REPLACE

Gleichnamig in der Zielfeile vorhandene Member werden überschrieben

PACK

Wenn angegeben, werden die Daten in der Zielfeile im gepackten Format gespeichert (Sollte man sich genau überlegen. Ohne ISPF sind die Daten nicht mehr zu lesen).

TRUNC

Abschneiden der Datensätze, wenn die Satzlänge der Sendedatei größer als die der Empfangsdatei ist.

SCLMSET(Y | N)

ISPF pflegt ein Bit im Directory, wenn SCLM eingesetzt wird.

ALIAS | NOALIAS

Bestimmt, ob Alias-Einträge mitkopiert werden, oder nicht.

Return Codes

0	Normales Ende
4	Member nicht verfügbar. Entweder ist die Sendedatei leer oder kein Member erfüllt die FROM-Angabe
8	FROMMEM nicht vorhanden
10	Es existiert keine Dateiliste zur DataID
12	Einer der folgenden Gründe: Member in Zielfeile bereits vorhanden und REPLACE war nicht angegeben Ein oder mehrere Member in der Zielfeile sind belegt Falsche DS-Org Dateiattribute für PACK-Option falsch OPEN-Fehler
16	Truncation Fehler
20	Schwerer Fehler

LMMREN – ISPF LM Service

```
►►"LMMREN DATAID (did) MEMBER (alt) NEWNAME (neu) "◄◄
```

Weist einem Member einer PO-Datei einen neuen Namen zu.

DATAID(did)

übergibt die Daten-ID der Datei, die geöffnet werden soll.

MEMBER(alt)

Alter Name des Members.

NEWNAME(neu)

Neuer Name des Members.

Return Codes

0	Normales Ende
4	Neuer Name bereits vorhanden
8	Member nicht gefunden
10	DataID unbekannt (LMINIT fehlt?)
12	Eins der folgenden:
	Falsche Parameter
	Datei nicht zum Schreiben geöffnet
	Kein PO-Format
20	Schwerer Fehler

LMMREP – ISPF LM Service

```
►►"LMMREP DATAID (did) MEMBER (mbr) ◄◄  
    [STATS (YES)]  
    [NO]
```

Überschreibt ein Member einer PO-Datei. Es muss gewährleistet sein, dass dem Befehl eine Zuweisung der Datei mit OUTPUT vorausging:

Nach erfolgreichem LMPUT führt LMREP die Aktualisierung des Directorys durch.

DATAID(did)

Daten-ID der Datei, in der der Update des Directorys vollzogen werden soll.

MEMBER(mbr)

Name des Members, das überschrieben werden soll.

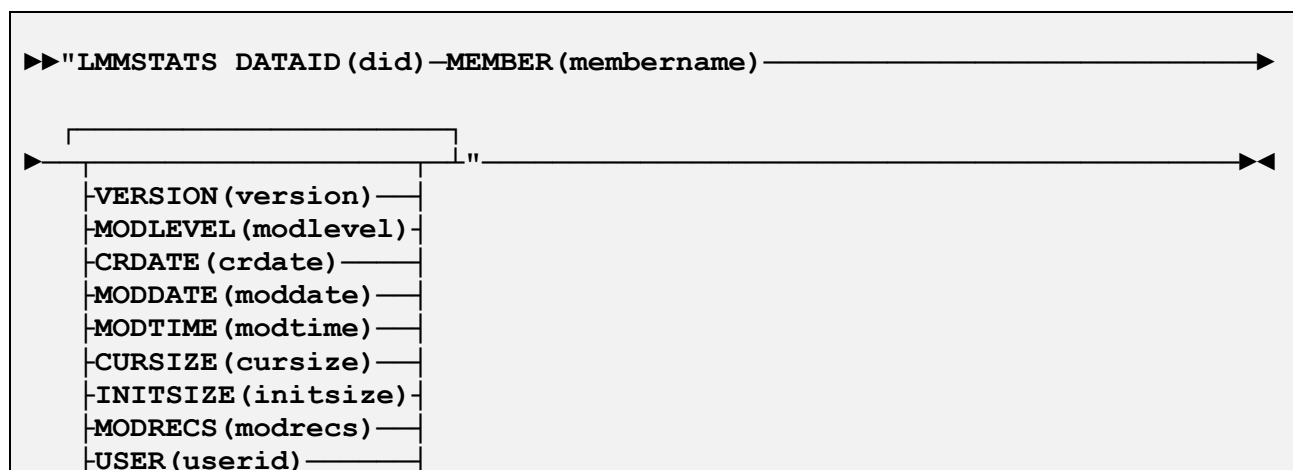
STATS()

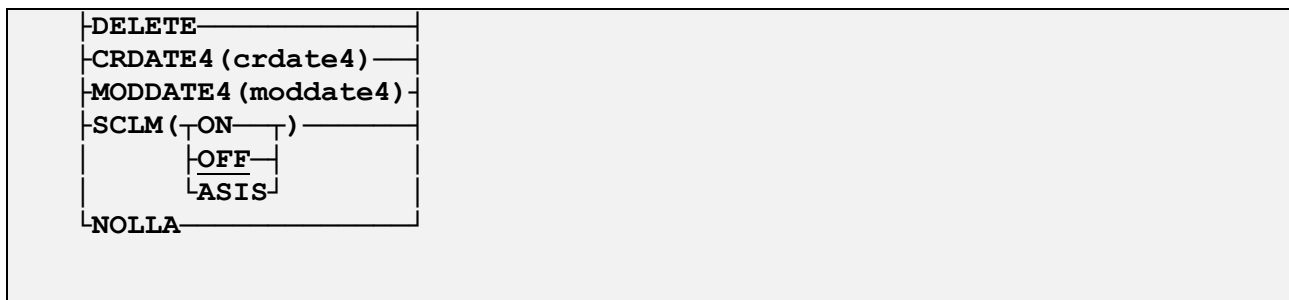
Entscheidet, ob die Memberstatistiken mitgeführt werden sollen, oder nicht. Wenn ja sind die einzutragenden Statistikwerte in folgenden Variablen bereit zu stellen:

Variable	Inhalt
ZLVERS	Versionsnummer (1-99)
ZLMOD	Modifikationslevel (0-99)
ZLCDATE	Erstellungstag. Achtung: Es gibt nationale Besonderheiten.
ZLMDATE	Datum Last Update (nationale Besonderheiten)
ZLMTIME	Uhrzeit der letzten Änderung (hh:mm)
ZLMSEC	Sekunden der letzten Änderung (2-stellig)
ZLCNORC	Aktuelle Anzahl Datensätze (0-65535)
ZLINORC	Anzahl der Datensätze nach dem ersten SAVE (0-65535)
ZLMNORC	Anzahl modifizierter Sätze (0-65535)
ZLUSER	Benutzerkennung des letzten Veränders
ZLC4DATE	Creationdate mit 4-stelligem Jahr
ZLM4DATE	Datum last modified mit 4-stelligem Jahr
ZLUSER8	8-stellige UserID, sofern dies im System erlaubt ist

Return Codes

- 0 Normales Ende
- 4 Member in Directory bereits vorhanden
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Eins der folgenden:
 - Datei nicht mit Output geöffnet
 - Kein PO-Format
 - Falsche Angaben für die Memberstatistik
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMMSTATS – ISPF LM Service



LMMSTATS ändert oder löscht die Memberstatistiken.

LMINIT und LMOPEN (OUTPUT) müssen vorausgehen.

Werden die Schlüsselworte nicht angegeben, werden ihre Werte berechnet.

DATAID(did)

Daten-ID der Datei, deren Memberstatistiken geändert werden sollen.

MEMBER(membername)

Member, dessen STATS-Werte verändert werden.

VERSION(version)

Versionsnummer (integer zwischen 1 und 99)

MODLEVEL(modlevel)

Modifikationslevel (integer zwischen 0 und 99)

CRDATE(crdate)

Erstellungsdatum im Format YY/MM/DD

MODDATE(moddate)

Datum der letzten Änderung im Format YY/MM/DD

MODTIME(modtime)

Uhrzeit der letzten Änderung

CURSIZE(cursize)

Aktuelle Anzahl Zeilen

INITSIZE(initsize)

Anzahl Zeilen beim ersten Speichern (0-2¹⁶)

MODRECS(modrecs)

Anzahl modifizierter Sätze (0-2¹⁶)

USER(userid)

UserID des letzten Updates

DELETE

Löscht die Statistikeinträge

CRDATE4(crdate4)

Erstellungsdatum im Format YYYY/MM/DD

MODDATE4(moddate4)

Datum der letzten Änderung im Format YYYY/MM/DD.

SCLMSET(ON|OFF|ASIS)

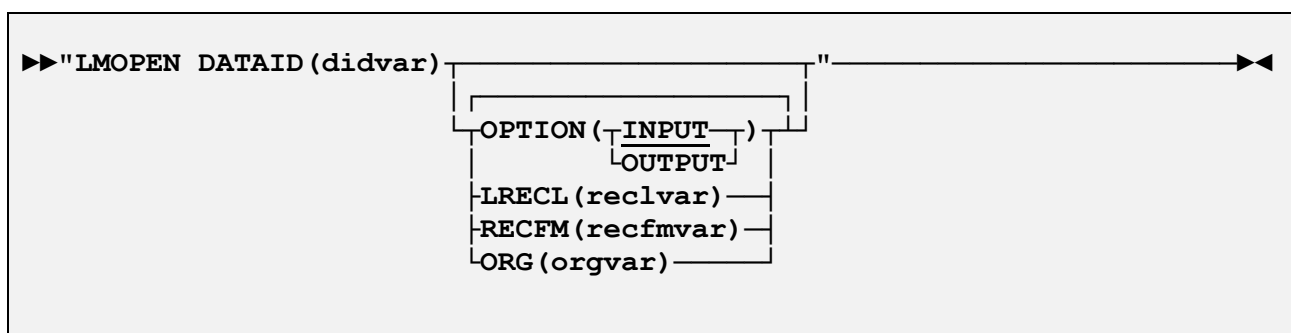
Setzt ein Bit, anhand dessen ISPF entscheiden kann, ob die letzte Änderung unter SCLM oder ISPF stattgefunden hat. ASIS lässt das Bit unverändert.

NOLLA

Liegt die Datei unter LLA, soll der Parameter benutzt werden, um sicher zu stellen, dass cached entries nicht benutzt werden.

Return Codes

- 0 Normales Ende
- 4 Datei leer oder keine Übereinstimmung der Membernamen
- 8 Member nicht gefunden
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Einer der folgenden Gründe:
 - Falscher Parameterwert
 - Datei nicht PO
 - DataID repräsentiert eine Dateiverknüpfung
 - Datei bereits für OUTPUT geöffnet
- 20 Schwerer Fehler

LMOPEN – ISPF LM Service

LMOPEN öffnet eine Datei über ihre Daten-ID. (Voraussetzung LMINIT):

DATAID(did)

übergibt die Daten-ID der Datei, die geöffnet werden soll.

OPTION()

INPUT Lesen
OUTPUT Schreiben

LRECL(reclvar)**RECFM(recfmvar)****ORG(orgvar)**

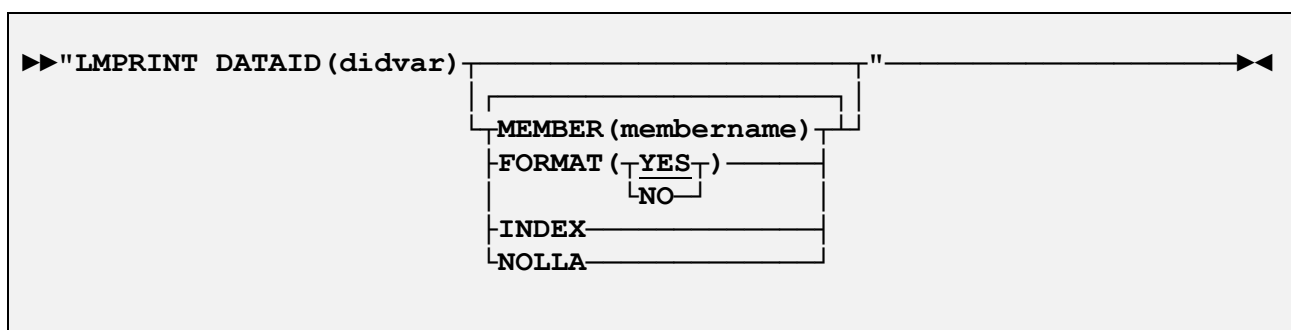
Name der Variablen, in der die Satzlänge, das Satzformat und die Organisationsform der Datei hinterlegt wird.

Beispiel

```
address ispexec
"lminit dataid(did) dataset(kurs.data) "
"lmopen dataid(&did) option(input) "
Öffnet die durch LMINIT ermittelte DID zum Lesen.
```

Return Codes

- 0 Normales Ende
- 8 Datei konnte nicht geöffnet werden
- 10 DataID unbekannt (LMINIT fehlt?)
- 12 Datei bereits geöffnet, Lesen mit MOD oder Schreiben mit SHR
- 16 Fehler bei der Interpretation von Variablen
- 20 Schwerer Fehler

LMPRINT – ISPF LM Service

LMPRINT führt eine Druckausgabe in das ISPF-LIST-Dataset durch. Gedruckt werden kann eine sequentielle Datei, ausgewählte Member einer PO-Datei oder ein Memberverzeichnis (spielt heute keine wirkliche Rolle mehr).

DATAID(did)

übergibt die Daten-ID der Datei, die gedruckt werden soll.

MEMBER(membername)

Name des Members, das gedruckt werden soll.

FORMAT(YES|NO)

Regelt, ob der Output formatiert ausgegeben werden soll.

INDEX

Eine Memberlist wird gedruckt

NOLLA

Liegt die Datei unter LLA, soll der Parameter benutzt werden, um sicher zu stellen, dass cached entries nicht benutzt werden.

Return Codes

0	Normales Ende
4	Leere Datei oder leeres Member oder kein Member passt zur Auswahl
8	Member nicht gefunden
10	DataID unbekannt (LMINIT fehlt?)
12	DS-Org oder Parameter falsch
20	Schwerer Fehler

LMPUT – ISPF LM Service

```
►►"LMPUT DATAID(did) MODE(INVAR) DATALOC(locvar) DATALEN(len) [NOBSCAN]"◄◄
```

LMPUT schreibt einen Datensatz. Bevor LMPUT eingesetzt werden kann, müssen LMINIT und LMOPEN (mit Option OUTPUT) erfolgreich beendet worden sein.

Soll ein Member einer PO-Datei beschrieben werden, muss nach allen LMPUT-Befehlen ein LMM-ADD- oder LMMREP-Service aufgerufen werden, um das Directory entsprechend zu aktualisieren.

Soll in eine Sequentielle Datei geschrieben werden, muss nach dem letzten LMPUT-Befehl LMCLOSE benutzt werden.

DATAID(did)

Daten-ID der Datei, in die geschrieben werden soll.

MODE(INVAR)

MODE() kennt unterschiedliche Parameter. Mit CLIST oder REXX, ist nur die Angabe INVAR möglich. Der zu schreibende Datensatz wird aus der Variablen aus DATALOC() gelesen.

DATALOC(locvar)

Bestimmt die Variable, aus der der zu schreibende Datensatz gelesen wird.

DATALEN(len)

Bestimmt die Länge des zu schreibenden Datensatzes.

NOBSCAN

Nachlaufende Blanks am Datensatzende werden nicht beschnitten, wenn in eine Datei mit variabler Satzlänge geschrieben wird (NO BackSCAN)

Beispiel

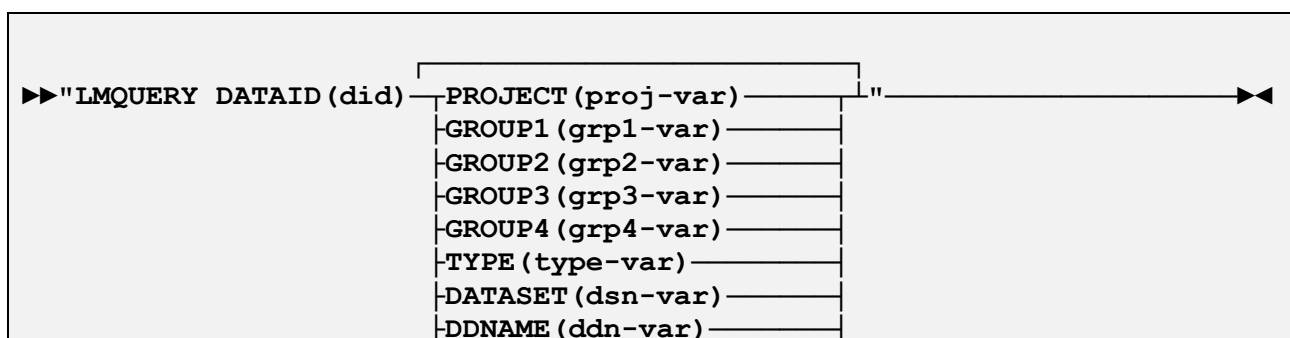
```
address ispxec
"lminit dataid(&did) dataset(kurs.data) "
"lmopen dataid(&did) option(output) "
"lmmfind dataid(&did) member(mbr1) "
say "Eingabe ?? (Erfassungsende durch <ENTER>)"
do forever
    parse external satz
    if length(satz) = 0 then leave
    "lmput dataid(&did) mode(invar)dataloc(satz) datalen(132) "
end
"lmmrep dataid(&did) "
"lmfree dataid(&did) "
```

In einer Erfassungsschleife können beliebig viele Datensätze erfasst werden, bis die Länge der Variablen SATZ gleich 0 ist (nur E ohne Eingabe). Wird ein Datensatz empfangen, wird er im Member MBR1 hinterlegt. Durch LMMREP erfolgt vor Programmende die Aktualisierung des Directorys.

Return Codes

- 0 Normales Ende
- 8 EOF erkannt
- 20 Schwerer Fehler

LMQUERY – ISPF LM Service



	VOLUME (vsn-var)	
	ENQ (enq-var)	
	OPEN (open-var)	
	LRECL (recl-var)	
	RECFM (recfm-var)	
	DSORG (dsorg-var)	
	ALIAS (alias-var)	
	PASSWORD (password-var)	
	OVOLUME (ovol-var)	

LMQUERY erlaubt das Einholen einer Vielzahl von Informationen über eine Datei.

DATAID(did)

Daten-ID der Datei, in die geschrieben werden soll.

PROJECT(proj-var)

PROJECT aus LMINIT

GROUP1(grp1-var)

GROUP1 aus LMINIT

GROUP2(grp2-var)

GROUP2 aus LMINIT

GROUP3(grp3-var)

GROUP3 aus LMINIT

GROUP4(grp4-var)

GROUP4 aus LMINIT

TYPE(type-var)

TYPE aus LMINIT

DATASET(dsn-var)

Vollständiger Dateiname aus LMINIT

DDNAME(ddn-var)

DD-Name aus LMINIT. Wenn kein DD-Name im LMINIT angegeben wurde, wird der DD-Name zurückgereicht, welcher durch ISPF generiert wurde.

VOLUME(vsn-var)

VOLUME-Angabe aus LMINIT

ENQ(enq-var)

ENQ-Parameter des LMINIT

OPEN(open-var)

Art des OPENS im LMOPEN-Service. Wurde die Datei nicht geöffnet, werden Blanks zurückgegeben.

LRECL(recl-var)

Die Satzlänge. Wurde die Datei nicht geöffnet, wird Blank zurückgegeben.

RECFM(recfm-var)

Das Satzformat. Wurde die Datei nicht geöffnet, wird Blank zurückgegeben. Einer der folgenden Werte kann zurückgegeben werden, wenn die Datei geöffnet wurde:

RECFM	Bedeutung
F	Sätze fester Länge
V	Sätze variabler Länge
U	Satzformat undefined (LOADLIBs)
B	Geblockte Sätze
T	Track-Überlauf
M	Maschinen-Steuerzeichen
A	ANSI-Code

DSORG(dsorg-var)

Dateiorganisation (PS oder PO)

ALIAS(alias-var)

Zeigt über Y oder N ob der Dateiname ein ALIAS ist.

PASSWORD(password-var)

Password aus LMINIT.

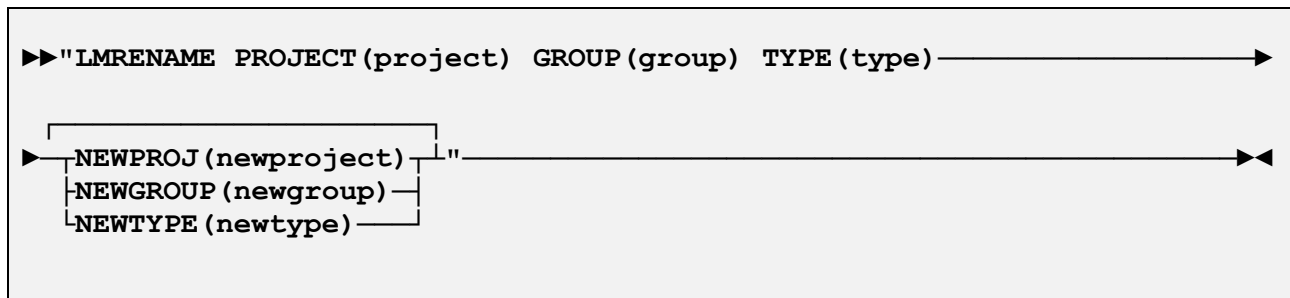
OVOLUME(ovol-var)

Vol-Ser-Angabe des Datenträgers, auf dem die Datei liegt.

Return Codes

- 0 Normales Ende
- 4 Keine vernünftige Info für das angegebene Keyword (Blank wird zurückgegeben)
- 10 Für DataID ist keine Datei zugewiesen (LMINIT fehlt?)
- 16 Trunction oder Translation Fehler beim Zugriff auf eine Dialogvariable
- 20 Schwerer Fehler

LMRENAME – ISPF LM Service



Umbenennen einer Datei mit drei Qualifiern (mit je maximal 8 Byte Länge).

DATAID(did)

übergibt die Daten-ID der Datei, die geöffnet werden soll.

PROJECT | GROUP | TYPE

Benennen die drei Qualifier der Datei, die umbenannt werden soll.

NEWPROJ | NEWGROUP | NEWTYPE

Benennt die neuen Qualifier. Mindestens eine der drei Angaben muss erfolgen.

Beispiel

address ispexec

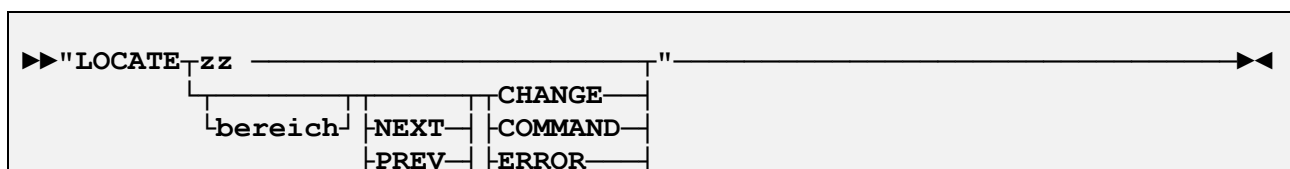
```
"LMRENAME PROJECT(SP1) GROUP(EDU) TYPE(DATA) " ,
  "NEWGROUP(TEST) "
```

Umbenennen der Datei SP1.EDU.DATA auf SP1.TEST.DATA

Return Codes

- 0 Normales Ende
- 4 Neuer Name existiert bereits
- 8 Eins der folgenden:
 - Datei nicht vorhanden
 - Rename oder Catalog missglückt
 - Dateiname ist ein Alias
- 12 Falsche Parameterwerte
- 20 Schwerer Fehler

LOCATE – ISPF Edit Macro



FIRST	EXCLUDED
LAST	LABEL
	SPECIAL
	INFOLINE
	MSGLINE
	NOTELINE

Blättert zu einer bestimmten Zeile und zeigt sie als oberste am Bildschirm.

zz

Zeilenzeiger (Label oder relative Zeilennummer).

Richtung

NEXT	Suche beginnt ab Cursorposition in Richtung Dateiende
FIRST	Die Suche beginnt am Dateianfang in Richtung Ende. Geändert wird der erste vorkommende Begriff
LAST	Wie FIRST aber vom Ende in Richtung Anfang
PREV	Wie NEXT aber in Richtung Dateianfang

CHANGE

Gesucht wird nach einer Zeile mit Änderungshinweis ==CHG>. Change kann mit CHG abgekürzt werden.

COMMAND

Zeilen mit einem unvollständigen Befehl im Zeilenkommandofeld (Kürzel COM).

ERROR

Es werden Zeilen mit Fehlerhinweis aufgesucht (==ERR>). Erlaubtes Kürzel ist ERR.

EXCLUDED

Sucht nach ausgeblendeten Zeilen. Mögliche Abkürzung ist X.

LABEL

Gesucht wird nach einer beliebigen Zeile mit Label. Als Kürzel ist LAB erlaubt.

SPECIAL

Special kann durch SPE abgekürzt werden und sucht nach Zeilen, die keine Daten enthalten. Im Detail können dies sein:

==BNDS>	Boundaryzeilen
==MASK>	Maskenzeilen
==MSG>	Nachrichtenzeilen
=NOTE=	Hinweiszeilen
==PROF>	Profilzeilen

`==TABS>` Tabulatorzeilen

INFOLINE

Sucht Zeilen mit `=====` im Zeilennummernfeld.

MSGLINE

Sucht Zeilen mit `==MSG>` im Zeilennummernfeld.

NOTELINE

Sucht Zeilen mit `=NOTE=` im Zeilennummernfeld.

bereich

Zwischen zwei Zeilenzeiger eingegrenzter Bereich. Zeilenzeiger kann ein Label oder eine relative Zeilennummer sein.

Hinweis

Wird ein Label angesteuert das nicht vorhanden ist, wird RC 20 gesetzt. Um dies zu erkennen sollte der Befehl LINENUM eingesetzt werden. LINENUM liefert in gleicher Situation RC 8.

Beispiele

`"locate note"`

Zeigt am oberen Bildschirmrand die nächste Zeile mit einem Hinweis.

`"locate first com"`

Blättert zur ersten Zeile mit einem nicht ausgeführten Zeilenbefehl.

`"locate prev x"`

Sucht die erste ausgeblendete Zeile oberhalb der Cursorposition.

Return Codes

0	Normales Ende
4	Zeile nicht gefunden
8	Datei oder Member leer
20	Schwerer Fehler

LOG – ISPF DM Service



```
>>LOG MSG(msg-id)
```

Schreibt in die ISPF-LOG-Datei

MSG(msg-id)

benennt die Nachricht, die in die LOG-Datei eingetragen werden soll.

Return Codes

0	Normales Ende
12	Falsche Syntax in der Nachricht oder Nachricht nicht gefunden
20	Schwerer Fehler

LRECL – ISPF Edit Macro

```
►►" (variable) = LRECL"◄◄
```

Übergibt die logische Satzlänge einer Datei. Sind Zeilensequenznummern gesetzt, sind sie in der Satzlänge enthalten. Wird die Länge des editierbaren Datensatzes (ohne Sequenznummern) benötigt, kann DATA_WIDTH eingesetzt werden.

variable

Enthält die logische Satzlänge.

Beispiele

```
"(recl) = lrecl"  
if recl > 132 then ...  
Die Satzlänge wird abgefragt.
```

Return Codes

0	Normales Ende
20	Schwerer Fehler

LRECL – JCL DD Parameter

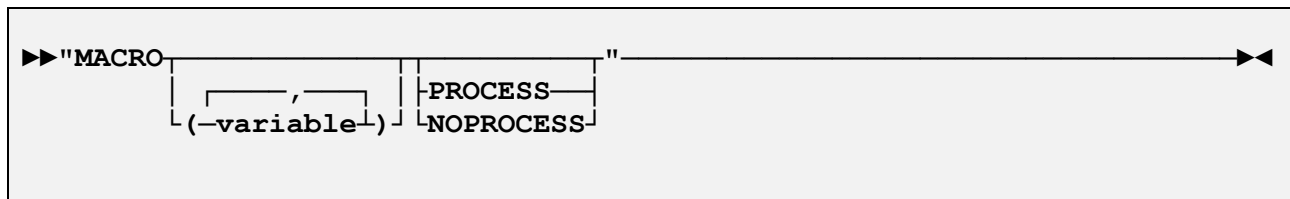
```
►►LRECL=satzlänge◄◄
```

Kann als Teil des DCB-Parameters oder auch außerhalb angegeben werden.

satzlänge

Enthält die logische Satzlänge.

MACRO – ISPF Edit Macro



Identifiziert das folgende Programm als Macro und wird sowohl in einem Programm-Macro, als auch in einem CLIST/REXX-Macro benötigt. In einem REXX-Macro muss es die erste Instruktion sein, die nicht vom REXX-Interpreter ausgeführt wird.

variable

Parameter, die bei Makroaufruf übergeben werden können. Die Zuweisung der Variablen erfolgt von links nach rechts. Parametertrennzeichen ist Komma oder Blank. Apostrophe(') oder Quotes("") können Parameter eingrenzen, wenn sie Blanks oder Komma enthalten.

Werden mehr Variable kodiert, als vom Anwender Parameter angegeben, sind die überzähligen Variablen Nullstrings. Werden mehr Parameter angegeben, als Variable kodiert sind, werden die Anwenderangaben in der letzten Variablen gesammelt.

PROCESS

Werden mit dem Makroaufruf zusammen Zeilenbefehle getippt, werden sie unmittelbar ausgeführt.

NOPROCESS

Zeilenkommandos des Anwenders werden erst nach Makroende ausgeführt, oder wenn im Macro eine Instruktion PROCESS erkannt wird.

NOPROCESS muss angegeben werden, wenn das Macro Zeilenkommandos als Input benötigt. Weitere Informationen können beim PROCESS-Befehl nachgelesen werden.

Beispiele

```

"macro (member)"
if length(member) = 0 then do
  zedsmsg="Member ??"
  address ispexec "setmsg msg(isrz000)"
  return
end

```

"copy after .zlast" member

Das Macro kopiert ein angegebenes Member an das Ende des aktiven Members.

```

address isredit
"macro NOPROCESS (spalte)
if spalte = "" then "(,spalte) = display_cols
else ,
if datatype(spalte) = "CHAR" then call fehl
"process"

```

Werte sollten auf Plausibilität geprüft werden, bevor das Macro seine Arbeit beginnt.

Return Codes

0	Normales Ende
8	Keine Parameter erlaubt
20	Schwerer Fehler

MACRO_LEVEL – ISPF Edit Macro

```
►►"(variable) = MACRO_LEVEL"◄◄
```

Liefert die Hierarchiestufe, in der das Macro läuft. Mögliche Werte sind 1 bis 255, Stufe 0 ist immer der Anwender (siehe auch unter LABEL).

variable

Enthält den entsprechenden Makrolevel.

Beispiele

```
"(level) = macro_level"
```

```
vorlvl=level-1
```

```
"label .zcsr = .anfang" vorlvl
```

Das Label wird für den Makroaufrufer gesetzt, wer immer es ist (Anwender oder ein Macro).

Return Codes

0	Normales Ende
20	Schwerer Fehler

MASKLINE – ISPF Edit Macro

```
►►"(variable) = MASKLINE"◄◄
```

```
►►"MASKLINE = daten"◄◄
```

Liefert den aktuellen Inhalt der Maskenzeile, oder belegt sie neu.

=MASKLINE

Hinterlegt in der angegebenen Variablen den aktuellen Inhalt der Maskenzeile.

MASKLINE=

Sorgt für die Neubelegung der Maskenzeile.

daten

Für `daten` können unterschiedliche Angaben gemacht werden:

- einfache oder eingegrenzte Zeichenketten, oder Variable
- Schablonen (< spalte,zeichenfolge >)
- Mischformate (string1 + string2,schlüsselwort + ...)
- Schlüsselworte
 - `LINE` Daten der aktuellen Zeile
 - `LINE zz` Daten der Zeile zz
 - `MASKLINE` Daten der Maskenzeile
 - `TABSLINE` Daten der TABS-Zeile

Beispiele

```
"maskline = < 45 '/*' 70 '*/'"
```

Integriert in die Maskenzeile auf den Spalten 45/46 und 70/71 Kommentarbeginn und -ende.

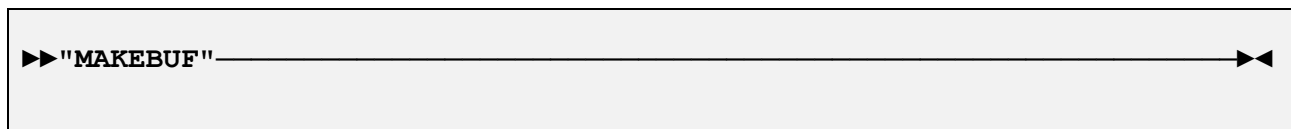
```
"maskline = ' '"
```

Die Maskenzeile ist leer.

Return Codes

0	Normales Ende
4	Daten abgeschnitten
20	Schwerer Fehler

MAKEBUF – TSO extended Command



Erzeugt einen neuen Puffer im STACK. Der standardgemäß initialisierte Puffer im STACK ist Puffer 0. MAKEBUF kann prinzipiell beliebig viele Pufferbereiche anlegen, wobei der erste durch MAKEBUF angelegte Puffer dann entsprechend Puffer 1 ist, der zweite Puffer 2 und so weiter.

Den wievielten Puffer die Instruktion MAKEBUF angelegt hat, kann im unmittelbaren Anschluss an den Befehl aus dem zurückgemeldeten Return Code (RC) abgelesen werden.

Siehe auch "QBUF" und "DROPBUF".

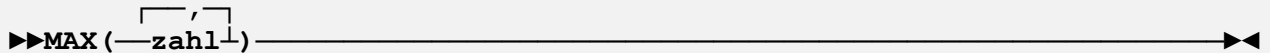
Hinweis

Mit gepufferten Stackbereichen zu arbeiten ist nicht empfehlenswert, da es eine andere Verarbeitungslogik erfordert und keinerlei Vorteile bringt.

Return Codes

- 1 Ein neuer Buffer nach dem Buffer 0 existiert jetzt im Stack
- 2 Ein zweiter Buffer existiert jetzt im Stack
- 3 Ein dritter Buffer existiert jetzt im Stack
- N Und so weiter

MAX() – REXX Built-in-Function



```
►►MAX(—zahl—)◄◄
```

Maximalwert aus einer Folge von Zahlen

Beispiele

```
say MAX(12,7,3,9)          12
say MAX(3.65,35,0.3)       35
say MAX(-6,-4.3,-74)       -4.3
```

MAXGENS – JCL DD Parameter

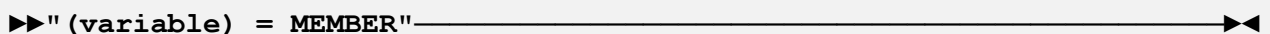


```
►►MAXGENS=maximum-generations◄◄
```

Angabe der maximalen Anzahl der Generationen einer PO/E-Typ2 Datei. Der Wert kann 0 sein, wenn die Vorzüge dieses Dateiformates genutzt werden sollen, aber keine Memberversionierung gewünscht ist.

Der höchstmögliche Wert ist installationsabhängig und kann gegebenenfalls in der Systemprogrammierung erfragt, oder über eine kleine REXX ermittelt werden.

MEMBER – ISPF Edit Macro



```
►►(variable) = MEMBER◄◄
```

Ermittelt den Namen des Members, das editiert wird. Wird eine sequentielle Datei bearbeitet, ist die Variable Blank.

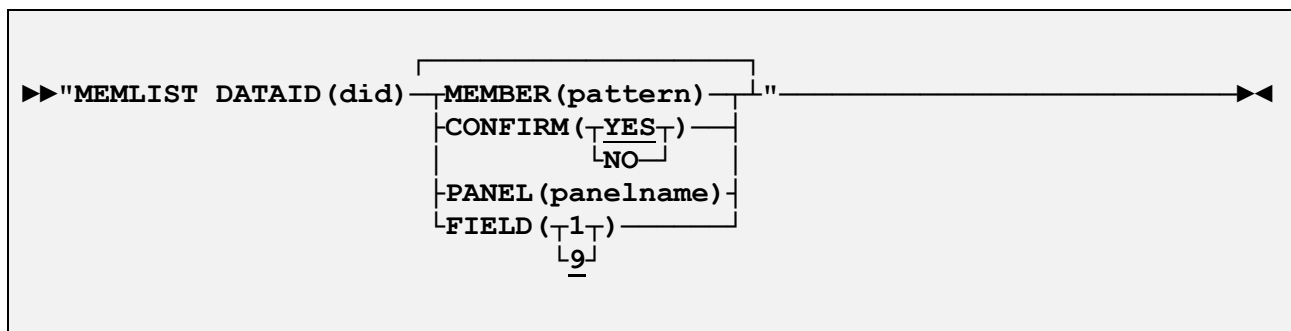
Beispiele

```
"(mbr) = member"
select
  when left(mbr,3) = "SMP" then ...
  when left(mbr,3) = "TST" then ...
  otherwise ...
end
```

Der Membername wird auf bestimmte Formalismen hin überprüft.

Return Codes

0	Normales Ende
20	Schwerer Fehler

MEMLIST – ISPF LM Service

Erlaubt den Zugriff auf die Library Utility Member List aus einem Dialog.

Der Service präsentiert eine Memberanzeige mit einem 1 bis 9 Byte langen Line-Command-Feld. Alle Library Utility Funktionen können ausgeführt werden (Edit, Browse, View, Print, Delete, Rename).

DATAID(did)

DatenID, die vom LMINIT zugewiesen wurde.

MEMBER(pattern)

Memberangabe mit Wildcards. Wird der Parameter nicht angegeben, wird das gesamte Directory angezeigt.

CONFIRM(YES|NO)

Steuert, ob beim Löschen von Members das "CONFIRM-DELETE"-Panel angezeigt wird.

PANEL(panelname)

Name des Panels, mit dem die Memberliste gezeigt werden soll. Dies kann ein selbstgeschriebenes Panel sein. Fehlt die Angabe, ist das Standardpanel in Verbindung mit FIELD(1) das ISRUDMM. Mit FIELD(9) wird defaultmäßig das Panel ISRUDSM benutzt.

FIELD(1|9)

Bestimmt die Länge des Eingabefeldes für LineCommands. Default ist 9.

Return Codes

0	Normales Ende
8	Leere Datei oder keine übereinstimmenden Member gefunden
10	Keine Datei mit der DataID verbunden (LMINIT fehlt?)
12	Möglich sind: Datei nicht PO Falscher Parameter Syntaxfehler
16	Truncation oder Translation Fehler
20	Schwerer Fehler

MEND – ISPF Edit Macro

```
"MEND"_____▶◀
```

Wird nur benötigt, wenn das CLIST/REXX-Macro unter MVS/370 (gibt es seit Jahrzehnten nicht mehr) im Batchmodus läuft. Unter z/OS wird das Statement nicht gebraucht, es stört aber auch nicht.

Beispiel

```
address isredit  
"macro"  
"change all ..."  
if rc = 0 then "save"  
else "cancel"  
"mend"  
exit
```

Läuft das Macro unter MVS370 im Batchmodus, muss Makroende mitgeteilt werden.

Return Codes

0	Normales Ende
20	Schwerer Fehler

MESSAGE – JCL JES Instruction

```
/*MESSAGE nachricht_____▶◀
```

Dieses Statement wird benutzt, um eine Nachricht an die Operatorkonsole zu übertragen, sobald JES2 den Job einliest.

Hinweis

Ist ebenfalls ein `/*ROUTE-XEQ-Statement` vorhanden und liegt das Message-Statement davor, wird die Nachricht an das Sendesystem, anderenfalls an das Empfangssystem übermittelt.

MGMTCLAS – JCL DD Parameter

```
►►MGMTCLAS=management-class◄◄
```

Diese Angabe wird nur in Verbindung mit SMS unterstützt. Ist SMS nicht installiert oder nicht aktiv, wird die Angabe ignoriert. Durch SMS wird die Angabe ignoriert, sofern es sich dabei um eine bereits katalogisierte oder nicht von SMS gemanagte Datei handelt.

Eine Managementklasse regelt für eine Datei Dinge wie Backup und Migration.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

MIN() – REXX Built-in-Function

```
►►MIN (—zahl—)◄◄
```

Minimalwert aus einer Folge von Zahlen

Beispiele

```
say MIN(12,7,3,9)      3
say MIN(3.65,35,0.3)   0.3
say MIN(-6,-4.3,-74)   -74
```

MODEL – ISPF Edit Macro

```
►►"MODEL modname AFTER zz"◄◄
    BEFORE  [NOTE]
            [NONOTE]
    CLASS classname
```

Kopiert ein Model vor oder hinter die angegebene Zeile. Werden Hinweiszeilen mit kopiert, kann im Macro nicht auf diese Bezug genommen werden.

modname

Name des Models, das kopiert werden soll (Details siehe IBM-Manual "Edit & Edit Macros").

AFTER, BEFORE

Zielortangabe (davor oder dahinter) Bezug nehmend auf den Zeilenzeiger.

zz

Zeilenzeiger in Form eines Labels oder einer relativen Zeilennummer.

NOTE | NONOTE

Erklärende Hinweise zum Model werden mit kopiert oder unterdrückt

CLASS classname

Ändert die Modelklasse auf die angegebene Klasse ab.

Beispiele

```
"model tbopen after .a nonote"
```

Ein Model für den TBOPEN-Befehl wird hinter die Zeile mit dem Label .A kopiert. Hinweiszeilen werden unterdrückt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

MOVE – ISPF Edit Macro

```
►►"MOVE member AFTER zz"
    BEFORE
```

MOVE kann ein Member einer PO-Datei in die aktuellen Daten übertragen. Das Member wird an seinem Ursprungsort gelöscht.

member

Name des Members der PO-Datei, das importiert werden soll.

AFTER | BEFORE

Zielort der zu importierenden Daten in Bezugnahme auf einen Zeilenzeiger.

zz

Zeilenzeiger (Linepointer) als Zielort. Die importierten Daten können vor oder hinter die entsprechende Zeile kopiert werden. Zeilenzeiger kann ein Label oder eine relative Zeilennummer sein.

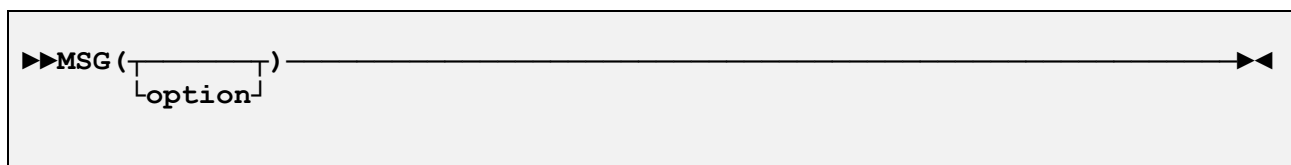
Beispiel

```
"move adr1 after .zlast"
```

Überträgt das Member ADR1 und fügt die Daten an das Ende der Datei an.

Return Codes

0	Normales Ende
8	EOF erreicht bevor der letzte Datensatz gelesen wurde
12	Einer der Gründe: Falscher Satzzeiger Member nicht gefunden BLDL Fehler
16	EOF erreicht bevor der erste Satz gelesen wurde
20	Schwerer Fehler

MSG() – TSO extended Function

Die Funktion übergibt die Zeichenfolge OFF oder ON. Mit Angabe einer Option wird das Verhalten bei der Message-Ausgabe gegebenenfalls beeinflusst.

Option

Als Option kann OFF oder ON angegeben werden. Dementsprechend erfolgt bei der Verarbeitung von TSO-Kommandos (k)eine Meldungsanzeige.

Hinweis

In vielen Fällen ist es sinnvoll, die Nachrichten von TSO-Kommandos zu unterdrücken, weil sie meist die Bildschirmaufbauten in einer Dialoganwendung stören. Werden diese Nachrichten nicht benötigt, kann der Suppress per MSG() erfolgen.

Sollen die Nachrichten im Bedarfsfall ausgewertet/angezeigt werden (im Fall fehlgeschlagener ALLOCs manchmal hilfreich), müssten die Nachrichten über OUTTRAP() eingefangen werden.

Wird die Rückmeldung von MSG() nicht ausgewertet, kann der Aufruf auch per CALL durchgeführt werden.

Beispiele

```
Oldmsg = MSG("OFF")  
Call msg "ON"
```

MSGCLASS – JCL JOB Parameter

►►MSGCLASS=ausgabeklasse◄◄

Eine Ausgabeklasse für Systemausgaben wird definiert. Mögliche Angaben sind A bis Z und 0 bis 9. Soll die Ausgabe am Terminal überprüft werden (JOBLOG-Protokoll), ist die Klasse X anzugeben.

MSGLEVEL – JCL JOB Parameter

►►MSGLEVEL=(statements,messages)◄◄

Steuert die Ausgabe des JES-JOBLOG-Protokolls.

statements

- 0 Nur die Job-Anweisung wird gelistet.
- 1 Alle JCL- und Prozeduranweisungen werden gelistet (Standard).
- 2 Alle JCL-Anweisungen des Eingabestromes werden gelistet.

messages

- 0 Nur JCL-Nachrichten (außer im Abbruchfall)
- 1 Alle JCL-, JES- und Operatornachrichten (Standard)

MVSVAR() – TSO extended Function

►►MVSVAR(varname)◄◄

Liefert Informationen aus MVS oder TSO. Folgende Variablen können ermittelt werden (installationsabhängig):

Variable	Bedeutung / Inhalt
SYSAPPCLU	APPC/MVS logical unit (LU)
SYSDFP	Version der Data Facility Products
SYSMVS	Version des MVS
SYSNAME	Name des Systems, auf dem die REXX läuft.
SYSOPSYS	Name, Version, Release und Modification-Level des z/OS
SYSSECLAB	Security Label Name der TSO-Session
SYSSMFID	SMF-ID
SYSSMS	Benennt das verfügbare SMS
SYSCLONE	Systemname
SYSPLEX	MVS-Plexname
SYMDEF	Symbolic Variables des MVS-Systems (siehe IBM-Manual)

NEWSTACK - TSO extended Command

►► "NEWSTACK" ◄◄

Mit NEWSTACK kann ein neuer STACK-Bereich definiert werden. Gleichzeitig werden alle Puffer und Elemente isoliert. An diese Elemente kann nicht mehr gelangt werden, bis der neue, aktuelle STACK-Bereich durch den Befehl DELSTACK wieder aufgelöst wird.

Nachdem NEWSTACK ausgeführt ist, werden alle Elemente, die durch die Instruktionen PUSH oder QUEUE in einen Pufferbereich gestellt werden, im neuen STACK abgelegt. Wird in einer EXEC eine Subroutine oder Funktion aufgerufen, wird von diesen (soweit nötig) ebenfalls der neue STACK benutzt. Elemente in einem hierarchisch darüber liegenden STACK-Bereich können nicht "angefasst" werden.

Prinzipiell können mehrere STACK-Bereiche definiert werden, es können jedoch nur die Elemente des jeweils zuletzt erzeugten STACK-Bereiches angesprochen werden. Um an die hierarchisch höher liegenden Elemente und Puffer zu gelangen, müssen die einzelnen STACK-Bereiche durch DELSTACK aufgelöst werden.

NONNUMBER – ISPF Edit Macro

►► "NONNUMBER" ◄◄

Ist gleichbedeutend mit NUMBER OFF

NOP – REXX Instruction



NOP ist eine sogenannte DUMMY-Instruktion (No OPeration), ohne weitere Bedeutung. In der Praxis wird NOP als Platzhalter einer THEN oder ELSE Klausel benutzt.

Beispiele

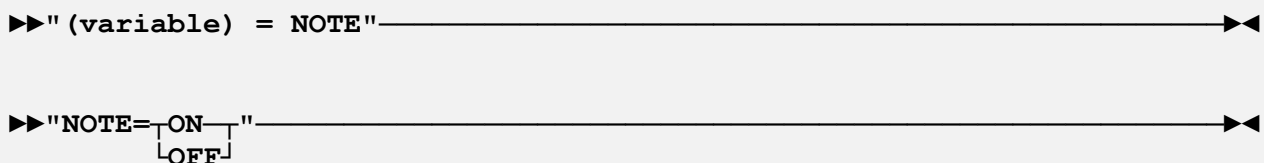
```
select
    when userid()='SP1' then nop /*keine Aktion*/
    when userid()='SP2' then say 'Hallo Alex'
    otherwise say 'Hallo Stranger'
end
```

Würde hier die NOP-Anweisung weggelassen werden, würde die nachfolgende WHEN-Anweisung als erste Instruktion nach THEN interpretiert und zu einem Syntaxfehler führen. NOP ist eine wirkliche Instruktion (auch wenn definitiv nichts getan wird) und in diesem Fall hinter THEN zwingend erforderlich.

```
:
trace ?R
nop
exit
```

Soll vor Ende des Programms der Trace aktiviert werden um noch Werte von Variablen abfragen zu können, wird der Trace erst zusammen mit dem nächsten Befehl aktiv. Wenn dieser Befehl nun leider EXIT ist, ist das Programm zu Ende, ohne dass der Trace gestartet wurde.

NOTE – ISPF Edit Macro



Übergibt oder setzt den Profileintrag für NOTE. Der Profileintrag wird berücksichtigt, wenn über den MODEL-Befehl die Syntaxbeschreibung für einen Dialog Management Service angefordert wird.

=NOTE

Der aktuelle Wert von NOTE wird in einer Variablen hinterlegt.

NOTE=

Der NOTE-Modus wird entsprechend im Edit Profil hinterlegt.

ON

Hinweiszeilen werden zusammen mit dem angeforderten MODEL eingefügt.

OFF

Die erklärenden Texte werden unterdrückt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

NOTIFY – JCL JOB Parameter

```
▶▶NOTIFY=userid
```

userid

ist die Benutzerkennung eines TSO-Benutzers, der bei Jobende benachrichtigt werden soll.

Hinweis

Wird trotz NOTIFY-Angabe in der JOB-Anweisung keine Nachricht über Job-Ende empfangen, liegt dies in der Regel an den Einstellungen des TSO-Benutzerprofiles.

Stellt sicher, dass das Profil auf INTERCOM eingestellt ist.

NULLS – ISPF Edit Macro

```
▶▶"(var1,var2) = NULLS"
```

```
▶▶"NULLS = [ON  
            [OFF  
            [ALL  
            [STD
```

Setzt oder übergibt den aktuellen Profilwert für NULLS.

var1

Enthält die aktuelle Schalterstellung (ON oder OFF).

var2

Zeigt auf ALL, STD oder Blank.

=NULLS

Der aktuelle Wert von NULLS wird in einer Variablen hinterlegt.

NULLS=

Der NULLS-Modus wird entsprechend im Edit Profil hinterlegt. Möglich sind:

ON

Nachlaufende Blanks in einem Feld werden als 00hex ausgegeben.

OFF

Nachlaufende Blanks in einem Feld werden als Blanks geschrieben.

ALL

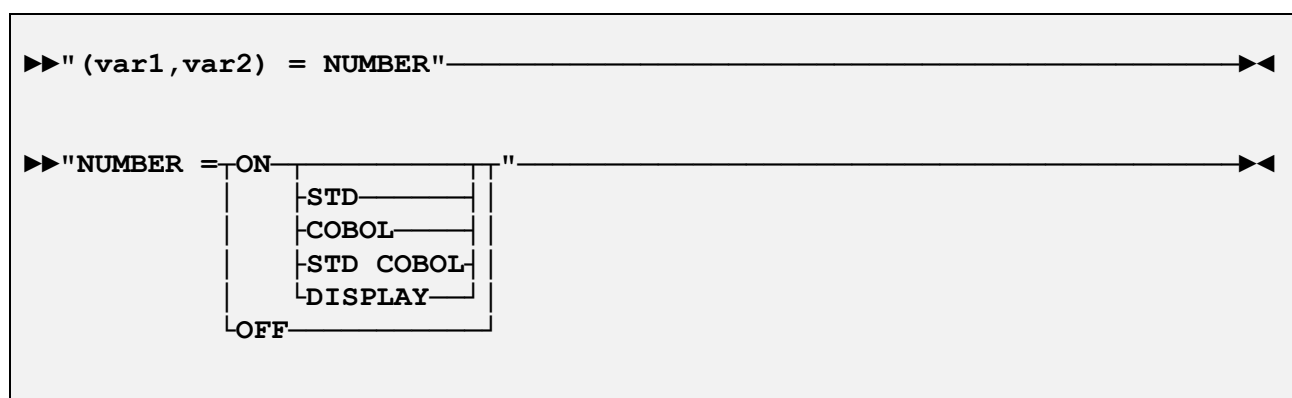
Nachlaufende Blanks in einem Feld und Blank-Felder werden geschrieben.

STD

Ein nachlaufendes Blank bleibt erhalten.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

NUMBER – ISPF Edit Macro

Setzt oder übergibt den aktuellen Wert des NUMBER-Modus. Der Nummernmodus steuert die automatische Generierung von Zeilensequenznummern.

var1

Enthält die Schalterstellung (ON oder OFF).

var2

Enthält eine der folgenden Kombinationen:

NONSTD	NOCOBOL	DISPLAY
STD	NOCOBOL	DISPLAY
NOSTD	COBOL	DISPLAY
STD	COBOL	DISPLAY
NONSTD	NOCOBOL	NODISPLAY
STD	NOCOBOL	NODISPLAY
NOSTD	COBOL	NODISPLAY
STD	COBOL	NODISPLAY

Die Werte STD, COBOL und DISPLAY können in var2 auch enthalten sein, wenn der Nummernmodus auf OFF zeigt. Entsprechend werden diese Angaben wieder als Defaultwerte benutzt, wenn der Nummernmodus wieder eingeschaltet wird.

=NUMBER

Der aktuelle Wert von NUMBER wird in einer Variablen hinterlegt.

NUMBER=

Der NUMBER-Modus wird entsprechend im Edit Profil hinterlegt.

ON

Prüft alle Zeilen auf gültige Sequenznummern und führt zur Neunummerierung, wenn durch Einfügungen die Zeilennummern außerhalb der Sequenz liegen würden. Die Operanden STD, COBOL und DISPLAY werden nur interpretiert, wenn der Nummernmodus eingeschaltet ist.

OFF

Der Nummernmodus ist ausgeschaltet.

STD

Die Sequenznummern werden im Standardfeld (Satzende) hinterlegt. Dabei enthalten die ersten sechs Byte die Zeilennummer, die letzten beiden Byte zeigen den Update-Level. Dies entspricht der Nummerierung aller nicht-COBOL-Daten.

COBOL

Die Sequenznummer ist in den ersten sechs Byte des Datensatzes hinterlegt. Diese Form entspricht allen Dateien mit LAST QUALIFIER "COBOL". COBOL kann COB abgekürzt werden.

STD COBOL

Die Sequenznummern werden auf beide Arten eingetragen.

DISPLAY

Das Datenfenster schließt die Sequenznummern ein, oder nicht. Wird eine 80-stellige Datei mit STD-Nummerierung geführt, sind die Sequenznummern erst sichtbar, wenn eine Rechtsverschiebung veranlasst wird. DISPLAY kann mit DIS abgekürzt werden.

Beispiele

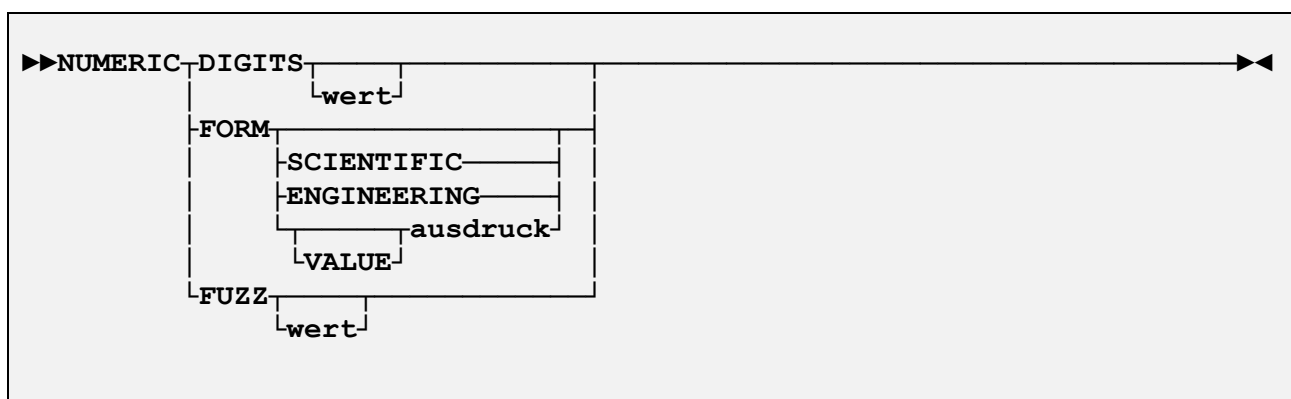
```
"(modus,params) = number"
"number off"
:
```

Ermittelt den IST-Zustand für NUMBER, setzt den Modus auf OFF und stellt den alten Zustand wieder her.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

NUMERIC – REXX Instruction



NUMERIC wird benötigt, um die standardgemäße Auflösung arithmetischer Operationen zu verändern (Details siehe TSO/E REXX Reference).

NUMERIC DIGITS

Steuert die Rechengenauigkeit (Default ist 9). Das Ergebnis wird auf die Anzahl Nachkommastellen-1 gerundet. Der Wert muss eine integere positive Zahl und größer als der Wert in NUMERIC FUZZ sein.

NUMERIC FORM

Form der Exponentialdarstellung in REXX. Folgt keine weitere Angabe, wird SCIENTIFIC (wissenschaftlich) angenommen, was bewirkt, dass bei arithmetischen Operationen oder Funktionen nur eine Vorkommastelle ungleich Null ausgegeben wird.

NUMERIC FUZZ

Rechengenauigkeit (wie viele Nachkommastellen werden bei einem numerischen Vergleich ignoriert - Default = 0). Anderenfalls wird entsprechend gerundet. Mögliche Angaben sind Werte zwischen 0 und einer Zahl, kleiner des Werts NUMERIC DIGITS.

Beispiel

```
/* REXX ** Division *****/
NUMERIC DIGITS 3
SAY 'Dezimal: 23:7=' 23 / 7
SAY 'Integer: 23:7=' 23%7 'Rest' 23 // 7
EXIT
```

Ablauf:

```
Dezimal: 23 : 7 = 3.29
Integer: 23 : 7 = 3 Rest 2
```

OPEN – DB2 Instruction

►►"OPEN cursor "◄◄

Der OPEN cursor wird benutzt um eine Zeilenweise Verarbeitung von Sätzen der Ergebnistabelle mittels FETCH-Anweisung abarbeiten zu können.

Cursor

Name des Cursors, wie er in der DECLARE-Anweisung angegeben wurde.

OUTPUT – JCL Special Statement

```
//label OUTPUT CKPTLINE=zeilen
               CKPTPAGE=seiten
               CKPTSEC=sekunden
               COPIES=anzahl
               DEST=printername
               LINECT=lines
               TITLE=titel
```

Das OUTPUT-Statement liefert eine Vielzahl von Möglichkeiten auf eine Druckausgabe einzuwirken. Nachfolgend die Gängigsten:

**CKPTLINE=
CKPTPAGE=**

Bestimmt, nach wie vielen (logischen) Seiten zu entsprechend vielen Zeilen ein Checkpoint in der Druckausgabe zu setzen ist. Auf diese Checkpoints kann durch entsprechende JES2-Operatorkommandos aufgesetzt werden.

CKPTSEC=

Legt die Zeit in Sekunden fest, nach der in einer laufenden Druckausgabe ein Checkpoint geschrieben wird (gängiger ist eine Mengenangabe über CKPTPAGE/CKPTLINES).

COPIES=

Liefert die Anzahl der Gesamtdruckausgaben, die von einer Ausgabe erstellt werden soll (siehe auch unter SYSOUT-DD-Angabe).

DEST=

Benennt den (lokalen) Drucker, über welchen die Druckausgabe abgewickelt werden soll (siehe auch SYSOUT-DD-Angabe).

LINECT=

Regelt, nach wie vielen Druckzeilen je Seite ein Seitenwechsel erfolgen soll (siehe auch JES2-Command /*JOBPARM).

TITLE=

Legt eine Titelzeile fest, die bei Druckausgabe auf das Trennblatt geschrieben wird.

Beispiele

```
//RPTDS OUTPUT COPIES=4
```

Es werden vier Exemplare des Outputs erstellt.

```
// OUTPUT COPIES=(, (3))
```

Die Druckausgabe erfolgt dreimal aber nicht sortiert hintereinander, sondern dreimal Seite1, dreimal Seite2, dreimal Seite3 und so weiter.

```
//LOCAL OUTPUT DEST=PRT44
```

Die Druckausgabe wird auf den Abteilungsdrucker PRT44 umgeleitet.

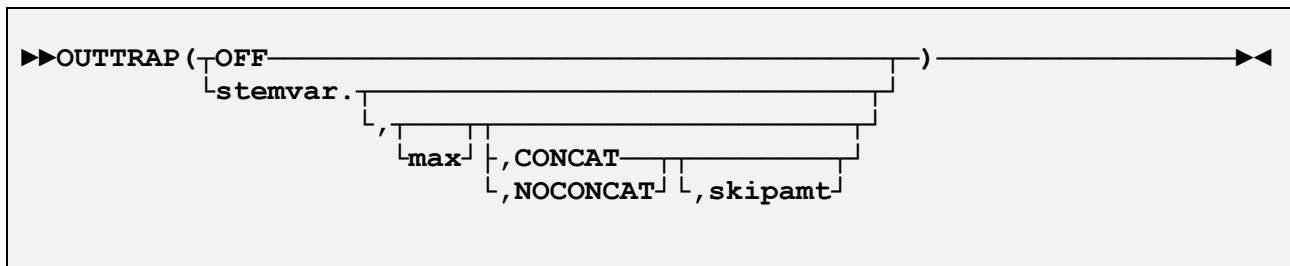
```
//PRINT OUTPUT LINECT=55
```

Seitenwechsel erfolgt abweichend vom Standard nach spätestens 55 gedruckten Zeilen je Seite.

```
//AUSGABE OUTPUT TITLE='Geheim Report'
```

In der dafür vorgesehenen Zeile auf dem Trennblatt der Druckausgabe wird der Text Geheim Report ausgegeben.

OUTTRAP() – TSO extended Function



Steuert die Umleitung von Command-Output.

OFF

Die Umleitung von Bildschirmmeldungen in Variable wird ausgeschaltet.

Stemvar.

Stem der den Command-Output aufnimmt.

max

Die maximale Zahl umzuleitender Ausgabezeilen (Default = 999.999.999).

CONCAT

Der Output von TSO-Kommandos wird umgeleitet, bis die maximale Anzahl der Ausgabezeilen erreicht ist. Erzeugt der erste Befehl drei Ausgabezeilen, werden diese in Variablen .1, .2 und .3 überstellt. Gibt der nächste Befehl zwei Ausgabezeilen aus, stehen diese in Variablen .4 und .5 (Default)

NOCONCAT

Der Stem wird bei jeder Befehlsausführung überschrieben.

Wird ein Unterprogramm aufgerufen und wurde im Hauptprogram OUTTRAP() gesetzt, so gilt dies auch nach unten, solange, bis das erste Programm beendet oder die Umleitung aufgehoben wird.

skipamt

Anzahl der Zeilen, welche übersprungen werden sollen, bevor das Trapping beginnt.

Zusätzliche Variable

Variable	Inhalt
stem.0=	Größter Index ausgegebener Zeilen
Stem.MAX	Anzahl Zeilen, die umgeleitet werden können
Stem.TRAPPED	Gesamtzahl aller umgeleiteten Zeilen
Stem.CON	Wert den CONCAT-Operanden

Beispiele

```
DMY = OUTTRAP ("ABC. " , 4 , "CONCAT")
```

Der erste Befehl erzeugt Ausgabezeilen

Variable	Inhalt
ABC.0=	3
ABC.1	Befehl 1, Ausgabezeile 1
ABC.2	Befehl 1, Ausgabezeile 2
ABC.3	Befehl 1, Ausgabezeile 3
ABC.4	ABC.4
ABC.MAX	4
ABC.TRAPPED	3
ABC.CON	CONCAT

Der zweite Befehl erzeugt zwei Ausgabezeilen, von denen die zweite Zeile nicht mehr umgeleitet wird */

Variable	Inhalt
ABC.0=	3
ABC.1	Befehl 1, Ausgabezeile 1
ABC.2	Befehl 1, Ausgabezeile 2
ABC.3	Befehl 1, Ausgabezeile 3
ABC.4	Befehl 2, Ausgabezeile 1
ABC.MAX	4
ABC.TRAPPED	5
ABC.CON	CONCAT

DMY = OUTTRAP("ausgabe.")

Leitet alle Ausgaben aller TSO-Kommandos zusammenhängend in den Variablenstamm "AUSGABE." um.

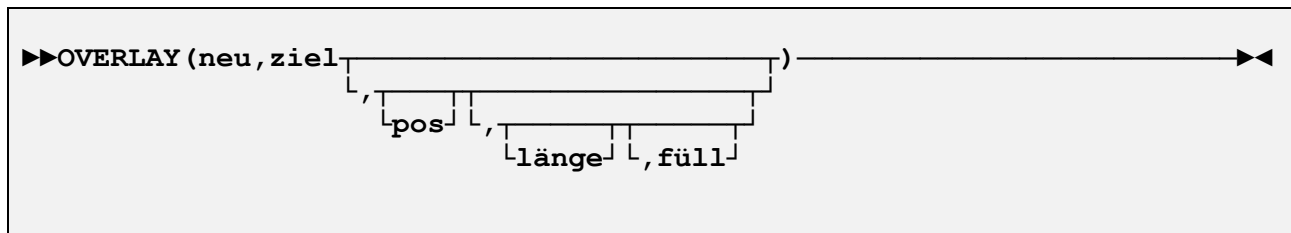
DMY = OUTTRAP("temp",0)

Unterdrückt die Ausgabe auf TSO-Kommandos hin.

Es soll eine Datei neu angelegt werden. Ist dies erfolgreich, soll die alte Datei gelöscht, ansonsten sollen die umgeleiteten Meldungen ausgegeben werden. Hier ein Beispiel, wie es auch mit CALL funktioniert (wenn das unmittelbare Funktionsergebnis nicht verarbeitet werden soll/muss):

```
CALL MSG "OFF"
CALL OUTTRAP "var."
"ALLOC DA(neue.datei) LIKE(alte.datei) NEW"
IF RC = 0 THEN "DELETE alte.datei"
ELSE,
  DO $ = 1 TO VAR.0
    SAY '**ERROR**' VAR.$
  END
```

OVERLAY() – REXX Built-in-Function

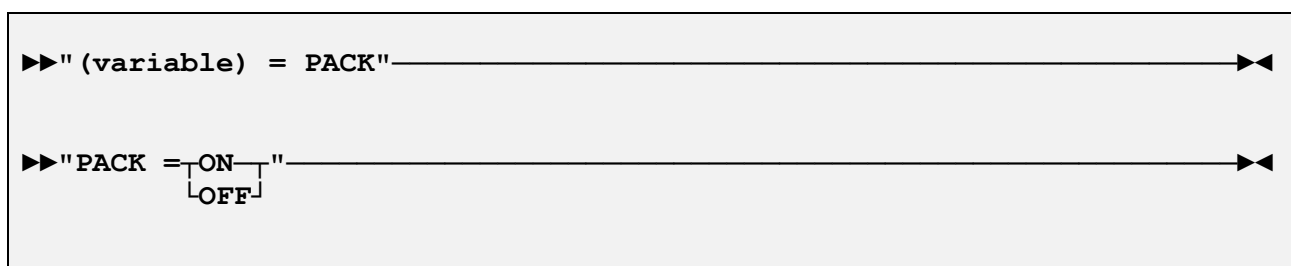


Überlagert die Ziel-Zeichenfolge an der angegebenen Position (ohne Angabe der Position wird 1 angenommen) mit der neuen Zeichenfolge. Wird eine Länge angegeben, die größer als die Länge der Ziel-Zeichenfolge ist, wird entsprechend mit dem Füllzeichen aufgefüllt.

Beispiele

<code>say OVERLAY('d', 'mein')</code>	<code>'dein'</code>
<code>say OVERLAY('.', 'abcde', 3, 2)</code>	<code>'ab. e'</code>
<code>say OVERLAY('c', 'abbde', 3)</code>	<code>'abcde'</code>
<code>say OVERLAY('fgh', 'abcde', 6)</code>	<code>'abcdefgh'</code>
<code>say OVERLAY('123', 'abc', 5, 6, '-')</code>	<code>'abc--123---</code>
<code>say OVERLAY('G', 'Meier', 1)</code>	<code>'Geier'</code>

PACK – ISPF Edit Macro



Übergibt oder setzt den PACK-Modus des Profils.

=PACK

Der aktuelle Wert von PACK wird in einer Variablen hinterlegt.

PACK=

Der PACK-Modus wird entsprechend im Edit Profil hinterlegt und steuert, ob Daten im gepackten Format gespeichert werden sollen, oder nicht (nicht empfehlenswert – wird PACK ON durchgeführt, kann nur noch der Editor die Daten lesen).

Return Codes

0	Normales Ende
20	Schwerer Fehler

PARM – JCL EXEC Parameter

►►PARM=wert

Übergibt Steuerungsinformationen an ein Programm. Der Wert darf maximal 100 Zeichen lang sein. Wenn ein PARM-Wert aus Unterparametern besteht, müssen Klammern oder Hochkomma benutzt werden. Enthält ein Wert Sonderzeichen, muss der Wert zwischen Hochkommata stehen.

Beispiele

PARM=TEST

Übergeben wir ein Parameter ohne Sonderzeichen

PARM= ' TEST ,NOLIST '

Übergeben wird ein Parameter mit Sonderzeichen (das Komma).

PARM= ' TEST ,FLOW=20 '

Übergeben wird ein Parameter mit Sonderzeichen (das Komma und das Gleichheitszeichen).

PARM= (TEST ,NOMSG)

Übergeben werden zwei Parameter ohne Sonderzeichen.

PARM= (TEST , ' BUFFERS=20 ' ,NOLIST)

Übergeben werden drei Parameter, von denen der zweite Sonderzeichen enthält.

PARMDD – JCL EXEC Parameter

►►PARMDD=ddname

Erlaubt das Lesen von Parametern aus einer Datei des angegebenen DD-Namens. Das Format muss F, FB, V oder VB sein. Die maximale Satzlänge sind 32KB. (Kommt Ihr Programm damit klar?)

PARSE – REXX Instruction

►►PARSE UPPER ARG EXTERNAL PULL VALUE ausdruck WITH VAR variable SOURCE zerlegemuster

PARSE erlaubt die Zuordnung von Daten aus unterschiedlichen Quellen zu einer oder mehreren Variablen nach den Regeln für PARSING.

PARSE ist einer der mächtigsten Befehle in REXX. Im Band-1 (Kochbuch) sind dem PARSE mehrere Seiten und Beispiele gewidmet. Es lohnt sich 😊

zerlegemuster

Ist eine durch BLANK voneinander getrennte Folge von Worten im Sinne von REXX (Variable). Ist keine Wortfolge angegeben, werden keine Variablen gefüllt, die Aktion aber wird soweit als möglich ausgeführt (durch PARSE PULL wird ein Satz aus dem STACK entladen). Möglich sind auch Trennzeichen und/oder Bytepositionen. Werden Trennzeichen über Variablen definiert, müssen diese in Klammern stehen.

UPPER

Setzt bei der Zuweisung der Variablen die Inhalte auf Upper-Cases um.

ARG

Die Variable(n) werden an die Prozedur als Eingabe übergeben (siehe auch ARG-Instruktion und ARG-Built-in-Funktion).

EXTERNAL

Liest immer von der Tastatur. Sind Eingaben vom Benutzer erforderlich, ist dies empfehlenswert, da PULL sonst aus dem STACK-Bereich liest, sofern dort Daten vorhanden sind. Wird PARSE EXTERNAL in einer Prozedur im BATCH benutzt, wird von SYSTSIN gelesen.

PULL

Eine Zeile der Queue (STACK) wird gelesen. Ist die Queue leer, wird von der Tastatur gelesen. Die Anzahl der Datensätze, die sich aktuell im STACK-Bereich befinden, kann durch die Funktion QUEUED() ermittelt werden (Schleifenverarbeitung).

VALUE

Muss benutzt werden, wenn die zu zerlegende Information ein Funktionsergebnis ist.

```
PARSE VALUE time() WITH std ':' min ':' sec
PARSE VALUE date() WITH tag monat jahr
```

SOURCE

Informationen über das Quellenprogramm werden ausgelesen. Die Informationen liegen als Wortkette vor. Die einzelnen Worte enthalten der Reihe nach:

- Die Zeichenfolge TSO
- Typ: COMMAND, FUNCTION oder SUBROUTINE
- Name des laufenden Programmes
- DD-Name der Datei, von der geladen wurde oder ?
- Name der Ladedatei oder ?
- Übergeordnetes Programm oder ?

- Environment, aus dem der Aufruf erfolgte
- Name des Adressraumes

```
PARSE SOURCE . TYP NAME .
```

VAR

Zerlegt wird der Inhalt einer Variablen. Dies kann über eine Wortkette oder über Bytepositionen erfolgen.

```
PARSE VAR "AUGUSTIN" 1 V1 3 V2 7 .
```

```
V1= "AU"
```

```
V2="GUST"
```

```
Pos1 = pos("°",var)
```

```
Pos2 = pos("^",var)
```

```
PARSE var . =(pos1) feld1 =(pos2) .
```

Hinweis

PARSE ist einer der mächtigsten Befehle in REXX. Hier lohnt sich ein Blick in den REXX Users Guide. Dem Parse-Befehl sind mehrere Seiten gewidmet.

PASSWORD – JCL JOB Parameter

```
►►PASSWORD=password◄◄
```

Übergibt ein RACF-Kennwort.

PASTE – ISPF Edit Macro

```
►►"PASTE" [bereich] [AFTER | BEFORE] [zeilennummer | label] [DELETE | KEEP] "◄◄
```

Kopiert oder überträgt Daten von der Zwischenablage in eine Edit Session.

Bereich

Name des Zwischenbereiches. Wurde bei CUT kein Name angegeben, wurden die Daten nach DEFAULT gestellt.

AFTER | BEFORE

Die Daten werden hinter oder vor einen Bezugspunkt kopiert oder übertragen.

Zeilennummer | Label

Der Bezugspunkt für das Kopieren oder Übertragen kann eine relative Zeilennummer oder ein Label sein.

DELETE | KEEP

Bestimmt, ob übertragen (Move) oder kopiert wird. Mit KEEP bleiben die Daten im Zwischenbereich erhalten. Durch DELETE wird die Zwischenablage nach dem Kopiervorgang gelöscht.

Return Codes

0	Normales Ende
12	Clipboard ist leer oder existiert nicht
20	Schwerer Fehler

PATH – JCL DD Parameter

▶▶PATH=pfadname◀◀

Definiert ein UNIX file.

pfadname

Dateiname in einem z/OS UNIX file system. Die Angabe ist case-sensitiv.

PATHDISP – JCL DD Parameter

▶▶PATHDISP=(KEEP,KEEP)
 [DELETE][DELETE]◀◀

Regelt, was mit einem UNIX File bei normalem (Parameter-1) oder abnormalem Ende (Parameter-2) passieren soll

KEEP

Die Datei bleibt bestehen

DELETE

Die Datei wird gelöscht

PATHMODE – JCL DD Parameter

►►PATHMODE=(zugriffs-attribut)

Bedingung für diese Angabe ist die Kodierung neben PATHOPTS=OCREAT.

Es gibt 14 verschiedene Zugriffsattribute. Lest diese bitte im Manual z/OS UNIX System Services User's Guide nach.

PATHOPTS – JCL DD Parameter

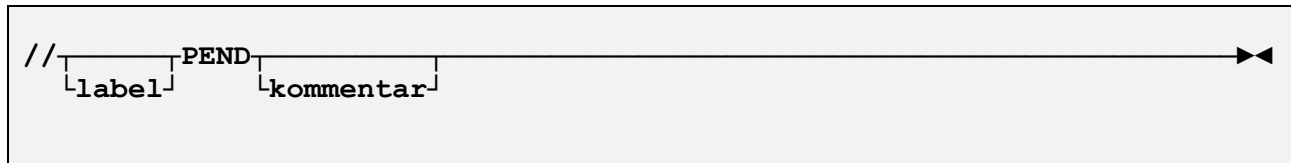
►►PATHOPTS=(ORDONLY
OWRONLY
ORDWR
OAPPEND
OCREAT
OSYNC
OTRUNC)

Bestimme den Zugriffsstatus des unter PATH= angegebenen UNIX Files. Hier wird zwischen der Access-Group und der Status-Group differenziert.

Access-Group	Bedeutung
ORDONLY	Öffnet zum Lesen
OWRONLY	Öffnet zum Schreiben
ORDWR	Öffnet für Update
Status-Group	Bedeutung
OAPPEND	Datei wird fortgeschrieben
OCREAT	Datei wird angelegt
OEXCL	Wenn nicht vorhanden, wird die Datei angelegt
OSYNC	Die Übertragung von Speicher in das File muss abgeschlossen sein, bevor der Service die Steuerung zurück gibt.
OTRUNC	Unter folgenden Bedingungen wird die Größe des Files auf NULL gesetzt: Die Datei unter PATH= existiert bereits Es handelt sich um ein "regular file" (kein Directory o.ä.). Der OPEN mit ORDWR oder OWRONLY war erfolgreich

Beispiel

```
//name DD PATH='/home/sp1/data/demo.txt',
//      PATHDISP=(KEEP,DELETE),
//      PATHOPTS=(OWRONLY,OCREAT,OEXCL),
//      PATHMODE=(SIRWXU,SIRGRP)
```

PEND – JCL Special Statement


```
// [label] PEND [kommentar]
```

Mit der PEND-Anweisung wird eine Instream-Prozedur abgeschlossen. Die PEND-Anweisung ist nur für Instream-Prozeduren vorgesehen und darf in katalogisierten Prozeduren nicht angegeben werden.

Beispiel

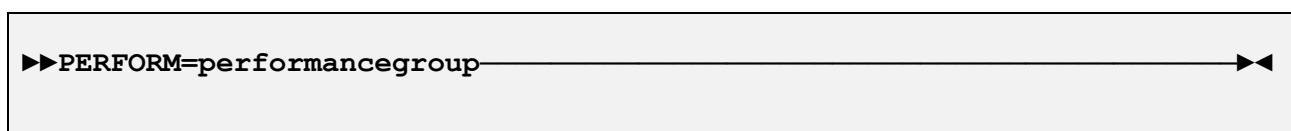
Beispiel für eine katalogisierte Prozedur zur Druckausgabe beliebiger Dateien:

Member einer JCL-PROCLIB:

```
//DRUCK      PROC  SYSOUT='A',DSN=,DISP='OLD'
//PRINT      EXEC  PGM=IEBGNER
//SYSUT1     DD    DSN=&DSN,DISP=&DISP
//SYSUT2     DD    SYSOUT=&SYSOUT
//SYSIN      DD    DUMMY
//SYSPRINT   DD    DUMMY
```

Startaufruf in der JCL:

```
//PRT        EXEC  DRUCK,DSN='WITTEMA.PDF.DATA(ADR1)'
```

PERFORM – JCL JOB Parameter


```
>>PERFORM=performancegroup
```

Perform erlaubt die Angabe einer Gruppennummer, welche die "Systemleistung" definiert. Die Performancegruppe ist ein Sammelsurium aus Dispatch-Priorities und ähnlichem.

Welche Performancegruppe angegeben werden kann und was sich im Detail dahinter verbirgt, ist installationsabhängig und kann nur bei der jeweils zuständigen Stelle der Systemprogrammierung des Unternehmens erfragt werden.

Hinweis

Heute werden derartige Definitionen im Allgemeinen über Angaben im Workload-Manager (WLM) erreicht. Auch hierzu kann die zuständige Systemabteilung Auskunft geben.

POS() – REXX Built-in-Function

```

>>POS(suchbegriff,string[,suchbeginn])

```

Übergibt die Position des Suchbegriffes innerhalb eines Strings, soweit er vorhanden ist. Wenn kein Suchbeginn angegeben ist, wird ab Stelle 1 des Strings gesucht (siehe auch INDEX()).

Beispiele

```

say POS('tag','Freitag')      5
say POS(' ','Freitag')       0
say POS(' ','So ist das')    3
say POS(' ','So ist das',4)   7

```

PQUERY – ISPF DM Instruction

```

>>"PQUERY PANEL(pname) AREANAME(area-name) AREATYPE(area-type-var)
    WIDTH(area-width-var) DEPTH(area-depth-var) ROW(row-number-name)
    COLUMN(col-number-name)

```

PQUERY liefert Informationen über eine bestimmte Area eines angegebenen Panels. Die entsprechenden Informationen liegen in den in Klammern angegebenen Variablen.

PANEL(pname)

Benennt das Panel, über welches die Informationen eingeholt werden sollen.

AREANAME(area-name)

Name der AREA, deren Attribute abgefragt werden sollen

AREATYPE(area-type-var)

Liefert den AREA-Typ (GRAPHIC oder DYNAMIC).

WIDTH(area-width-var)

Liefert die Spaltenbreite der AREA.

DEPTH(area-depth-var)

Liefert die Anzahl der Zeilen. Bei EXTEND(OFF) ist dies die Anzahl Zeilen in der AREA. Bei EXTEND(ON) ist dies die Anzahl der Zeilen, welche die AREA groß ist, nachdem der BODY sich erweitert hat. Dies ist abhängig von der Bildschirmgröße, in der das PQUERY-Kommando abgesetzt wird.

ROW(row-number-name)

Liefert die Nummer der obersten Zeile der Area.

COLUMN(col-number-name)

Liefert die Nummer der äußerst linken Spalte der Area.

Return Codes

0	Normales Ende
8	Das Panel enthält die angegebene AREA nicht
12	Panel nicht gefunden
16	Speicherüberlauf. Nicht alle Werte wurden zurückgegeben
20	Schwerer Fehler

PREPARE – DB2 Instruction

```
►►"PREPARE name FROM :host-var"◄◄
```

Die PREPARE-Anweisung erzeugt aus einem String ein ausführbares SQL-Statement. DB2-technisch gesehen wird zwischen einem "Character-String" (der Inhalt einer REXX-Variablen) und einem "Prepared-Statement" unterschieden

Name

Statementname (z.B.: S1 für das erste Statement)

:host-var

Name der REXX-Variablen, in welcher das Statement hinterlegt ist.

PRESERVE – ISPF Edit Macro

```
►►"(variable) = PRESERVE"—————►◄  
  
►►"PRESERVE =ON—"—————►◄  
          └OFF┐
```

Ermöglicht oder verhindert die Speicherung von nachlaufenden Blanks bei Sätzen variabler Länge.

=PRESERVE

Die Variable enthält den aktuellen Zustand des Preserve.

PRESERVE [=]

Legt den Zustand für PRESERVE fest. Möglich sind die Schreibweisen mit und ohne das Gleichheitszeichen.

Beispiel

```
"PRESERVE OFF"
```

Am Zeilenende stehende Blanks werden entfernt, bevor die Daten gesichert werden.

Return Codes

0	Normales Ende
6	Satzformat nicht variabel
16	Fehler beim Setzen der Variablen
20	Schwerer Fehler

PRIORITY – JCL JES Instruction

```
/*PRIORITY prio—————►◄
```

Das Priority-Statement kann benutzt werden, um einem Job, von den Standards abweichend, eine höhere Auswahlpriorität zuzuweisen. Mögliche Angaben liegen zwischen 0 (Minimum) und 15 (Maximum). Diese Angabe überschreibt eine gegebenenfalls gemachte PRTY-Angabe in der JOB-Anweisung.

Hinweis

In den Spool-Ladeparametern kann dafür Sorge getragen werden, dass eine von außen gelieferte Prioritätsangabe bei Einlesen des Jobs ignoriert wird.

PROC – JCL Special Statement

```
// [label] PROC [parms] [kommentar] →
```

PROC

Mit der PROC-Anweisung wird eine katalogisierte oder Instream- Prozedur begonnen. Bei katalogisierten Prozeduren kann die PROC- Anweisung entfallen, wenn keine symbolischen Parameter definiert werden.

Label

Bei der Instream-Prozedur muss ein Name angegeben werden, unter dem dann die Prozedur aufgerufen wird.

Operanden

Als Operanden werden symbolische Parameter definiert, die wahlweise mit Werten vorbesetzt werden können. Die symbolischen Parameter werden in der PROC-Anweisung ohne Ampersand(&) geschrieben und müssen mit dem Gleichheitszeichen(=) abgeschlossen werden.

Wenn einem Parameter weder ein Standardwert zugeordnet wurde, noch beim Prozeduraufruf ein Wert zugewiesen wird, wird er als Leerzeichenfolge behandelt.

parms

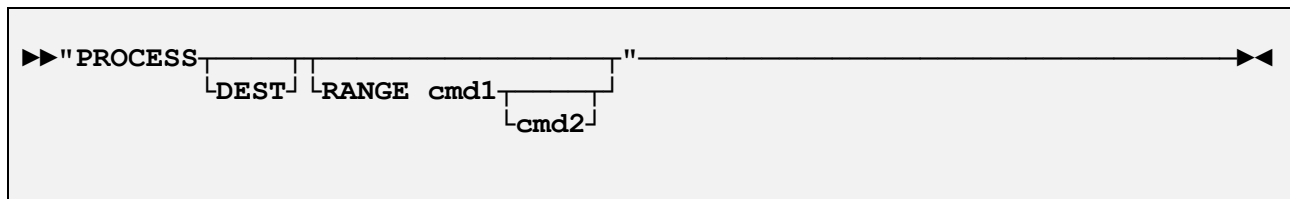
Hier können Grundzuweisungen für Variablen erfolgen, welche Gültigkeit behalten, sofern sie beim Aufruf der Prozedur nicht entsprechend versorgt werden.

Hinweis

Folgende Zeichenfolgen dürfen nicht als symbolische Parameter definiert werden, da sie vom System reserviert sind:

ACCT	JCLERR	REGION
ADDRSPC	PARM	ROLL
COND	PERFORM	SPARM
DATE	PGM	TIME
DYNAMBR	PROC	
DPRTY	RD	

PROCESS – ISPF Edit Macro



Erlaubt dem Macro den Zeitpunkt zur Ausführung von Zeilenbefehlen und Datenveränderungen von der Tastatur zu steuern. Wird eine Zeile angesteuert, bevor das PROCESS-Kommando ausgeführt wurde, ist der durch den Bediener veränderte Inhalt nicht zu erkennen (siehe auch NOPROCESS-Schlüsselwort im MACRO-Befehl).

DEST

Angabe eines Zielortes in Form des Zeilenkommandos A oder B. Die relative Nummer der Zeile wird in der Systemvariablen .ZDEST hinterlegt.

RANGE

wird gefolgt von den Namen von maximal zwei Linecommands, die der Anwender eingeben kann. Über RANGE_CMD kann erkannt werden, welches Kommando getippt wurde. Die Eingabe kann in einfacher Form, mit einer Zahl, oder als Blockbefehl erfolgen. Die Systemvariablen .ZFRANGE und .ZLRANGE enthalten die relative Nummer der ersten und letzten Zeile des Bereiches (kann die gleiche Zeilennummer sein).

cmd1 cmd2

Die Befehle können maximal sechsstellig sein. Sinnvoll ist aber eine Maximallänge von fünf Byte, da die Nutzung als Blockbefehl anderenfalls nicht möglich ist. Der Name kann aus Buchstaben und Sonderzeichen außer Minus(-) und Apostroph(') bestehen.

Beispiel

```

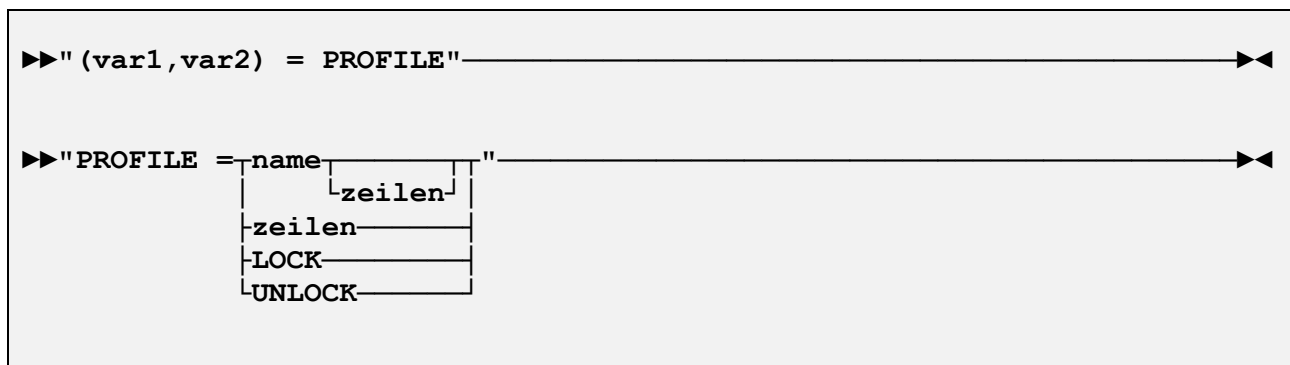
address ispxec "control errors return"
"macro NOPROCESS"
"process range $"
if rc > 16 then do
    zedsmg="kurze nachricht"
    zedlmsg="lange nachricht"
    address ispxec "setmsg msg(isrz000)"
    return
end
"(first) = linenum .zfrange"
"(last) = linenum .zlrange"
do cnt=first to last
    "(dataline) = line" cnt
    /* Verarbeitung aller Zeilen */
end

```

Der Bediener kann mit dem Aufruf des Makros einen Zeilenbereich durch die Angaben \$, \$n oder \$\$/\$\$ festlegen, die entsprechenden Zeilen werden vom Macro verarbeitet.

Return Codes

0	Normales Ende
4	Bereichsangabe im Macro vorgesehen aber nicht angegeben
8	Zielangabe im Macro vorgesehen aber nicht angegeben
12	Bereichs- und Zielangabe vorgesehen aber nicht angegeben
16	Unvollständige/widersprüchliche Zeilenbefehle angegeben
20	Schwerer Fehler

PROFILE – ISPF Edit Macro

Zeigt das aktuelle Profil, zieht ein anderes Profil an oder erzeugt ein neues, sperrt das Profil oder gibt es frei.

=PROFILE

Namen des Profils und den Hinweis LOCK oder UNLOCK,

PROFILE=

Entspricht der Eingabe des Anwenders.

name

Sucht in der Edit-PROFILE-Tabelle. Wird ein Eintrag gefunden, wird dieses neue Profil gültig. Wenn nicht, wird ein neues Profil erzeugt.

zeilen

Anzahl Zeilen des Profils (0 bis 8), die anzuzeigen sind. Bei 0 wird das Profil nicht sichtbar. Ohne Zahl wird das Profil eingeblendet. TABS und MASK werden nur gezeigt, wenn sie nicht leer sind.

LOCK, UNLOCK

Das Profil wird für Änderungen gesperrt oder freigegeben.

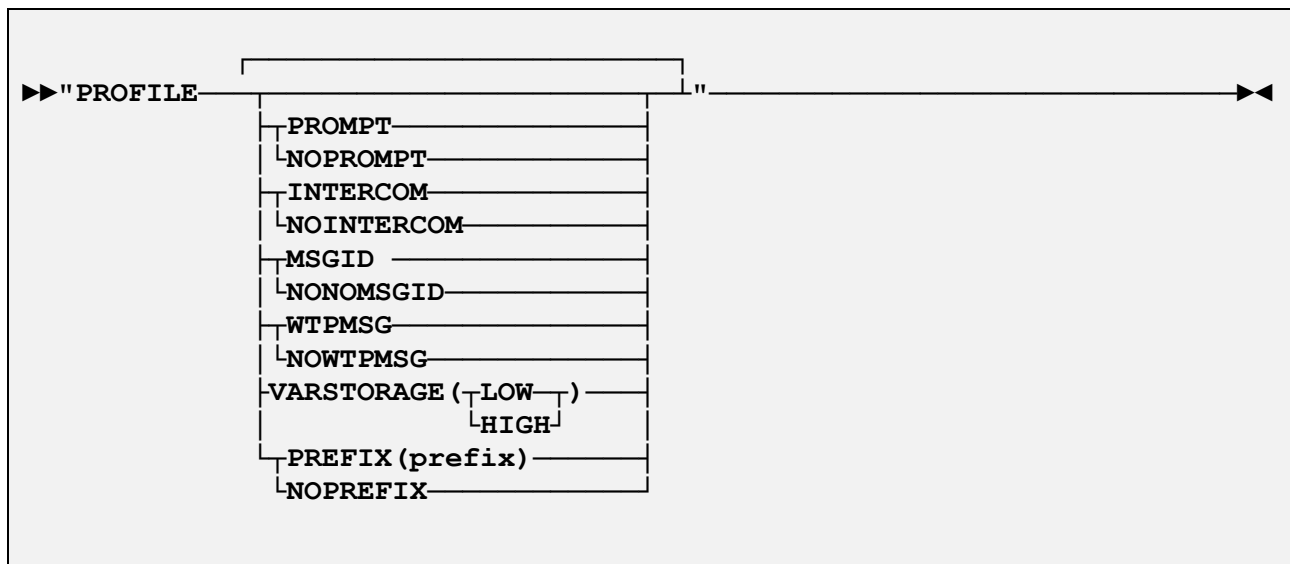
Beispiel

```
"(,lockmode) = profile"
```

Über LOCKMODE kann erkannt werden, ob das Profil gesperrt ist.

Return Codes

0	Normales Ende
20	Schwerer Fehler

PROFILE – TSO Command**PROMPT | NOPROMPT**

Bei fehlenden oder falschen Informationen eines Befehles wird (k)eine Rückfrage gestellt.

INTERCOM | NOINTERCOM

Nachrichten anderer TSO-Benutzer sollen (nicht) empfangen werden.

Die Angabe NOINTERCOM empfiehlt sich auch als Teil einer Prozedur, wenn während des Ablaufes die Bildschirmmasken nicht durch ankommende SEND-Messages gestört werden sollen. Es ist aber darauf zu achten, dass bei Prozedurende wieder auf INTERCOM geschaltet wird.

MSGID | NOMSGID

Einer Meldung wird die Meldungsnummer (nicht) vorangestellt.

PREFIX(prefix)

Wird als erster Namensteil vor den Namen einer Datei gestellt, wenn keine Hochkomata benutzt werden und das TSO zweifelsfrei feststellen kann, dass es sich hierbei um einen DSN handelt.

NOPREFIX

Dateinamen werden mit keinem Prefix versehen. Von dieser Möglichkeit sollte nicht Gebrauch gemacht werden.

WTPMSG | NOWTPMSG

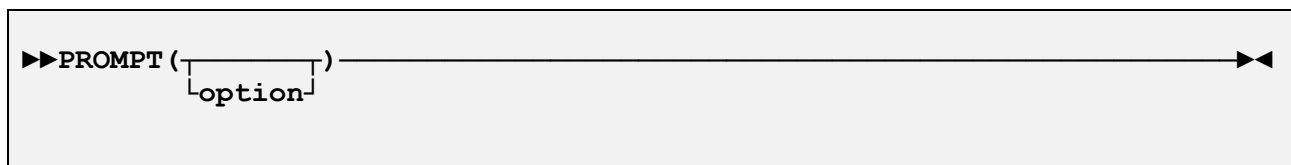
Zeigt oder unterdrücke die "Write-To-Operator-Messages".

VARSTORAGE(LOW|HIGH)

Regelt, dass Variablen aus OUTTRAP() oder privilegierten Commands oberhalb der 16MB-Grenze abgelegt werden können (oder nicht).

Return Codes

0	Normales Ende
12	Fehler. Eine Meldung wurde ausgegeben

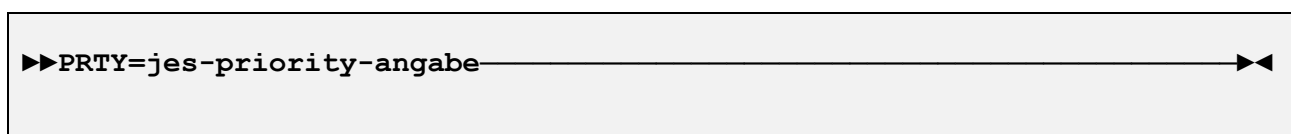
PROMPT() – TSO extended Function

Liefert die Aktuelle Einstellung des Promptings aus dem TSO-Kommando PROFILE.

option

Ohne Angabe von option wird nur der aktuell eingestellte Wert für das Prompting zurückgegeben.

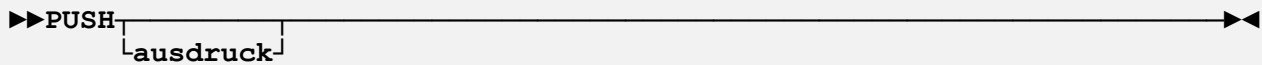
Wird ON oder OFF angegeben, wird das Prompting entsprechend gesetzt. Gleichzeitig wird in der Zielvariablen aber auch der vorher gültige Wert abgelegt.

PRTY – JCL JOB Parameter

PRTY liefert die Auswahlpriorität des SPOOL-Systems. Mögliche Angaben in der Job Control sind die Werte von 0 (Minimum) bis 13 (Maximum).

In einigen Unternehmen werden Prioritätsangaben, welche über Job Control geliefert werden, bei Übergabe der Jobs an den Internal Reader ignoriert.

PUSH – REXX Instruction

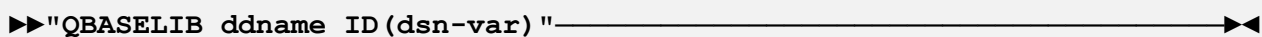


Der aufgelöste Ausdruck wird im STACK-Bereich nach dem Prinzip LIFO (Last In First Out) eingefüllt. Wird kein Ausdruck angegeben, wird im Stack ein Nullstring (keine Leerzeile sondern eine Art EOF) hinterlegt. Die aktuelle Anzahl von Sätzen in der Queue kann durch die Built-in-Funktion QUEUED() ermittelt werden.

Beispiel

```
/* REXX
*/
"alloc dd(eingabe) da(EDU.DATA(ADRTEST)) shr reuse"
"execio * diskv eingabe (stem input. Finis"
Cnt = 0
Do $=1 to input.0
    If input.$ = ??? then do
        Cnt=Cnt+1
        Queue input.$
    End
End
Push "Datensätze mit Treffern:" Cnt
"alloc dd(ausgabe) sysout(H)"
"execio" queued() "diskw ausgabe (finis"
"free dd(eingabe ausgabe)"
exit
```

QBASELIB – ISPF DM Instruction



Liefert die Namen der Datasets, welche unter dem angegebenen DD-Namen zugewiesen sind.

ddname

kann neben den klassischen ISPF-DD-Namen (ISPLIB, ISPSLIB, usw.) jeder beliebig andere DD-Name sein.

ID(dsn-var)

Die Variable enthält die Dateinamen, welche unter dem entsprechenden DD-Namen zugewiesen sind vollqualifiziert. Dies können maximal 255 Dateien sein (Limit im TSO).

Return Codes

0	Normales Ende
4	Der angegebene DD-Name ist nicht definiert
16	Translation- oder Truncation-Fehler
20	Schwerer Fehler

QBUF - TSO extended Command

►►"QBUF"◄◄

Übergibt die Anzahl durch MAKEBUF angelegter Pufferbereiche als Return Code. Wurde durch MAKEBUF kein zusätzlicher Pufferbereich erzeugt, wird RC 0 übergeben.

Beispiel

```
:
"MAKEBUF"          /* Puffer 1 angelegt */
:
"MAKEBUF"          /* Puffer 2 angelegt */
:
"MAKEBUF"          /* Puffer 3 angelegt */
:
:
"DROPBUF"          /* Puffer 3 gelöscht */
"QBUF"
SAY 'Aktiver Puffer' RC  /* Ausgabe 2*/
```

Hinweis:

Das Arbeiten mit gepufferten Stacks ist nicht empfehlenswert. Es bedingt eine andere Verarbeitungslogik und bietet keinerlei Vorteile.

QELEM - TSO extended Command

►►"QELEM"◄◄

Übergibt die Anzahl von Elementen (Datensätzen, Variablen o.ä.), die im aktuellen Pufferbereich eingestellt sind (dies ist der letzte durch MAKEBUF erzeugte Puffer. Die Anzahl der Elemente wird durch die Variable RC mitgeteilt. Wird hierüber eine 0 gemeldet, sind im aktuellen Pufferbereich keine Daten hinterlegt.

QELEM ist zudem eine Hilfe, um ein versehentliches Löschen von Datenpuffern durch die Anweisung DROPBUF zu verhindern. So könnte diese Anweisung nur dann erlaubt werden, wenn nach einem vorangegangenen QELEM-Befehl die Variable RC den Wert 0 enthält.

Im Gegensatz zu QELEM kann durch die Built-in-Funktion QUEUED() nicht nur die Anzahl Elemente im aktuellen Puffer, sondern die Gesamtanzahl von Elementen im STACK ermittelt werden.

Hinweis:

Das Arbeiten mit gepufferten Stacks ist nicht empfehlenswert. Es bedingt eine andere Verarbeitungslogik und bietet keinerlei Vorteile.

QLIBDEF – ISPF DM Service

```
►►"QLIBDEF ddname ID(dsn-var) [TYPE(typ-var)]"◄◄
```

Liefert die per LIBDEF zugewiesenen Datasets und deren Zugriffstyp.

ddname

Es können alle DD-Namen angegeben werden, welche in einem LIBDEF-Statement erlaubt sind.

ID(dsn-var)

Bis zu 15 Dateinamen, welche im letzten aktiven LIBDEF unter dem angegebenen DD-Namen zugewiesen wurden.

TYPE(typ-var)

Liefert eine Information, welcher Art die Zuweisung im LIBDEF war (DATASET, EXCLDATA, etc.)

Return Codes

0	Normales Ende
4	Der angegebene DD-Name hat keine aktive LIBDEF-Zuweisung
12	Ein falscher Typ für ISPPROF wurde angegeben
16	Translation oder Truncation Fehler
20	Schwerer Fehler

QTABOPEN – ISPF DM Service

```
►►"QTABOPEN LIST(list-var)"◄◄
```

QTABOPEN erlaubt festzustellen, welche ISPF-Tabellen zum jeweiligen Zeitpunkt geöffnet sind. Mit TBSTATS und TBQUERY können dann Details erfragt werden.

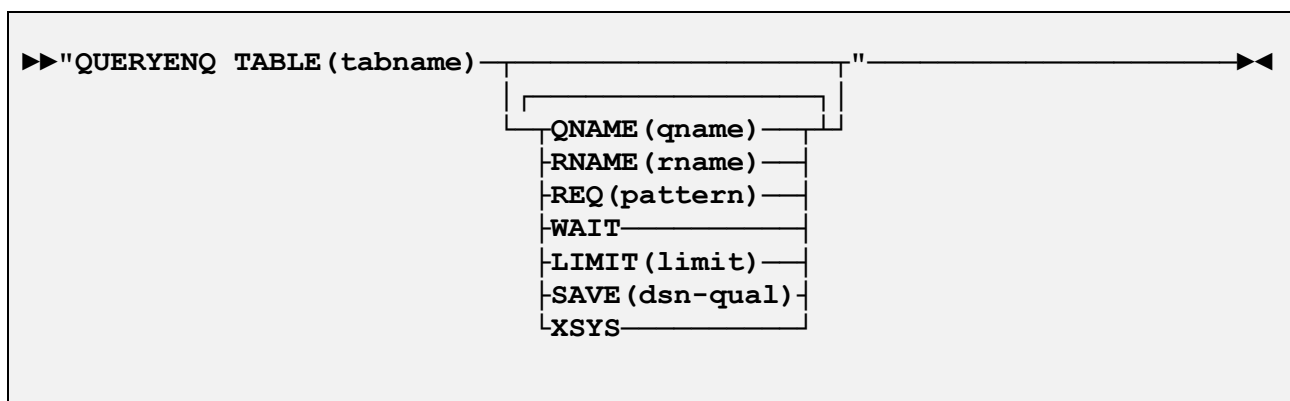
LIST(list-var)

Liefert die Namen aller zum jeweiligen Zeitpunkt geöffneten ISPF-Tables. Die angegebene Variable wird als Prefix benutzt. In der Variablen prefix0 wird die Anzahl der geöffneten Tabellen ausgegeben. In prefix1 und folgende stehen die Namen.

Return Codes

- 0 Normales Ende
- 4 Liste unvollständig (nicht genügend Speicher zum Listaufbau vorhanden)
- 12 Prefix zu lang
- 20 Schwerer Fehler

QUERYENQ – ISPF DM Service



QUERYENQ erlaubt die Abfrage aller, oder bestimmter, System Enqueues.

TABLE(tabname)

Die Tabelle muss nicht vorhanden sein. Sie wird mit NOWRITE zur Verfügung gestellt und kann – so gewollt – mit TBEND gelöscht werden.

QNAME(qname)

Variable oder Prefix zur Informationsaufnahme. Der Prefix muss mit Stern(*) enden. Default ist "*". Sie enthält den Namen eines Query Typs, der abgefragt werden soll. Query Namen sind beispielsweise SYSDSN, oder SPFEDIT.

RNAME(rname)

Variable oder Prefix zur Informationsaufnahme. Der Prefix muss mit Stern(*) enden. Default ist "*". Sie enthält den Namen einer Ressource, der abgefragt werden soll.

REQ(pattern)

Wird benutzt, um die Suche nach Enqueues zu limitieren. Pattern kann die üblichen Wildcards unter ISPF (Stern(*) und Prozent(%)) enthalten.

WAIT

Alle wartenden Enqueues werden angezeigt.

LIMIT(limit)

Maximalgröße der Tabelle. Defaultwert ist 5000. Die Angabe Null(0) bedeutet unbegrenzt.

SAVE(dsn-qual)

8-Byte Dateinamens-Qualifier, der benötigt wird, wenn die Informationen auf Datei ausgegeben werden sollen. Der Name des Datasets ist (nach TSO-Konventionen): prefix.dsn-quali.ENQLIST. Der Satzaufbau ist von links nach rechts: Owner, System, Disposition, Hold, Scope, Global, QName, RName.

XSYS

Enqueues von anderen Systemen werden berücksichtigt (Standard = NEIN). Dies kann gravierende Performanceeinbußen mit sich bringen.

Hinweis

Unter Umständen werden nicht alle Enqueue-Daten zurückgegeben, wenn sehr viele Requests mit der gleichen QNAME/RNAME-Kombination vorhanden sind. Unter diesen Umständen können maximal 84 Requests ermittelt werden.

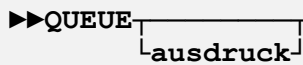
Felder der Tabelle

Feldname	Länge	Bedeutung
ZENJOB	8	Job / Adressraum der den ENQ hält oder angefordert hat.
ZENQNAME	8	Namensteil des ENQ
ZENRNAME	255	Ressourceteil des ENQ
ZENDISP	5	SHARE oder EXCLU
ZENHOLD	4	OWN oder WAIT
ZENSCOPE	7	SYSTEM oder SYSTEMS
ZENSTEP	7	STEP oder Blank
ZENGLOBL	6	GLOBAL oder Blank
ZENSYST	8	Systemname
ZENRESV	7	RESERVE oder Blank

Return Codes

- 0 Normales Ende
- 8 Das Panel enthält die angegebene AREA nicht
- 12 Panel nicht gefunden
- 16 Speicherüberlauf. Nicht alle Werte zurückgegeben
- 20 Schwerer Fehler

QUEUE – REXX Instruction



Der aufgelöste Ausdruck wird im STACK abgelegt. Dies hat eine Verarbeitung nach dem Motto FIFO (First In First Out) zur Folge.

Wird kein Ausdruck angegeben, wird ein Nullstring hinterlegt. Dies wird beim Auslesen des Stackbereichs wie EOF interpretiert.

QUEUED() – REXX Built-in-Function



Übergibt die Gesamtanzahl der Zeilen im aktiven STACK-Bereich.

Ist der Aktive STACK-Bereich in mehrere Pufferbereiche gegliedert, wird hier die Anzahl der Datensätze aus allen Pufferbereichen übergeben (siehe auch unter "QELEM").

Beispiel

```
/* REXX */
say 'Dateinamen vollqualifiziert eingeben'
pull datei
"newstack"
"alloc fi(ein) da('strip(datei,',')) shr reuse"
"execio * diskre ein (finis"
say "In" datei "sind" queued() "Datensätze gespeichert"
"delstack"
exit
```

QSTACK - TSO extended Command



Übergibt die Anzahl existierender STACK-Bereiche in der Variablen RC. Wurde während Prozedurablauf kein neuer STACK erzeugt, beinhaltet RC den Wert 1 für den originären STACK. Werden im Ablauf STACK-Bereiche in geschachtelten Subroutinen oder untergeordneten EXECs gebildet, werden diese ebenfalls von QSTACK erfasst.

Beispiel

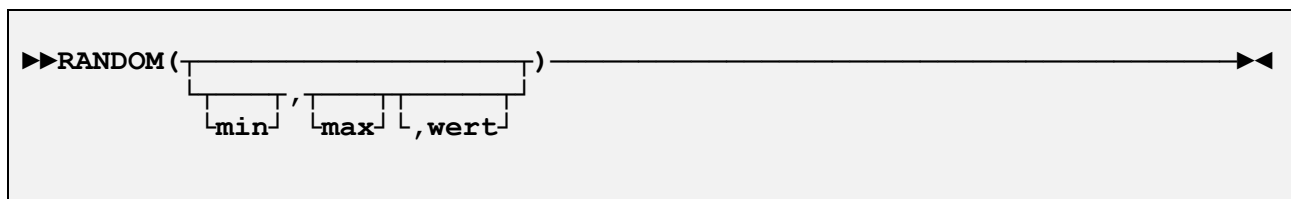
```

:
"NEWSTACK"                /* STACK 2 wird erzeugt */
CALL sub1
"QSTACK"
SAY 'Anzahl STACKs:' RC    /* RC = 5 */
EXIT

sub1:
"NEWSTACK"                /* STACK 3 wird erzeugt */
:
"QSTACK"
SAY 'Anzahl STACKs:' RC    /* RC = 3 */
CALL sub2
RETURN

sub2:
"NEWSTACK"                /* STACK 4 wird erzeugt */
:
"NEWSTACK"                /* STACK 5 wird erzeugt */
:
RETURN

```

RANDOM() – REXX Built-in-Function

Zufallszahl zwischen 0 und 999, wenn max und min nicht angegeben ist. Diese beiden Werte können ansonsten die Streuung der Pseudo-Zufallszahl steuern. Der zwischen max und min liegende Bereich darf nicht größer als 100000 sein.

Wird der Randomizer mehrfach in einer Schleife aufgerufen und beim ersten Aufruf des Randomizers wert mitgegeben, sorgt dies für eine bessere "Zufälligkeit".

Beispiele

```

say RANDOM()              329
say RANDOM(7,14)          12
say RANDOM(5)             43

/* rexx */
do forever
  say 'Zahlenraten, Eingabe (0 bis 9) '
  pull zahl
  if ^datatype(zahl,"W") ! zahl > 9 then exit
  if zahl=random(0,9) then say 'Treffer, OK'
  else say 'Falsch geraten, Stümper'
end

```

RANGE_CMD – ISPF Edit Macro

```
►►" (variable) = RANGE_CMD"◄◄
```

Liefert das Bereichskommando des Anwenders, das an das Macro übergeben wurde. Es muss ein PROCESS-Befehl im Macro vorausgehen.

variable

Enthält das Bereichskommando.

Beispiel

```
"macro NOPROCESS"  
"process range $ #"  
" (befehl) = range_cmd"  
if befehl = "$" then ...  
else ...
```

Das Macro kann unterschiedlich auf die eingegebenen Bereichsbefehle reagieren.

Return Codes

0	Normales Ende
4	Kein Zeilenbefehl eingegeben
8	Ungültiges Zeilenkommando
20	Schwerer Fehler

RCHANGE – ISPF Edit Macro

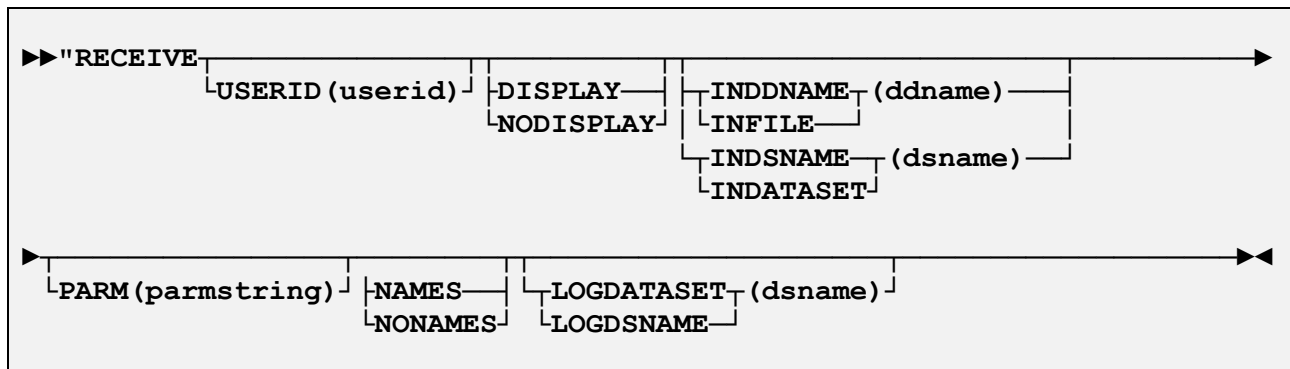
```
►►"RCHANGE"◄◄
```

Wiederholt den letzten CHANGE-Befehl. Wird der Befehl nach dem Ereignis ZEICHENFOLGE NICHT GEFUNDEN eingesetzt, wirkt er aufs Neue, abhängig von den Angaben FIRST oder NEXT, beziehungsweise LAST oder PREV im CHANGE-Kommando.

Return Codes

0	Normales Ende
4	Zeichenfolge nicht gefunden
8	Fehler. Neue Zeichenfolge länger als die alte. Ein Ersetzen war mindestens einmal nicht möglich
12	Syntaxfehler
20	Schwerer Fehler

RECEIVE – TSO Command (privilegiert)



RECEIVE gehört zu den privilegierten TSO-Kommandos. Das heißt: je nach Definitionen im Unternehmen, kann der Befehl nicht von jedem ausgeführt werden. (siehe auch TSOEXEC-Kommando) **DSNAME | DATA**

Mit Receive können Daten empfangen werden, welche durch das TRANSMIT-Kommando im Spool der Empfangs-LPAR angeliefert wurden oder in einer Datei im entsprechenden Format vorliegen.

USERID(userid)

Erlaubt den Empfang von Daten, welche an eine Kennung ungleich der eigenen geschickt wurden.

DISPLAY | NODISPLAY

Regelt die Anzeige oder Unterdrückung folgender Informationen

- MSGDATASET oder MSGDSNAME
- MSGDDNAME oder MSGFILE
- MESSAGE oder MSG
- TERMINAL

INDDNAME | INFILE

Bezieht sich auf den DD-Namen einer preallocated Datei

INDSNAME | INDATASET

Das Format kann sein: F, FB, V, VB, oder U. Für Satzformat F und FB wird in der Regel Satzlänge 80 erwartet (ist installationsabhängig).

NAMES | NONAMES

Regelt, welche Art von Informationen gegebenenfalls im Log-Dataset abgelegt werden.

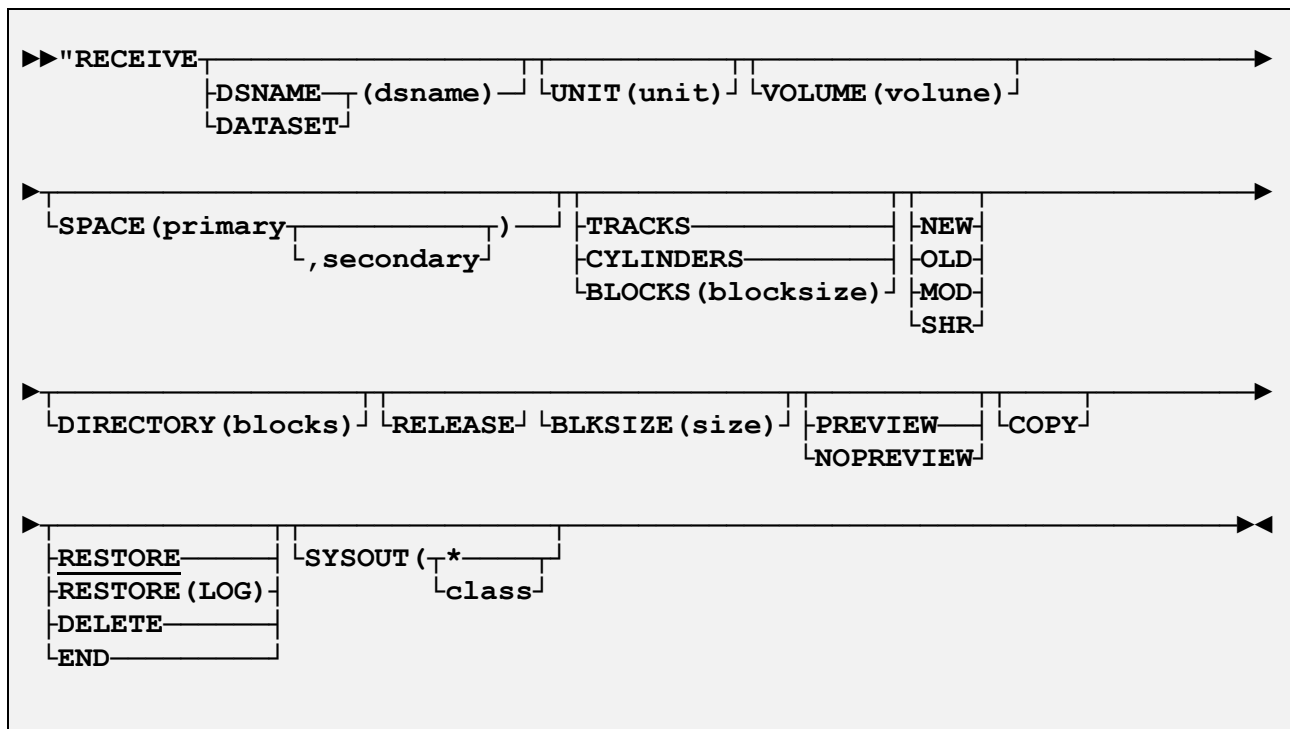
LOGDATASET | LOGDSNAME

Sofern gewollt (aber wer will das schon wirklich) könnte hier der Name einer Logdatei abweichend vom Standard definiert werden.

PARM(parmstring)

Ob diese Angabe nötig ist, ist installationsabhängig. Ist dies der Fall, kann die Systemprogrammierung hierzu Auskunft geben. Für gewöhnlich ist dies aber nicht der Fall.

Sofern der RECEIVE keine Daten aus dem Spoolsystem übernehmen soll, kann er auch direkt auf eine vom TRANSMIT Befehl (XMIT) erstellte Datei als Input benutzen. In diesem Fall benötigt das RECEIVE Kommando einige Informationen, welche es beim Nutzer zurückfragen wird. Hier sind die entsprechenden Angaben, welche auch in Kombination angegeben werden können.

**DSNAME | DATASET**

Datei, in welche received wird. Ist die Datei nicht vorhanden, wird sie angelegt.

UNIT, VOLUME, SPACE, ...

Diverse Angaben, falls die Datei neu angelegt werden soll.

SYSOUT(

JES-Outputklasse, welche die Message des Receive Befehls entgegennimmt (anderenfalls Bildschirmausgabe)

PREVIEW | NOPREVIEW

Zeigt die Daten an (oder auch nicht) wie sie gespeichert sind.

COPY

Die Daten werden nicht entladen, sondern unter einem anderen Namen gespeichert und können zu einem späteren Zeitpunkt per RETRIEVE INDATASET entladen werden.

RESTORE

Die Daten sollen extrahiert werden (Default, wenn nicht angegeben)

RESTORE(LOG)

Die Daten werden in das definierte LOG entladen. Dies ist eher sehr ungewöhnlich und kann nicht mit dem Parameter DATASET oder DSNNAME kombiniert werden.

DELETE

Die Datei wird gelöscht ohne die Daten zu extrahieren.

END

RECEIVE wird beendet. Die Daten verbleiben aber im Spool für eine spätere Verarbeitung.

Return Codes

0	Normales Ende
4	Ergebnis OK, es wurden aber Warnings ausgegeben
8	Verarbeitung unvollständig
12	Verarbeitung fehlerhaft beendet
16	Schwerer Fehler (AbEnd)

RECFM – ISPF Edit Macro

```
►►" (var1,var2) = RECFM"◄◄
```

Liefert das Bereichskommando des Anwenders, das an das Macro übergeben wurde. Es muss ein PROCESS-Befehl im Macro vorausgehen.

Var1

Enthält einen der beiden Werte:

F	für feste Länge
V	für variable Länge

Var2

Liefert in Ergänzung zum Inhalt der ersten Variablen einen der nachfolgenden Werte:

B	Blocked
S	Standard oder Spanned Records
M	Maschinen-Steuerzeichen
A	ANSI-Steuerzeichen

Beispiel

```
address isredit  
"(format) = recfm"  
if format = "F" then "(breite) = data_width"  
else breite = "(,breite) = display_cols"  
Ermittelt die Satzlänge für feste oder variable Satzlänge.
```

Return Codes

0	Normales Ende
20	Schwerer Fehler

RECFM – JCL DD Parameter



Bestimmt das Satzformat. Hier werden nur die gängigen beschrieben.

F | FB | FBA | FBM

Es sind immer Sätze fester Länge, gegebenenfalls geblockt und/oder mit Ansicode- oder Machine-code-Steuerzeichen.

V | VB | VBA | VBM

Variable Länge, ansonsten wie oben

VS | VBS

Spanned Records. Hier kann die Satzlänge/Blockgröße größer als die Spurkapazität sein.

U

Undefined. Wird nur für Lademodul-Bibliotheken benutzt.

RECOVERY – ISPF Edit Macro

```
►►" (variable) = RECOVERY"_____►◄  
  
►►"RECOVERY = [ON  
               [OFF]"_____►◄
```

Liefert den aktuellen RECOVERY-Modus, oder setzt ihn

=RECOVERY

Der aktuelle Wert von RECOVERY wird in einer Variablen hinterlegt.

RECOVERY=

Der RECOVERY-Modus wird entsprechend im Edit Profil hinterlegt.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

REFDD – JCL DD Parameter

```
►►REFDD=[* .ddname_____►◄  
        [* .stepname.ddname]
```

Diese Angabe wird nur in Verbindung mit SMS berücksichtigt. Für SMS-gemanagte Dateien können auf diese Weise Informationen wie Satzformat, Dateorganisation und SPACE-Angaben übernommen werden.

REGION – JCL JOB-/EXEC Parameter

```
▶▶REGION=nnn[K]
                [M]
```

Bestimmt die Größe des virtuellen Adressraumes für einen Job. Wenn angegeben, wird eine REGION-Angabe in der EXEC-Anweisung ignoriert.

Wird keine REGION-Größe angegeben, wird ein Standard von 1MB (=1024 K) angenommen (außer, dies ist im Unternehmen anders definiert).

REMPop – ISPF DM Service

```
▶▶"REMPop[ALL]"
```

Entfernt das letzte POP-UP-Fenster, das mit ADDPOP erstellt wurde. Nach Absetzen des REMPOP-Services werden alle folgenden Panels entweder "normal" oder in einem vorher definierten Fenster dargestellt.

ALL

Alle definierten POP-UP-Windows die unter dem gleichen SELECT-Level liegen, werden entfernt. Alle folgenden Panels werden auf dem gesamten Bildschirm dargestellt.

Return Codes

0	Normales Ende
16	In diesem Select Level existiert keine Window-Definition
20	Schwerer Fehler

RENAME – TSO Command

```
▶▶"RENAME dateiname-alt dateiname-neu
      dateiname(member-alt) (member-neu)"
```

Umbenennen einer sequentiellen Datei oder eines Member einer PO-Datei.

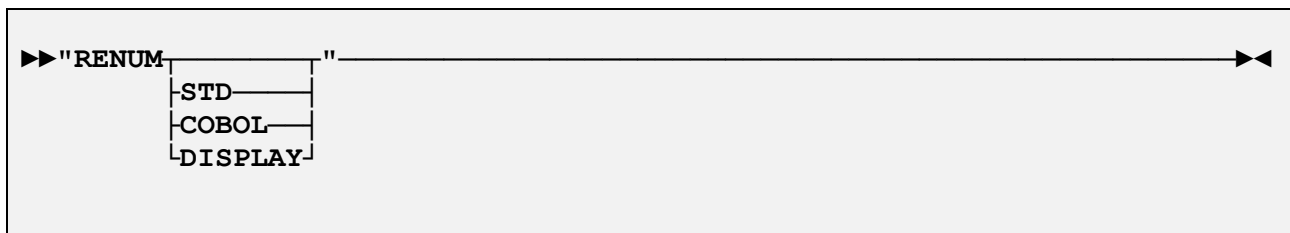
Soll ein Member einer PO-Datei umbenannt werden, sind die LM-Services vorzuziehen, da RENAME die Datei exklusiv belegt.

Analog könnte auch der IDCAMS-Befehl ALTER oder der LMMREN-Service benutzt werden (ist in vielen Fällen die bessere Wahl).

Return Codes

- 0 Normales Ende
- 12 Fehler. Eine Fehlernachricht wurde ausgegeben

RENUMBER – ISPF Edit Macro



Schaltet den Nummernmodus ein (sofern er nicht bereits aktiv ist) und nummeriert die Zeilen beginnend mit Zeilennummer 000100 in einer Schrittweite von 100 neu

STD

Für die Nummerierung wird das Standardfeld genutzt

COBOL

Die Zeilennummern werden im COBOL-Feld eingetragen

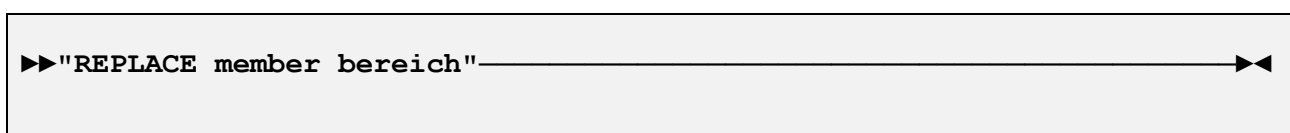
DISPLAY

Die Breite des Datenfensters schließt die Sequenznummern ein.

Return Codes

- 0 Normales Ende
- 20 Schwerer Fehler

REPLACE – ISPF Edit Macro



Überschreibt ein Member der PO-Datei. Existiert das Member nicht, wird es angelegt (impliziter CREATE).

member

Name des zu überschreibenden Members.

bereich

kann durch relative Zeilennummern oder Label festgelegt werden und definiert die Zeilen im aktuellen Member, mit denen das Überschreiben stattfinden soll.

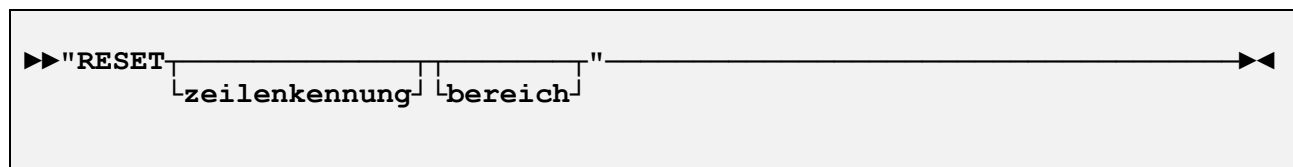
Beispiel

```
"replace adr2 5 25"
```

Das Member ADR2 wird mit den Zeilen 5 bis 25 des aktuellen Members überschrieben.

Return Codes

- 0 Normales Ende
- 8 Einer der folgenden Gründe
 - Falscher Linepointer
 - Member nicht gefunden
 - BLDL Fehler
- 12 Syntaxfehler (Membername oder Linepointer falsch)
- 20 Schwerer Fehler

RESET – ISPF Edit Macro

Wird RESET ohne alle Operanden angegeben, setzt der Befehl den Status aller Zeilen zurück (Label bleiben erhalten), und löst alle noch offenen Zeilenbefehls-Widersprüche auf.

zeilenkennung

Als Kennung einer Zeile sind möglich

Zeilenkennung	Kurz	Bedeutung
CHANGE	CHG	Setzt die Hinweise ==CHG> zurück
COMMAND	COM	Löscht unvollständige Zeilenkommandos
ERROR	ERR	Setzt die Hinweise ==ERR> zurück
EXCLUDED	X	Zeigt ausgeblendete Zeilen wieder an
LABEL	LAB	Setzt LABEL zurück
SPECIAL	SPE	Blendet alle NON-DATA-Zeilen wieder aus. Dies sind: =PROF> (die Profilzeilen) =MASK> (die Maskenzeile) =BNDS> (die Boundaries) =TABS> (die Tabulatorzeile) ==MSG> (die Nachrichtenzeilen) ==NOTE= (Hinweiszeilen)

bereich

Begrenzt den Zeilenbereich, innerhalb dessen der RESET-Befehl wirksam werden soll. Der Bereich kann über relative Zeilennummern oder Label geliefert werden.

Beispiele

```
"reset lab"
```

Alle Label werden entfernt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

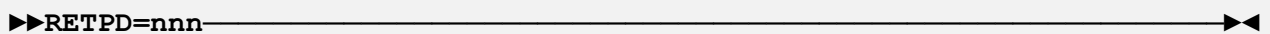
RETPD – JCL DD Parameter

Diagram illustrating the RETPD parameter syntax: RETPD=nnn. The parameter is enclosed in a box with arrows indicating its position within a line of code.

Retention Period benennt eine Anzahl Sicherungstage. Das Systemdatum zugrunde gelegt, berechnet das System daraus ein Verfalldatum für eine Datei

nnn

Wert in Tagen zwischen 0 und 93000

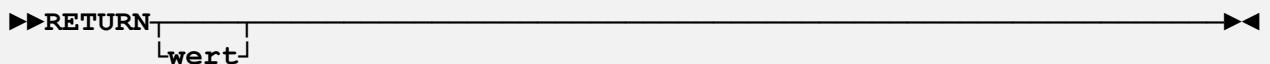
RETURN – REXX Instruction

Diagram illustrating the RETURN instruction syntax: RETURN wert. The instruction is enclosed in a box with arrows indicating its position within a line of code. The word 'wert' is shown in a small box below the instruction.

RETURN wird genutzt, um aus einer EXEC zu einer höher liegenden (oder auch zurück zu einer CLIST), beziehungsweise zurück zum Punkt des Aufrufes zu kommen. Dabei können Variablen zurückgereicht werden. Existiert keine übergeordnete Prozedur, führt die Anwendung der Instruktionen RETURN und EXIT zum gleichen Ergebnis (die Prozedur wird beendet).

Der Rückgabewert liegt im rufenden Programm im Falle einer Subroutine in der Variablen RESULT.

Erfolgte der Aufruf als Function, tritt der retournierte Wert im Ablauf an die Stelle der Funktion. Gibt eine Funktion keinen Wert zurück, führt dies zum Abbruch.

Wird ein Hauptprogramm mit RETURN beendet und ein Wert zurückgegeben, muss dieser numerisch, integer zwischen 0 und 4095 sein.

Beispiel

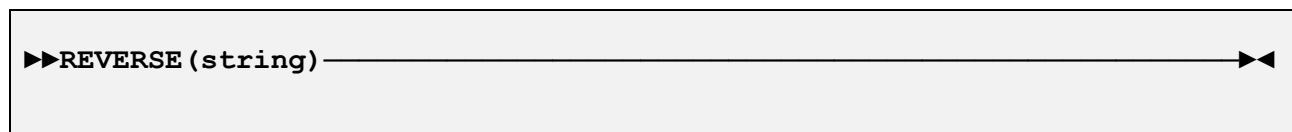
Hauptprogramm	Unterprogramm
<pre> /** REXX * RX1 *****/ \$=2 DO 2 \$=\$*\$ CALL RX2 \$=RESULT SAY 'Wert von \$:' \$ END </pre>	<pre> /* REXX * RX2 *****/ ARG \$ SAY 'Wert von \$:' \$ \$=\$*\$ RETURN \$ </pre>

Ablauf

```

Wert von $: 4
Wert von $: 16
Wert von $: 256
Wert von $: 65536

```

REVERSE() – REXX Built-in-Function

Kehrt die Zeichenfolge eines Strings um.

Beispiele

```
say REVERSE('llot tsi XXER')      'REXX ist toll'
```

```

/* REXX
*/
:
call listdsi "EINGABE FILE"
parse value reverse(sysdsname) llq "." .
llq = reverse(llq)
if llq <> "DATA" then ...
:

```

Ermittelt den Low-Level-Qualifier einer Datei, welche über den DD-Namen EINGABE zugewiesen ist.

RFIND – ISPF Edit Macro

```
"RFIND"_____▶◀
```

Wiederholt den letzten FIND-Befehl. Wird der Befehl nach dem Ereignis ZEICHENFOLGE NICHT GEFUNDEN eingesetzt, wirkt er aufs Neue, abhängig von den Angaben FIRST oder NEXT, beziehungsweise LAST oder PREV im FIND-Kommando.

Return Codes

0	Normales Ende
4	Zeichenfolge nicht gefunden
12	Syntaxfehler
20	Schwerer Fehler

RIGHT – ISPF Edit Macro

```
▶▶"RIGHT scrollamount"_____▶◀
```

RIGHT wird benutzt, um den Bildschirmausschnitt nach rechts zu verschieben.

scrollamount

MAX	Zum Datenende
HALF	um eine halbe Seite
PAGE	um eine ganze Seite
CURSOR	Die Cursor-Zeile wird die oberste.

Beispiele

"right max"

Zeigt den rechten Rand der Datei.

Return Codes

0	Normales Ende
4	Keine sichtbaren Datenzeilen
8	Keine Datenzeilen anzuzeigen
12	Scrollamount fehlt
20	Schwerer Fehler

RIGHT() – REXX Built-in-Function

```
►►RIGHT(string,länge[,füllzeichen])◄◄
```

Übergibt eine Zeichenfolge in der angegebenen Länge. Situationsabhängig wird nach der angegebenen Anzahl Byte von rechts auf der linken Seite abgeschnitten, oder mit dem angegebenen Füllzeichen aufgefüllt (Standard ist BLANK)

Beispiele

```
say RIGHT('Freibier',10)      '  Freibier'  
say RIGHT('Freibier',10,'*') '**Freibier'  
say RIGHT('Hans ist doof',4)  'doof'  
say RIGHT('25',4,'0')        '0025'
```

RMACRO – ISPF Edit Macro

```
►►"(variable) = RMACRO"◄◄  
  
►►"RMACRO = macroname"◄◄
```

Erfragt oder bestimmt ein Recoverymacro für das aktuelle Profil. Das Makro wird im Recoveryfall ausgeführt, bevor die Daten vom Editor gezeigt werden.

=RMACRO

Überstellt den Namen des gültigen Recoverymacros der angegebenen Variablen.

RMACRO=

Legt ein Recoverymakro fest und trägt den Namen in das aktuelle Profil ein.

Return Codes

0	Normales Ende
12	Falscher Macroname
20	Schwerer Fehler

ROLLBACK – DB2 Instruction

```
►►"ROLLBACK"◄◄
```

Wird benutzt, um alle Änderungen innerhalb einer Transaktionsklammer rückgängig zu machen, falls eine Aktion nicht durchführbar war.

Generell könnte hier auch auf bestimmte sogenannte SAVEPOINTS zurückgesetzt werden, welche aber im Vorfeld gesetzt werden müssen. Dies würde hier aber den Rahmen sprengen.

ROUTE – JCL JES Instruction

```
/*ROUTE PRINT name  
      XEQ
```

Die ROUTE-Anweisung kann kodiert werden, um Druckausgaben oder Verarbeitung auf einem anderen als dem einlesenden Rechner durchzuführen.

Für die Umweisung einer Druckausgabe, bzw. der Verarbeitung sollen jeweils getrennte ROUTE-Anweisungen kodiert werden.

PRINT

Ein Remote-Drucker wird für die Druckausgabe benannt.

XEQ

Ein Remote-System wird mit der Verarbeitung betraut.

SAVE – ISPF Edit Macro

```
►►"SAVE"◄◄
```

Schreibt die Daten auf Platte zurück. In diesem Zusammenhang sind auch die Beschreibungen unter DATA_CHANGED, AUTOSAVE, CANCEL und END informativ.

Beispiel

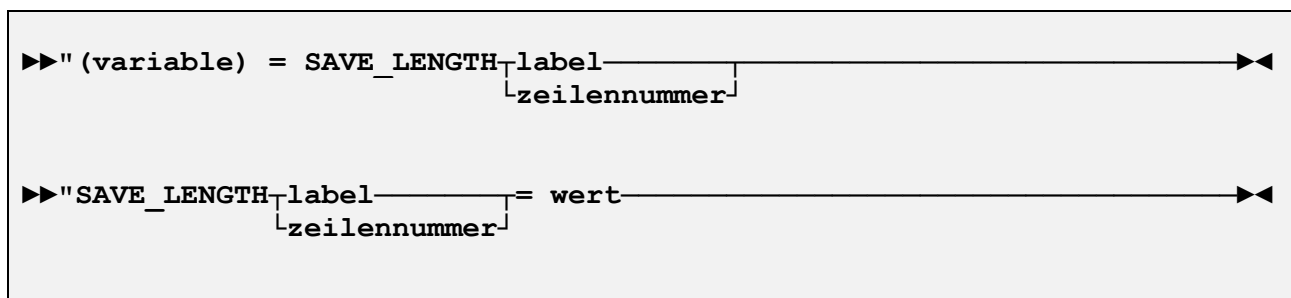
```
"(asave) = autosave"  
if asave = "OFF" then "save"
```

Sofern nicht automatisch bei END gesichert wird, führt das Macro die Sicherung der Daten durch.

Return Codes

0	Normales Ende
4	Neues Member gesichert
12	Keine Sicherung. Platzprobleme n Directory oder PDS
20	Schwerer Fehler

SAVE_LENGTH – ISPF Edit Macro



Setzt oder liefert die Länge, die benutzt werden soll, um Sätze in variabler Länge zu speichern. Es erlaubt grundsätzlich nicht, die Sätze zu verkürzen, aber sie können mit Blanks aufgefüllt werden.

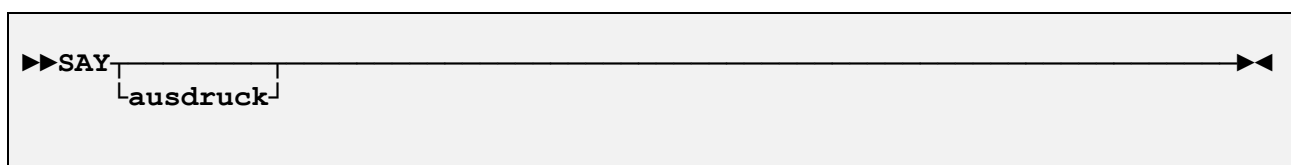
Beispiel

Beispiele sind zu finden im Member ISRSETLN der von der IBM gelieferten ISPF-REXX-Bibliothek.

Return Codes

0	Normales Ende
4	Wert außerhalb des Gültigkeitsbereichs
6	Satzformat nicht variabel
16	Fehler beim Setzen von Variablen
20	Schwerer Fehler

SAY – REXX Instruction



Schreibt den angegebenen Ausdruck an die Standardzuordnung für Output (wenn die EXEC in einem TSO-Adressraum läuft, an das Terminal, in einem MVS-Adressraum an den Zielort des SYST-SPRT-Statements), wobei enthaltene Variable vor Ausgabe aufgelöst werden.

Der Ausdruck kann beliebig lang sein.

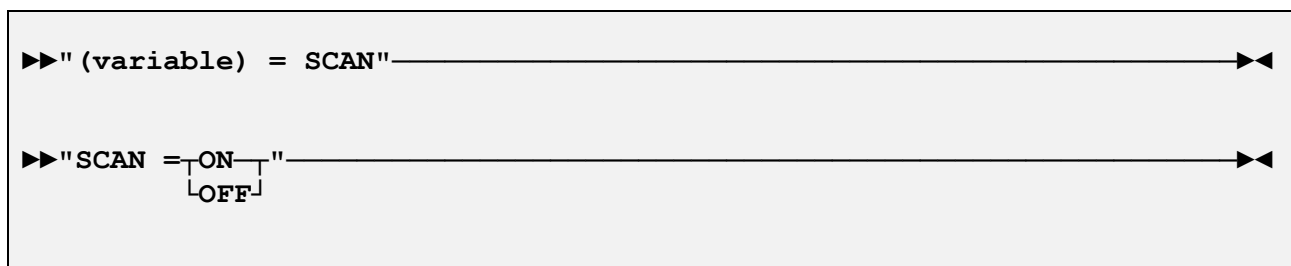
Wird die SAY-Instruktion ohne Ausdruck angegeben, wird eine Leerzeile erzeugt.

Beispiel

```
:
zahl=100
say zahl 'dividiert durch 4 =>' zahl/4

/* Zeigt an: 100 dividiert durch 4 => 25 */
```

SCAN – ISPF Edit Macro



Der Scanmodus entscheidet, wie Variablensubstitution in einer CLIST vor Übergabe an den Editor stattfindet. Auch wenn im Grunde hier nicht auf CLIST eingegangen wird, sollte der Befehl nicht verschwiegen werden.

=SCAN

Der aktuelle Wert von SCAN wird in einer Variablen hinterlegt.

SCAN=

Der SCAN-Modus wird entsprechend gesetzt.

ON

Variable in Befehlszeilen werden automatisch ersetzt.

OFF

Das automatische Ersetzen von Variablen wird nicht durchgeführt.

Beispiel

```
"scan off"
"line &z='&&SYSTIME ist eine Funktion'"
```

Der CLIST-Befehlsprocessor führt eine einmalige Substitution der Variablen durch. &z wird durch seinen Wert ersetzt, von &&SYSTIME wird das erste Ampersand(&) entfernt. Da der Scanmodus ausgeschaltet ist, wird die Variable &SYSTIME vom Editor nicht interpretiert.

Return Codes

0	Normales Ende
20	Schwerer Fehler

SCHENV – JCL JOB Parameter

```
►►SCHENV=schenvname◄◄
```

Routet einen Job innerhalb eines Sysplex auf diejenige LPAR, auf der das angegebene Scheduling Environment aktiv ist.

Hinweis

Wenn ein bestimmtes Environment (z.B.: ein DB2-System) nicht immer auf der gleichen LPAR läuft, ist es sinnvoller, den SCHENV-Parameter zu benutzen, als mit einer JES-Route-Anweisung die Verarbeitung auf eine bestimmte LPAR zu legen. Ist das DB2-System dort nicht aktiv, führt dies zum Abbruch.

Beispiel

```
//SP1 JOB RZ, '37.E.05', CLASS=E, SCHENV=DB2A
```

Die Verarbeitung wird von JES auf die LPAR innerhalb des PLEX geroutet, auf dem das DB2A aktiv ist.

SEEK – ISPF Edit Macro

```
►►"SEEK string [bereich] [NEXT] [CHARS] [X] [sp1] [sp2]"◄◄
    [ALL] [PREFIX] [NX]
    [FIRST] [SUFFIX]
    [LAST] [WORD]
    [PREV]
```

SEEK sucht nach einer Zeichenkette. Die Parameter sind die gleichen, wie beim FIND. Im Gegensatz zu FIND erlaubt der SEEK-Befehl das Aufsuchen jeder Zeile, ohne den Zeilenstatus zu verändern. Innerhalb eines Makros sollte SEEK gegenüber FIND den Vorzug erhalten.

string

Zeichenkette, nach der gesucht wird.

bereich

Eine von/bis Angabe in Form zweier Marken (Labels). Die Angabe von relativen Zeilennummern ist nicht möglich.

Richtung

NEXT	Suche beginnt ab Cursorposition in Richtung Dateiende
ALL	Suche beginnt am Dateianfang in Richtung Ende. Gesucht und geändert werden alle vorkommenden Begriffe
FIRST	Die Suche beginnt am Dateianfang in Richtung Ende. Geändert wird der erste vorkommende Begriff
LAST	Wie FIRST aber vom Ende in Richtung Anfang
PREV	Wie NEXT aber in Richtung Dateianfang

Position

CHARS	Die Zeichen müssen wie angegeben aufeinander folgen, können aber an beliebiger Stelle vorkommen
PREFIX	ZEICHENFOLGE MUSS EIN Wortanfang sein
PREF	
SUFFIX	Zeichenfolge muss ein Wortende sein
SUF	
WORD	Die Zeichenfolge muss ein Wort sein

X

Der Befehl wirkt nur in ausgeblendeten Zeilen (eXcluded lines).

NX

Der Befehl wirkt nur in eingeblendeten Zeilen (Not eXcluded lines)

sp1 sp2

Wirkungsbreite des SEEK-Befehls (linke/rechte Spalte). Wird nur ein Wert angegeben, wird er als definitive Spaltenangabe interpretiert.

Hinweise zum SEEK-Befehl

Nicht case-sensitiv. Enthält string Sonderzeichen oder Blanks, oder ist sie gleichnamig zu Operanden des Befehles, müssen Apostrophe (') benutzt werden. SEEK wirkt innerhalb der Boundary-Grenzen.

Wird NEXT/PREV und ein Bereich kodiert und der Cursor steht außerhalb, wird der Bereich durchsucht. Sonst beginnt die Suche ab Cursor.

Beispiel

```
"seek last p'####' word"
```

Sucht nach dem letzten vorkommenden 4-stellig numerischen Wert, der die Wortbedingung erfüllt.

Return Codes

0	Normales Ende
4	Zeichenfolge nicht gefunden
12	Syntaxfehler
20	Schwerer Fehler

SEEK_COUNTS – ISPF Edit Macro

```

▶▶"(var1,var2) = SEEK_COUNTS"

```

Diese Zuweisung liefert die Ergebnisse des letzten SEEK-Befehles.

var1

Name der Variablen, in der die Anzahl der gefundenen Zeichenfolgen hinterlegt werden.

var2

Name der Variablen, in der die Anzahl der Zeilen mit positivem Suchergebnis abgelegt ist.

Beispiel

```
"(ok,lines) = seek_counts"
```

Die Anzahl gefundener Zeichenketten liegt in der Variablen OK, die Anzahl betroffener Zeilen in der Variablen LINES.

Return Codes

0	Normales Ende
20	Schwerer Fehler

SELECT – DB2 Instruction

```

▶▶"SELECT db2-feld FROM schema . table-name
    ORDER BY db2-feld
WHERE AND | OR ausdruck FOR UPDATE OF db2-feld"

```

Bestimmt eine Auswahl von Tabellenfeldern, welche in die Ergebnistabelle übertragen werden sollen.

Generell könnten auch alle Felder einer Tabelle angesprochen werden (`SELECT * FROM`). Dies würde aber eine sehr viel komplexere Nacharbeit nach sich ziehen und wird an dieser Stelle nicht weiter berücksichtigt.

Db2-feld

Feldname(n) der Tabelle die in der Ergebnistabelle landen sollen.

Table-name

Name der Tabelle, aus welcher die Feldinhalte gelesen werden sollen.

ORDER BY

Erlaubt die Sortierung der Ergebnistabelle

WHERE

Benennt einen Ausdruck, der TRUE sein muss, damit der Satz in die Verarbeitung einfließt.

Ausdruck

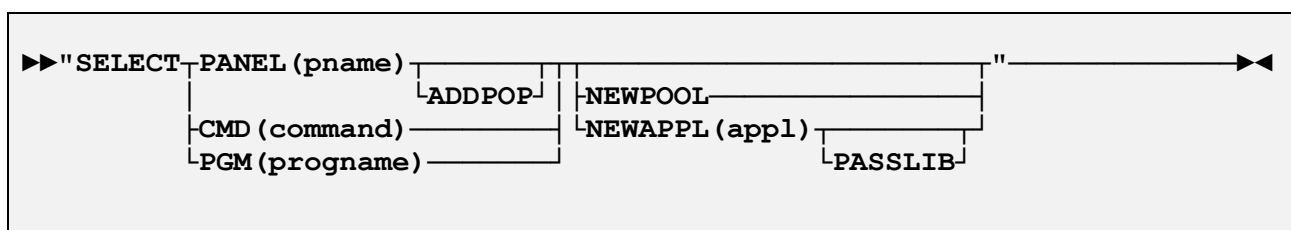
Für den Ausdruck gibt es einige Variationen. Es kann mit den üblichen Vergleichsoperatoren gearbeitet werden. Wird ein Feldinhalt gegen einen String verglichen, muss dieser zwischen Apostrophen stehen. Bei generischen Suchen kann die LIKE-Klausel benutzt werden, innerhalb derer das Prozentzeichen für beliebig steht (analog zu * im ISPF-Umfeld)

```
"SELECT name,geburts_datum FROM table01 WHERE name = :ISPNAME"
"SELECT name,hobby          FROM table01 WHERE hobby LIKE '%BALL%'"
"SELECT name,beruf          FROM table01 WHERE name LIKE 'A%'"
```

FOR UPDATE

Kann alle oder bestimmte Felder vor konkurrierenden Updates schützen.

SELECT – ISPF DM Service



Kann ein Auswahl-Panel, Kommando (auch REXX-Programm oder CLIST) oder Programm (Lademodul) aufrufen. Die Operanden im Einzelnen haben folgende Bedeutung:

PANEL(pname)

ruft ein SELECTION-Panel auf. Im Unterschied zu herkömmlichen Datenpanels wird hier eine Auswahl unter verschiedenen angebotenen Möglichkeiten getroffen. Die Steuerung im Ablauf übernimmt der &ZSEL-Service (vergleiche ISPF-Menü 3 - Utilities).

ADDPOP

Impliziert, dass das entsprechende Panel als Window aufgerufen wird.

Ist ADDPOP angegeben und wird das Selectionpanel verlassen, wird intern ein REMPOP durchgeführt.

CMD(command)

benennt eine CLIST, ein REXX-Programm oder ein TSO-Kommando, das ausgeführt werden soll. Übergabewerte für CLIST oder REXX können innerhalb der Klammern angegeben werden. Für den Aufruf von CLISTen oder REXX-Programmen kann ein führendes Prozentzeichen(%) mitgegeben werden.

PGM(progname)

benennt ein Lademodul, das aufgerufen werden soll.

NEWAPPL(appl)

Legt fest, dass eine neue Applikation aufgerufen wird. Durch appl wird eine 1 bis 4 Byte lange Applikationskennung geliefert, unter der innerhalb ISPPROF neue Member mit folgendem Inhalt eröffnet werden:

xxxxPROF	Applikations-Profile-POOL
xxxxEDIT	Applikations-EDIT-Profile
xxxxCMDS	Applikations-COMMAND-Table

Wird NEWAPPL kodiert, aber keine Applikationskennung angegeben, wird der Standard ISP angenommen. Hieraus ergeben sich die Membernamen innerhalb ISPPROF

ISPPROF	Benutzer-Profile-POOL
ISPEDIT	Benutzer-EDIT-Profile
ISPCMDS	Benutzer-COMMAND-Table

PASSLIB

steuert, dass alle aktuellen Zuweisungen der ISPF-Bibliotheken übergeben werden. Dieser Parameter ist nur in Verbindung mit NEWAPPL(appl) im Bedarfsfall zu kodieren.

NEWPOOL

veranlasst, dass ein neuer SHARED-Pool erzeugt werden soll, auch wenn keine neue Applikation aufgerufen wird. Wird der SELECT-Service verlassen, ist der alte SHARED-Pool wiederhergestellt.

Return Codes

0	Normales Ende
4	Normales Ende. RETURN wurde abgesetzt oder die

- EXIT-Anweisung im &ZSEL-Service wurde ausgeführt
- 12 Das angegebenePanel wurde nicht gefunden
- 16 TRUNC-Fehler bei der &ZCMD oder &ZSEL Variablen
- 20 Schwerer Fehler

SELECT – REXX Instruction

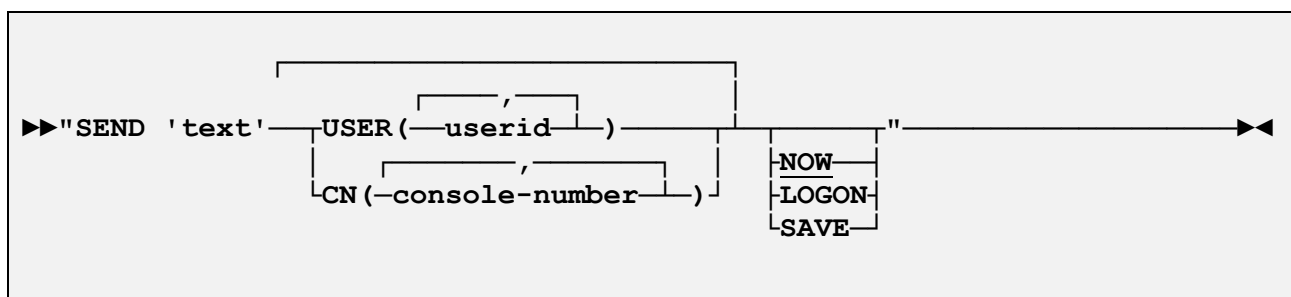
```
SELECT
  WHEN ausdruck THEN instruktion
  WHEN ausdruck THEN instruktion
  :
  OTHERWISE instruktion
END
```

Sind innerhalb eines Verarbeitungsabschnittes mehrere Bedingungen zu überprüfen, die sich gegenseitig ausschließen sollen, kann dies grundsätzlich durch in sich geschachtelte IF-THEN-ELSE-Anweisungen bewerkstelligt werden, die SELECT-Anweisung bietet hier aber eine wesentliche Vereinfachung.

Beispiel

```
/* REXX */
say 'Geben Sie Ihr Alter an'
parse external alt
select
  when datatype(alt) = 'CHAR' then txt='in die Anstalt'
  when alt < 18 then txt='noch in die Schule'
  when alt > 65 then txt='nach Mallorca'
  otherwise txt='sofort an die Arbeit'
end
say 'Sie gehören' txt
exit
```

SEND – TSO-Kommando



Sendet einen beliebigen Text an andere TSO-Benutzer oder an eine Operatorkonsole. Ist das TSO-Profil des Empfängers auf NOINTERCOM gesetzt, wird die Nachricht nicht gesendet.

Der Empfänger erhält mit der Nachricht auch die Benutzerkennung des Absenders.

NOW

Sendet nur, wenn der Empfänger angemeldet ist. NOW ist Default.

LOGON

Sendet die Nachricht beim nächsten LOGON des Empfängers

SAVE

Speichert die Nachricht in der SYS1.BROADCAST ohne sie zu senden. Sie kann per LISTBC-Befehl oder durch einen LOGON empfangen werden.

Return Codes:

0	SEND erfolgreich
12	Fehler beim SEND

SESSION – ISPF Edit Macro

```
►►" (var1,var2) = SESSION"_____◄◄
```

Liefert den Typ der Session, in welcher das Macro läuft und ob SCLM aktiv ist.

Var1

Enthält den Sessientyp und kann sein: EDIF, EDIT, VIEW oder VIIF

Var2

Enthält SCLM, wenn SCLM aktiv ist, oder vier Sterne(****) wenn nicht.

Return Codes

0	Normales Ende
20	Schwerer Fehler

SET – JCL Spezielle Statement

```
//[label] SET variable=wert
```

Die SET-Anweisung kann benutzt werden, um symbolische Parameter mit Werten zu besetzen. Werden mehrere Parameter definiert, sind die einzelnen Parameter mit Komma voneinander zu trennen.

Sind Sonderzeichen in den Wertzuweisungen enthalten, müssen diese Zwischen Apostrophe eingeschlossen werden.

Hinweise:

Eine SET-Zuweisung wird überschrieben:

- von einer nachfolgenden mit dem gleichen Namen.
- von einer Zuweisung innerhalb einer Prozedur
- von einer Zuweisung in einer EXEC-Anweisung

Beispiele

Im SET-Statement werden 3 Werte gesetzt, welche in der JCL über &AA, &BB und &CC angesprochen werden können:

```
//SET3 SET DS=SP1.DEMO.DATA,DC=DCLAS03,  
// DSP=(NEW,KEEP)  
//PR3 PROC ...  
//S3 EXEC PGM=...  
//DD7 DD DSN=&DS,DATACLAS=&DC,DISP=&DSP  
:  
// PEND  
//S1 EXEC PROC=PR3,CD=DCLAS08
```

Aufgelöst sieht dies folgendermaßen aus:

```
//DD7 DD DSN=SP1.DEMO.DATA,  
// DATACLAS=DCLAS08,  
// DISP=(NEW,KEEP)
```

Die Inhalte von &DS und &DSP sind entsprechend der Zuweisungen aus dem SET-Statement. &DC wurde aufgrund der Zuweisung im Prozeduraufruf überschrieben.

SET CURRENT SQLID – DB2 Instruction

```
▶▶"SET CURRENT SQLID=[string]  
[host-var]"
```

Liefert die Möglichkeit, die Rechte eines bestimmten Benutzers für eine Verarbeitung zu nutzen.

SETLANG() – TSO extended Function

SETLANG ()

└sprachcode┘

Übergibt den aktuell eingestellten Sprachcode. Wird einer der gültigen Sprachcodes als Argument übergeben, wird dieser gesetzt

Sprachcode

Ist einer aus der nachfolgenden Tabelle:

Code	Sprache
CHS	Vereinfachtes Chinesisch
CHT	Traditionelles Chinesisch
DAN	Dänisch
DEU	Deutsch
ENP	US Englisch – alles in Großbuchstaben
ENU	US Englisch – Mixed Case
ESP	Spanisch
FRAU	Französisch
JPN	Japanisch
KOR	Koreanisch
PTB	Brasilianisch Portugiesisch

Wie so oft sind Theorie und Praxis nur in der Theorie gleich. In der Praxis müsste die Sprachumgebung installiert sein, damit es tut. Andernfalls erlebt ihr folgendes:

```
/* REXX */
call setlang "DEU"
call gaga /* ist unbekannt */
exit
IRX0300E The message module IRX06DEU could not be loaded
IRX0670E EXECIO error while trying to GET or PUT a record
IRX0300E The message module IRX02DEU could not be loaded
IRX0250E System abend code 013, reason code 00000024.
IRX0300E The message module IRX02DEU could not be loaded
IRX0255E Abend in host command execio or address ...
```

SETMSG – ISPF DM Service

►►"SETMSG MSG(msg-id) "

└COND┘

zeigt eine Meldung mit dem nächsten ISPF-Panel. Auf diesem Weg kann eine Funktion eine Nachricht an ein Panel übergeben, die als SHORT-Message am Terminal erscheint.

MSG(msg-id)

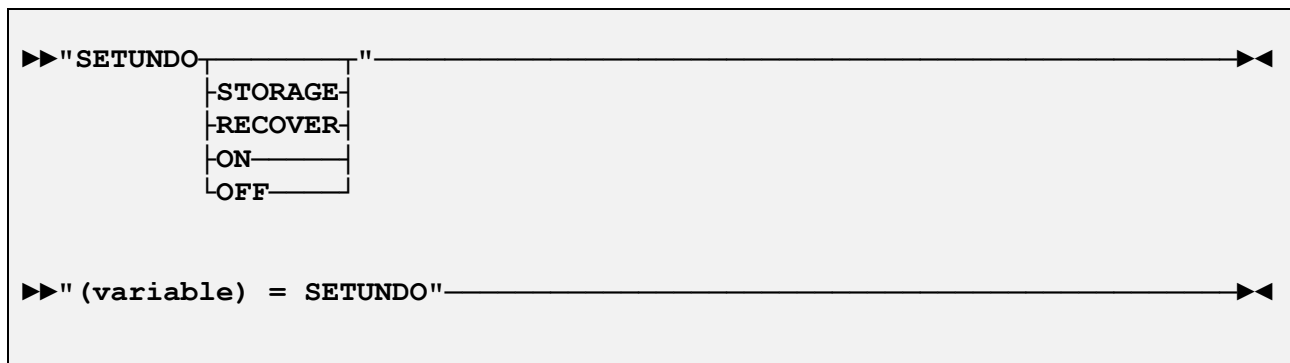
zeigt auf die Nachricht, die am Panel abgebildet werden soll.

COND

regelt, dass die Nachricht nur dann im nächsten Panel gezeigt wird, wenn kein anderweitiger SETMSG-Befehl vorausging.

Return Codes

- 0 Normales Ende
- 4 SETMSG mit COND und es war noch eine Meldung offen
- 12 Nachricht nicht gefunden
- 20 Schwerer Fehler

SETUNDO – ISPF Edit Macro

Der SETUNDO-Befehl ermöglicht die UNDO-Funktion im Macro. Dies hat keine Auswirkung auf die UNDO-Funktion selbst.

SETUNDO wird ignoriert, wenn die UNDO STORAGE Funktion bei Installation nicht gesetzt wurde.

STORAGE

Erlaubt, Änderungen im Speicher rückgängig zu machen.

RECOVER

Erlaubt Änderungen im Recovery-File. Sofern das Recovery ausgeschaltet ist, setzt das SETUNDO RECOVER das Recovery auf ON.

ON

Gleichbedeutend mit STORAGE

OFF

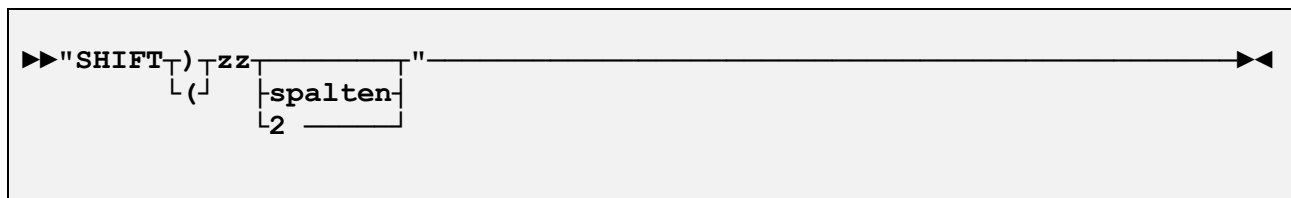
Verhindert SAVES im Speicher. Sofern RECOVERY eingeschaltet ist, benutzt das UNDO-Kommando das Recovery-File

Variable

Enthält den aktuellen Wert von Recovery. Dies kann sein: RECOVER, STORAGE oder OFF.

Return Codes

- 0 Normales Ende
SETUNDO wurde aus- oder eingeschaltet oder der Status blieb unverändert, weil es bereits aus- oder eingeschaltet war.
- 20 Schwerer Fehler. Meist falsche Parameter

SHIFT COLUMN – ISPF Edit Macro

Führt eine Spaltenverschiebung nach links oder rechts durch. Der Befehl wirkt innerhalb der Boundarygrenzen.

Es gelten die gleichen Regeln wie im Column-Shift-Zeilensbefehl des Editors.

)

Spaltenverschiebung nach rechts.

(

Spaltenverschiebung nach links.

zz

Zeilenzeiger kann eine relative Zeilennummer oder ein Label sein.

spalten

Anzahl zu verschiebender Spalten

Beispiel

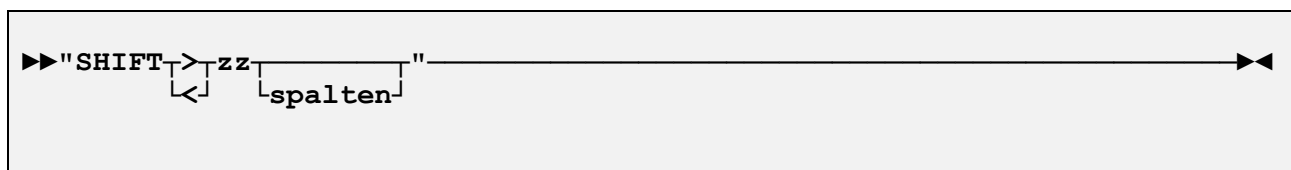
```
"(fline) = linenum .zfrange"
"(lline) = linenum .zlrangle"
do cnt=fline to lline
  "shift )" cnt "3"
end
```

Alle Zeilen zwischen einem vom Anwender ausgewählten Bereich werden inhaltlich um drei Spalten nach rechts verschoben.

Return Codes

0	Normales Ende
12	Falsche Zeilennummer
20	Schwerer Fehler

SHIFT DATA – ISPF Edit Macro



Führt eine Datenverschiebung nach links oder rechts durch. Der Befehl wirkt innerhalb der Boundarygrenzen.

Es gelten die gleichen Regeln wie im Data-Shift-Zeilenbefehl des Editors.

>

Datenverschiebung nach rechts.

<

Datenverschiebung nach links.

zz

Zeilenzeiger kann eine relative Zeilennummer oder ein Label sein.

spalten

Anzahl zu verschiebender Spalten (Default 2)

Beispiel

```

(fline) = linenum .zfrange"
(lline) = linenum .zlrange"
"bounds = "lbnd rbnd
do cnt=fline to lline
    "shift >" cnt "20"
end

```

Daten zwischen .ZFRANGE und .ZLRANGE werden innerhalb der BNDS um maximal 20 Byte nach rechts verschoben. Ist dies nicht ohne Datenverlust möglich, wird an der BND-Grenze Halt gemacht. Zeilen, innerhalb derer die Verschiebung um 20 Byte nicht möglich war, erhalten im Zeilennummernfeld mit dem Hinweis ==ERR>.

Return Codes

0	Normales Ende
12	Falsche Zeilennummer
20	Schwerer Fehler

SIGN() – REXX Built-in-Function

►►SIGN (zahl)◄◄

Vorzeichen für numerische Werte. Übergibt als Rückmeldung -1 für negativ, 0 wenn zahl 0 ist, oder 1

Beispiele

```
say SIGN('5.67')      1
say SIGN('-0.5')      -1
say SIGN('0.0')       0
```

SIGNAL – REXX Instruction

►►SIGNAL ziel◄◄

ON ERROR
FAILURE
HALT
NOVALUE
SYNTAX

NAME ziel

OFF ERROR
FAILURE
HALT
NOVALUE
SYNTAX

Wenn das Ziel der SIGNAL-Instruktion angesprungen wurde, steht die Sourcecode-Zeilenummer, von der weg gesprungen wurde, in einer speziellen Variablen namens SIGL. Dies kann eine wertvolle Hilfe bei Ablaufverfolgung sein, um den Absprungpunkt feststellen zu können.

ziel

Kommt einem GOTO gleich. Davon abgesehen, dass es Ausnahmen geben kann, ist davon dringend abzuraten.

ON | OFF

schaltet die entsprechende Sprungbedingung ein oder aus.

ERROR

Return Code ungleich 0

FAILURE

Negativer Return Code

HALT

Unterbrechung durch PA-1-Taste (Attention-Key).

NOVALUE

Referenz einer nicht definierten Variablen

SYNTAX

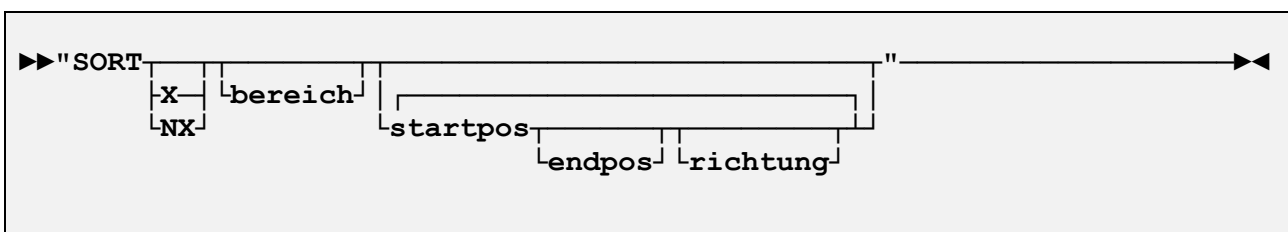
Syntaxfehler

Hinweis

Grundsätzlich lässt sich das SIGNAL-Statement auch wie GOTO in der CLIST benutzen. Auf diese Art soll hier aber nicht eingegangen werden, da dies bei EXECs nicht ungefährlich ist. An dieser Stelle nur den Hinweis: Wer SIGNAL als GOTO missbraucht, hat verdient, was er bekommt.

Ausnahme bildet der Sprung ans Programmende bei Erkennen eines Fehlers. Aber auch hier kann die CALL-Instruktion benutzt werden.

Siehe auch Variable RC und SIGL.

SORT – ISPF Edit Macro

Sort bringt die Datensätze einer Datei in die gewünschte Reihenfolge.

Die Spaltenbereiche, die für die Sortierung berücksichtigt werden, sind abhängig von den Boundarygrenzen. Liegen diese Grenzen nicht auf Satzanfang und -ende, werden die Datensätze zerrissen. Satzteile außerhalb von Boundary bleiben stehen, der Bereich innerhalb verändert sich.

Es ist dementsprechend immer sinnvoll, vor der Sortierung die Boundarygrenzen zu prüfen. Andererseits hält man mit dieser Charakteristik auch ein mächtiges Werkzeug in der Hand. Sollen beispielsweise nebeneinander stehende Zahlenkolonnen sortiert werden, ist dies durch entsprechendes setzen der Boundarygrenzen ohne weiteres möglich.

Für die Festlegung der neuen Reihenfolge der Sätze wird ein Byte-für-Byte-Vergleich durchgeführt. Ist der HEX-Modus eingeschaltet, erfolgt die Sortierung gemäß der HEX-Werte der Zeichen.

X oder NX

Nur ausgeblendete (eXcluded) oder nicht ausgeblendete (Not eXcluded) Zeilen werden sortiert.

bereich

Zwischen zwei Label eingegrenzter Zeilenbereich

startpos

Beginnposition des Sortierfeldes

endpos

Endeposition des Sortierfeldes. Anders als in vielen anderen Sortierprogrammen, wird hier nicht die Länge des Sortierfeldes in Byte, sondern die wirkliche Endeposition angegeben.

richtung

Sortierrichtung.

- A Aufsteigend (ascending)
- D Absteigend (descending)

Beispiele

"sort"

Sortiert die gesamte Datei innerhalb Boundary in aufsteigender Richtung

"sort .a .b d"

Sortiert den Zeilenbereich zwischen den Labeln .A und .B innerhalb Boundary in absteigender Richtung

"sort .a .b 1 10 a 25 30 d"

Der Zeilenbereich zwischen den Labeln .A und .B wird über zwei Sortierfelder in eine neue Reihenfolge gebracht. Hauptsortierkriterium sind die Spalten 1 bis 10 in aufsteigender, Zweitkriterium die Spalten 25 bis 30 in absteigender Reihenfolge.

Return Codes

- 0 Normales Ende
- 4 Die Zeilen waren bereits in der gewünschten Sortierfolge
- 8 Keine zu sortierenden Datensätze
- 16 Zu wenig Speicher um SORT auszuführen
- 20 Schwerer Fehler

SOURCELINE() – REXX Built-in-Function

►►SOURCELINE([zahl])◄◄

Wird in der Klammer kein Wert angegeben, wird die Zeilennummer der letzten Zeile im Sourcecode ausgegeben. Enthält zahl eine gültige Zeilennummer, kleiner oder gleich der größten vorhandenen Zeilennummer innerhalb des Sourcecodes, wird deren Zeileninhalt ausgegeben.

Sofern die REXX compiled ist und der Sourcecode nicht eingebettet wurde, liefert die Funktion SOURCELINE() eine 0.

Beispiele

```
say SOURCELINE()      10
say SOURCELINE(2)     '/* ERSTELLT: 02.2000 */'
```

```
:
do x=2 while strip(sourceline(x)) <> ""
    say sourceline(x)
end
:
```

SPACE – JCL DD Parameter

►►SPACE=(TRK, (pri, sek[, dir][, RLSE][, CONTIG])◄◄

TRK
CYL
blk

TRK | CYL

Die Zuordnung erfolgt in Spuren (TRACKS) oder Zylindern.

blk

Durchschnittliche Blocklänge. Entgegen der irrigen Annahme darf hier nicht "BLK", sondern nur die Blockgröße stehen (wird in Spuren umgerechnet).

pri

Erstzuweisung mit der angegebenen Größenordnung (Spuren, Zylinder). Kann vom System in bis zu fünf Teilbereiche (EXTENTS) zerlegt werden.

sek

Erweiterung, wenn die primäre Menge nicht ausreicht. Eine Erweiterung kann so oft durchgeführt werden, bis maximal 16 Teilbereiche einer Datei je Datenträger existieren. Ist keine Sekundärmenge angegeben, kann nicht erweitert werden. Dies erzeugt einen Abbruch (Dateiüberlauf), wenn die Primärmenge nicht ausreicht.

dir

Wird dieser Parameter nicht angegeben, wird eine sequentielle Datei eingerichtet. Wird ein Wert größer als 0 eingegeben, wird eine PO-Datei erzeugt, wobei dir-blk die Anzahl der anzulegenden Directory-Blocks enthält. Pro Directoryblock (256 Byte lang) können etwa 5 Membereinträge erfolgen. Die Größe des Directorys kann im Nachhinein nicht geändert werden.

RLSE

Nicht benutzter Speicherplatz wird bei CLOSE für eine Datei freigegeben (problematisch im Falle eines Abbruchs).

CONTIG

Bestimmt, dass der für die primäre Menge zu reservierende Speicherplatz zusammenhängend sein muss (keine EXTENTS). Ist dies wegen Platzmangels auf einer Platte nicht möglich, wird der Job abgebrochen.

SPACE() – REXX Built-in-Function

```
►►SPACE(string,zahl([füllzeichen])◄◄
```

Formatiert die durch BLANK abgegrenzten Worte in einem String mit durch zahl angegebenen Füllzeichen zwischen den Worten.

Beispiele

say SPACE('Ist ja toll',2)	'Ist ja toll'
say SPACE('Ist ja toll',1,'-')	'Ist-ja-toll'
say SPACE('Ist ja toll',0)	'Istjatoll'

STATS – ISPF Edit Macro

```
►►"(variable) = STATS"◄◄  
  
►►"STATS = [ON]  
[OFF]"◄◄
```

Setzt oder liefert den aktuellen Wert für STATS. Ist STATS ON gesetzt, werden die Memberstatistiken in einer PO-Datei mit gepflegt, sobald ein Member gesichert wird.

=STATS

Der aktuelle Wert von STATS wird in einer Variablen hinterlegt.

STATS=

Der STATS-Modus wird entsprechend im Edit Profil hinterlegt.

Return Codes

0	Normales Ende
20	Schwerer Fehler

STATUS – TSO Command

Der STATUS-Befehl sein hier nur der Vollständigkeit halber erwähnt. Generell könnten wir damit Informationen Jobs, Started Tasks oder LOGON-Sessions gewinnen. Sinnvollerweise würde man derartiges aber über die REXX-API von SDSF bewerkstelligen.

Beispiele hierzu sind im Band 1 zu finden.

STEPLIB – JCL Special DD Statement

```
//STEPLIB DD parameter
```

Die STEP-DD-Anweisung zeigt auf eine oder mehrere Lademodul-Bibliotheken, aus denen Programme aufzurufen sind. Die Definition ist nur für den STEP gültig, hinter dessen EXEC-Anweisung sie kodiert ist. Sie wird gebraucht, wenn die Programme über die LINKLIST nicht gefunden werden, oder wenn sie gezielt aus einer bestimmten Bibliothek geladen werden sollen (z.B. beim Testen).

Es können maximal so viele STEPLIB-DD-Anweisungen kodiert werden, als es Programmaufrufe gibt. muss vor der ersten EXEC-Anweisung liegen.

STORAGE() – TSO extended Function

```
►►STORAGE (adresse [länge] [daten])
```

Übergibt eine Anzahl Byte von einer angegebenen Hex-Adresse im Arbeitsspeicher. Die Funktion kann in einem MVS-, TSO- und NON-TSO-Adressraum benutzt werden.

Es können nur Bereiche gelesen werden, die nicht als FETCH PROTECTED definiert sind.

Geschrieben werden kann nur in ungeschützte Speicherbereiche.

Generell kann die Funktion auf 31-Bit-Adressen (=1 bis 8 Hex-Zeichen) oder auf 64-Bit-Adressen (=9 bis 17 Hex-Zeichen) zugegriffen werden. Zu beachten ist, dass beim Umgang mit 64-Bit-Adressen gegebenenfalls die Rechengenauigkeit zu ändern ist (siehe Instruction NUMERIC DIGITS).

Bei der Benutzung von STORAGE() sollte man generell wissen, was man tut. Hier detailliert zu beschreiben, was möglich ist, würde jedem Rahmen sprengen. Was man wo vorfinden kann, ist im Manual für die Beschreibung der Speicherstrukturen und/oder im REXX Reference Manual unter STORAGE() nachzulesen.

Beispiele

```
CVT = STORAGE(10,4)
```

Liest die Startadresse der Common Vector Table aus dem Speicher

```
Replace = STORAGE("0035D41F", , "TSO/E REXX")
```

Schreibt den String "TSO/E REXX" im Speicher an die entsprechende Adresse. Rückgabewert ist der Speicherinhalt vor dem Überschreiben.

STORCLAS – JCL DD Parameter

```
▶▶STORCLAS=storageclass▶▶
```

Diese Angabe wird nur in Verbindung mit SMS berücksichtigt. Die Storage-Klasse liefert über einen definierten Namen Angaben wie UNIT= und VOL= bei einer NICHT-SMS-gemanagten Datei.

Mögliche Namensangaben sind bei der für das Space-Management zuständigen Stelle zu erfragen.

SYSOUT – JCL DD Parameter

```
▶▶SYSOUT=[klasse]
          [*,
          [DEST=name
          [,COPIES=anzahl
          [,OUTLIM=anzahl
          [,HOLD=[YES
                  NO]
          ]
          ]
▶▶
```

klasse| *

Als klasse sind A bis Z und 0 bis 9 zugelassen. Wird als klasse Stern(*) angegeben, wird die Ausgabeklasse aus dem Operanden "MSGCLASS=" der Job-Anweisung benutzt.

Wird angegeben SYSOUT=(,INTRDR), wird die Ausgabe direkt an den Internal Reader des JES weitergeleitet. Auf diese Weise kann ein Step die Job-Control-Anweisungen für einen nächsten Job übergeben (gegebenenfalls in Abhängigkeit zur Erfüllung einer entsprechenden COND-Bedingung) und so eine gewisse Automation erreicht werden.

DEST=name

Benennt den lokalen Drucker, über den die Ausgabe erfolgen soll.

COPIES=anzahl

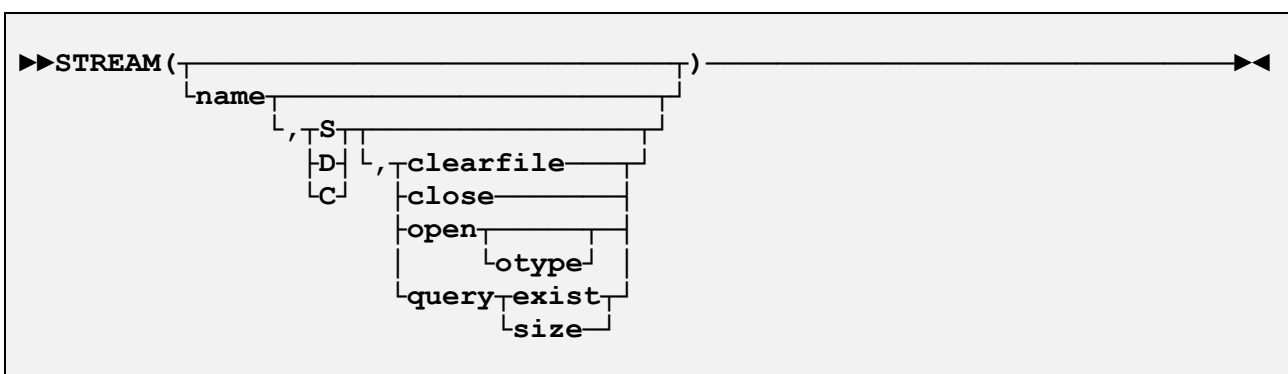
Bestimmt die Anzahl der Exemplare, die auf Drucker ausgegeben werden sollen, nicht die Anzahl der Kopien. COPIES=1 ist gleichzusetzen mit keiner Angabe.

OUTLIM=anzahl

Mit dem Parameter OUTLIM= wird die maximal zulässige Anzahl von Ausgabesätzen für die SYS-OUT-Datei festgelegt.

HOLD=

Mit HOLD=YES wird die Ausgabe in die OUTPUT-QUEUE gestellt, aber nicht ausgegeben. Der HOLD-Status kann durch den Terminalbenutzer, oder durch den Operator aufgehoben werden. Fehlt der Operand HOLD, so ist HOLD=NO der Standardwert.

STREAM() – UNIX Function

Liefert den Status eines Datenstroms oder das Ergebnis eines Befehls.

Name

Der Name eines Streams. Die Angabe ist case-sensitiv.

S

Gibt den Status des Streams aus. Möglich sind ready, error oder unknown (fall kein OPEN erfolgte).

D

Wie S mit einer Ausnahme: Wenn error zurückgegeben wird steht im Anschluss ein Begleittext, in welchem die Fehlernummer und der Reason Code enthalten sind.

C

Commandausführung. Das Command wird als dritter Parameter übergeben.

clearfile

Ist nur für persistente Streams zur Initialisierung vor einer Schreiboperation angebracht.

close

Schließt den Stream und schreibt EOF

open replace

Setzt vor dem Schreiben EOF zur Initialisierung.

open append

Führe eine Fortschreibung durch

open read

Öffnet zum Lesen

open write

Öffnet zum Schreiben

query exist

Gibt bei Vorhandensein den vollen Pfadnamen zurück. Die Datei muss hierzu nicht geöffnet werden.

query size

Ermittelt die Größe

Hinweis

Es ist sicher zu stellen, dass das richtige Environment geladen und eingestellt ist.

Beispiel

```
Say stream("myfile.txt","c","size")          /* gibt die Dateigröße aus */  
Say stream("myfile.txt","c","query exist") /* Full pathname */
```

STRIP() – REXX Built-in-Function

```

▶▶STRIP(string,option,char)

```

Entfernt führende oder am Ende eines Strings stehende Zeichen entsprechend der gewählten Option.

Wird außer des Strings keine Angabe gemacht, wird als Option B(oth)=führend und nachfolgend, sowie als char BLANK angenommen.

option

Mögliche Optionen sind:

- B Both (Führend und Nachfolgend) ist Default.
- L Leading (Führend)
- T Trailing (Nachfolgend)

Beispiele

```

say STRIP(' Na sowas ',B)      'Na sowas '
say STRIP(' Na sowas ')      'Na sowas '
say STRIP(' Na sowas ',L)     'Na sowas '
say STRIP(' Na sowas ',T)     ' Na sowas '
say STRIP('0038.400',,0)      '38.4 '

```

```

/* REXX
*/
/* Erwartet wird eine vollqualifizierte DSN-Angabe.
   Falls keine Apostrophe mitgeliefert wurden, werden sie gesetzt.
   Wir wollen aber keine unnötigen Abfragen kodieren
*/
Dsn = ""strip(arg(1),,"")""
Say dsn
:

```

SUBCOM – TSO extended Command

```

▶▶"SUBCOM umgebungsname"

```

Um zu gewährleisten, dass die gewünschte Systemumgebung auch wirklich verfügbar ist, kann die SUBCOM-Instruktion genutzt werden.

Beispiel

SUBCOM ISPEXEC

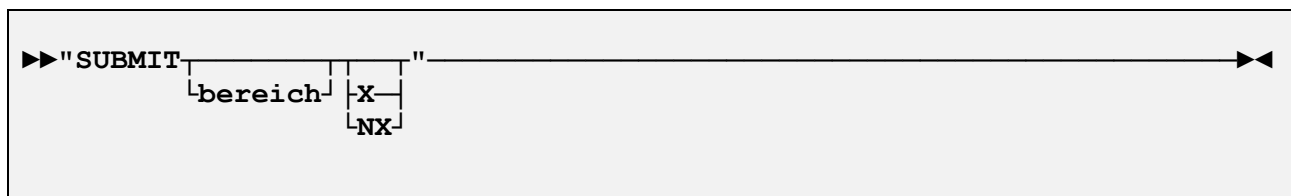
Die Verfügbarkeit von ISPEXEC wird geprüft. Kann ISPEXEC angesprochen werden, wird die Variable RC auf 0 gesetzt, ansonsten auf 1. Soll beispielsweise eine Datei im Ablauf einer EXEC editiert werden und wenn möglich, dabei auf den TSO-EDIT verzichtet werden, könnte das folgendermaßen aussehen:

```
/* rexx */
parse upper arg datei .
subcom ispexec
if rc=0 then address ispexec "edit dataset("datei")"
else "edit" datei /* Standard TSO-Line-Editor */
exit
```

Return Codes

- 0 Umgebung vorhanden
- 1 Umgebung nicht vorhanden (muss gegebenenfalls geladen werden)

SUBMIT – ISPF Edit Macro



Übergibt die Daten an den Internal Reader des JES. SUBMIT kann mit SUB abgekürzt werden.

bereich

Zeilenbereich, der von zwei Labels, oder zwei relativen Zeilennummern eingegrenzt wird.

X | NX

Berücksichtigt nur Ein- oder ausgeblendete Zeilen.

Beispiele

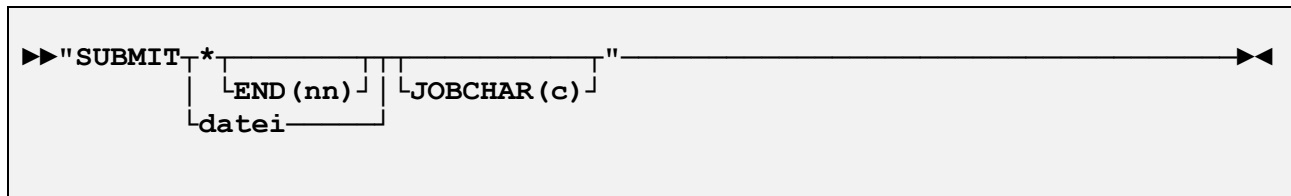
`"submit"`

Alle Zeilen, die der Editor im virtuellen Adressraum hält, werden an den Internal Reader zur Ausführung im Batch übergeben.

Return Codes

0	Normales Ende
20	Schwerer Fehler

SUBMIT – TSO Command



Übergibt die Daten an den Internal Reader des JES. SUBMIT kann mit SUB abgekürzt werden.

*

Die an den Internal Reader zu übergebende JCL wird im STACK erwartet.

END(nn)

Endemerkmak. Ist ein beliebiger 2-Byte-Begriff, welche das Ende der JCL definiert, wenn er im Stack gelesen wird (wirkt wie ein EOF).

Datei

Die auszuführende JCL liegt in der angegebenen Datei.

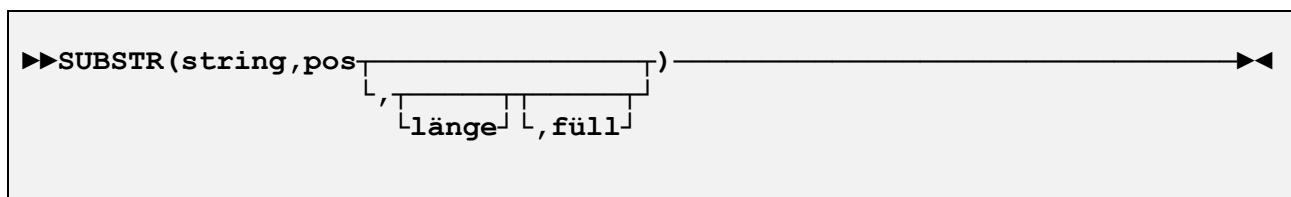
JOBCHAR(c)

Es kann ein Job-Character-Byte angegeben werden, welches die UserID zum Jobnamen ergänzt.

Return Codes

- ```
0 JCL erfolgreich an den Internal Reader übergeben
12 Nicht erfolgreich. Eine Fehlermeldung wurde ausgegeben
```

## SUBSTR() – REXX Built-in-Function



Übergibt den Teil einer Zeichenfolge beginnend ab pos in der angegebenen Länge. Zeigt die Längenangabe über die Länge des Strings hinaus, wird mit dem angegebenen Füllzeichen aufgefüllt

(Standard ist BLANK). Wird keine Länge angegeben, wird die Zeichenfolge bis zum Ende übergeben.

### Beispiele

```
say SUBSTR('12:45:33',1,2) '12'
say SUBSTR('12:45:33',4,2) '45'
say SUBSTR('12:45:33',7,2) '33'
say SUBSTR('12:45:33',3) '45:33'
```

Bitte beachtet hier die Möglichkeiten des Parsings:

```
Parse var '12:45:33' std ':' min ':' sec
Say std '12'
Say min '45'
Say sec '33'
```

## SUBWORD() – REXX Built-in-Function

►►SUBWORD(string,pos[ ,anzahl])◀◀

Übergibt eine Teilzeichenfolge in Form von Worten im Sinne von REXX. Das mit pos angegebene Wort bildet das erste Wort der Teilzeichenfolge, ab dem die angegebene Anzahl von Worten übernommen wird.

Die gebildete Zeichenfolge wird immer ohne führende oder nachfolgende BLANKS geliefert, beinhaltet aber alle Leerstellen zwischen den Worten.

### Beispiele

```
say SUBWORD('Bitte ein Bier',2) 'ein Bier'
say SUBWORD('Bitte ein Bier',2,1) 'ein'
say SUBWORD('Bitte ein Bier',4) ' '
```

## SYMBOL() – REXX Built-in-Function

►►SYMBOL(name)◀◀

Meldet den Status des durch name angegebenen Symbols zurück. Dabei bedeuten:

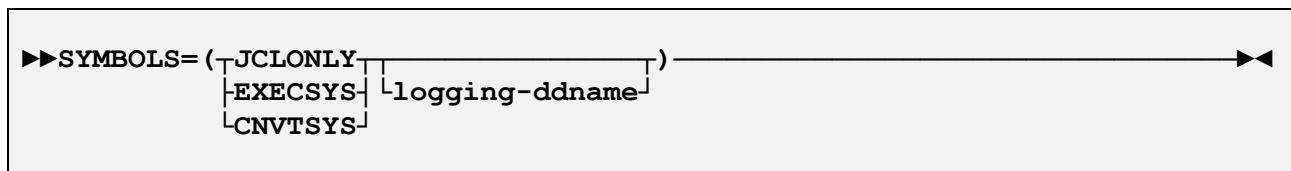
|     |                             |
|-----|-----------------------------|
| BAD | Unbekannt                   |
| VAR | Variable mit Wertzuordnung  |
| LIT | Variable ohne Wertzuordnung |

### Beispiele

Angenommen: DROP A.3;J=3

|                   |     |                   |
|-------------------|-----|-------------------|
| say SYMBOL('J')   | VAR |                   |
| say SYMBOL(J)     | LIT | enthält 3         |
| say SYMBOL('A.J') | LIT | enthält A.3       |
| say SYMBOL(2)     | LIT | eine Konstante    |
| say SYMBOL('*')   | BAD | ungültiges Symbol |

## SYMBOLS – JCL DD Parameter



Informiert JES, wie die Substituierung von Variablen vorstattengehen soll.

### JCLONLY

Symbole, die im Export-Parameter definiert sind, werden substituiert

### EXECSYS | CNVTSYS

Die Interpretation der Symbols kann entweder auf dem ausführenden System (EXECSYS) oder auf demjenigen, auf dem die Conversion-Phase durchlaufen wird (CNVRTSYS)

### Beispiel

#### Beispiel

```

//MYEXPR1 EXPORT SYMLIST=(USER,MVS)
// SET USER=&SYSUID
// SET MVS=&SYSNAME
//TSO1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=CNVTSYS
 EX '&USER..DEMO.EXEC(SAGWAS)' 'Heinz ist doof' EXEC
//*
//TSO2 EXEC PGM=IKJEFT01
//SYSEXEC DD DISP=SHR,DSN=&USER..DEMO.EXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=CNVTSYS

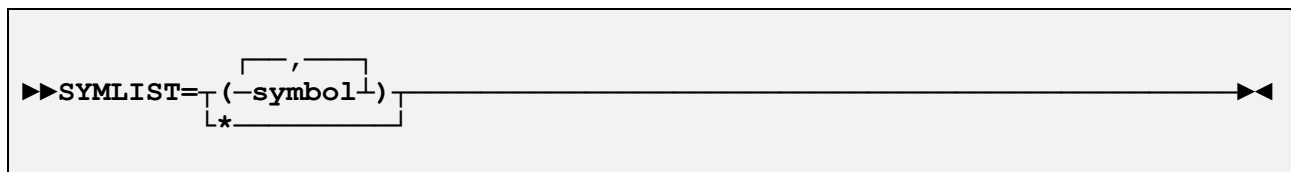
```

```

 %sagwas 'Achtung: Jetzt kommen die Profile-Member'
 listd '&user..&mvs..ispf.profile' m
/*
//TSO3 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
 listd '&user..&sysname..ispf.profile' m

```

## SYMLIST – JCL DD Parameter



Gibt dem Internal Reader eine Liste von Symbols bekannt, die substituiert werden sollen. Der Stern(\*) zeigt an, dass alle Symbols betroffen sind (EXPORT, JES, JCL)

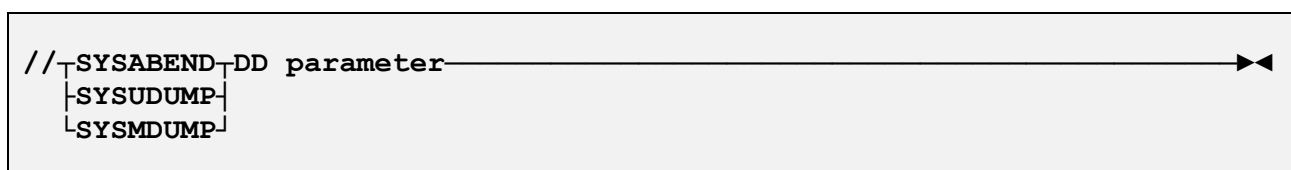
### Beispiel

```

//MYEXPR1 EXPORT SYMLIST=(USER,MVS)
// SET USER=&SYSUID
// SET MVS=&SYSNAME
//TSO1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=CNVTSYS
 EX '&USER..DEMO.EXEC(SAGWAS)' 'Heinz ist doof' EXEC
/*
//TSO2 EXEC PGM=IKJEFT01
//SYSEXEC DD DISP=SHR,DSN=&USER..DEMO.EXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=CNVTSYS
 %sagwas 'Achtung: Jetzt kommen die Profile-Member'
 listd '&user..&mvs..ispf.profile' m
/*
//TSO3 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
 listd '&user..&sysname..ispf.profile' m

```

## SYSABEND | SYSMDUMP | SYSUDUMP – JCL Special DD Statement



Sind für DUMP-Ausgaben zuständig

### **SYSABEND**

Dumpte User- und Systembereiche. Unter anderem auch LSQA, IOS

### **SYSMDUMP**

Enthält die System-Areas. Der Dump ist unformatiert und muss mit einem Tool wie IPCS (oder ein vergleichbares von einem Fremdanbieter) gelesen werden. Es ist nicht hilfreich, für SYSMDUMP eine Zuweisung nach SYSOUT zu machen.

### **SYSUDUMP**

Umfasst die User-Areas.

## **SYSCPUS() – TSO extended Function**

►►SYSCPUS(stem)◄◄

Liefert in einem Variablenstamm Informationen über alle CPUs, welche ONLINE sind.

### **Beispiele**

```
/* REXX
*/
erg = SYSCPUS("CPUS.")
say "Ergebnis der CPU-Abfrage:" erg
say "Anzahl der ONLINE-CPUs. :" CPUS.0
do i = 1 to CPUS.0
 say "CPU" i "hat CPU Info" CPUS.i
end
```

## **SYSDSN() – TSO extended Function**

►►SYSDSN(dateiname)◄◄

Teilt durch eine der nachfolgenden Rückmeldungen mit, ob die Datei vorhanden und verfügbar ist. Die Angabe in der Klammer kann eine sequentielle Datei, eine PO-Datei oder ein Member einer PO-Datei sein.

| Meldung | Bedeutung              |
|---------|------------------------|
| OK      | Datei/Member vorhanden |

|                                                 |                          |
|-------------------------------------------------|--------------------------|
| MEMBER SPECIFIED BUS DATASET IS NOT PARTITIONED | Selbsterklärend          |
| DATASET NOT FOUND                               | Selbsterklärend          |
| ERROR PROCESSING REQUESTED DATASET              | Kommt faktisch nie vor   |
| PROTECTED DATASET                               | Datei ist RACF-geschützt |
| VOLUME NOT ON SYSTEM                            | Platte nicht gemounted   |
| INVALID DATASET NAME, dateiname                 | Selbsterklärend          |
| MISSING DATASET NAME                            | Kein Dateiname angegeben |
| UNAVAILABLE DATASET                             | Datei belegt             |

Ist eine Datei zum Zeitpunkt der SYSDSN-Abfrage von DFHSM ausgelagert, bewirkt SYSDSN einen RECALL.

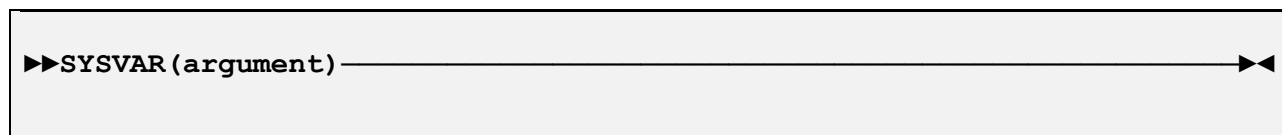
### Beispiel

```

/* REXX
 Überprüfung auf Verfügbarkeit
 einer beliebigen Datei in Schleife
*/
do forever
 say 'Datei vollqualifiziert eingeben'
 pull datei
 if length(datei)=0 then leave
 if sysdsn("'"datei"'") = 'OK' then
 erg= datei 'verfügbar'
 else say datei 'nicht verfügbar'
end
exit

```

## SYSVAR() – TSO extended Function



Übergibt eine Vielzahl von Informationen vom MVS oder TSO an die Prozedur. Einige der möglichen Argumente werden im Folgenden in Klassen eingeteilt beschrieben.

| Benutzerinformationen  |                                              |
|------------------------|----------------------------------------------|
| SYSPREF                | Dateiprefix aus TSO-Benutzerprofil           |
| SYSPROC                | Name der LOGON-Procedure                     |
| SYSUID                 | Benutzerkennung                              |
| Terminal-Informationen |                                              |
| SYSLTERM               | Anzahl der Zeilen am Bildschirm              |
| SYSWTERM               | Anzahl der Spalten am Bildschirm             |
| Prozedurablauf         |                                              |
| SYSENV                 | Laufzeitumgebung (BACK oder FORE)            |
| SYSISPF                | ACTIVE wenn ISPF verfügbar, sonst NOT ACTIVE |
| SYSNEST                | YES wenn ein UPROG, ansonsten NO             |
| SYSPCMD                | Enthält den letzten ausgeführten TSO-Befehl  |

| Systeminformationen |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| SYSRACF             | RACF-Status. Möglich sind:<br>AVAILABLE<br>NOT AVAILABLE<br>NOT INSTALLED |
| SYSTSOE             | Version des TSO im Format 3010 (Vers.3Rel.1)                              |

## TABS – ISPF Edit Macro

|                                                        |
|--------------------------------------------------------|
| ▶▶" (var1,var2) = TABS"_____▶▶                         |
| ▶▶"TABS = [ON] [OFF] [tabnzeichen] [ALL] [STD]"_____▶▶ |

Übergibt oder setzt den Modus, in dem der Tabulator arbeitet.

### =TABS

Füllt die Variablen mit der aktuellen Einstellung für den Tabulator.

var1enthält ON oder OFF, var2 enthält ALL oder STD

### TABS=

Der TABS-Modus wird entsprechend im Edit Profil hinterlegt.

### ON | OFF

Der Tabulator (hard- oder softwaremäßig) wird ein- oder ausgeschaltet.

### tabnzeichen

Zeichen als Sprungbefehl für den logischen TAB. Wird eines der folgenden benutzt (- = ' < , & ( + ! "), müssen Apostrophe gesetzt werden.

### ALL

Zeichen an der Halteposition werden durch TAB-Steuerzeichen ersetzt.

### STD

Zeichen ungleich Blank an der Halteposition werden nicht durch Steuerzeichen ersetzt. Die Halteposition in einer Zeile, die durch Daten überlagert wird, kann vom Tabulator nicht mehr angesprungen werden.

**Return Codes**

|    |                 |
|----|-----------------|
| 0  | Normales Ende   |
| 20 | Schwerer Fehler |

**TABSLINE – ISPF Edit Macro**

```
▶▶"(variable) = TABSLINE"—————▶▶

▶▶"TABSLINE = daten"—————▶▶
```

Liefert den aktuellen Inhalt der Tabulatorzeile, oder belegt sie neu.

**=TABSLINE**

Hinterlegt in der Variablen den aktuellen Inhalt der Tabszeile.

**TABSLINE=**

Sorgt für die Neubelegung der Tabszeile.

**daten**

Nur Stern(\*), Minus(-) und Underscore(\_) können eingesetzt werden. Minus und Underscore kann zur Positionierung des Cursors innerhalb der Tabs-Haltepunkte eingesetzt werden. Für daten ist möglich:

- einfache oder eingegrenzte Zeichenketten, oder Variable
- Schablonen (< spalte,zeichenfolge >)
- Mischformate (string1 + string2,schlüsselwort + ...)
- Schlüsselworte
  - LINE [zz]      Daten der aktuellen oder angegebenen Zeile
  - MASKLINE      Daten der Maskenzeile
  - TABSLINE      Daten der TABS-Zeile

**Beispiele**

```
"tabslime = < 45 ,*, 70 ,*"
```

Setzt Tabulatorhaltepunkte auf Spalte 45 und 70.

```
"tabslime = tabslime + <45,' '"
```

Entfernt den Tabulatorstop auf Spalte 45.

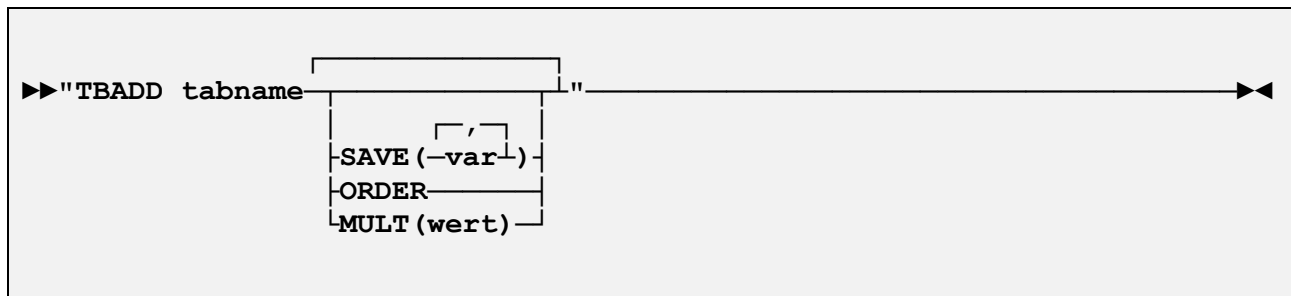
```
"tabslime = ''"
```

Die Tabulatorzeile wird zurückgesetzt.



**Return Codes**

|    |                         |
|----|-------------------------|
| 0  | Normales Ende           |
| 4  | Daten abgeschnitten     |
| 16 | Falsche Daten ignoriert |
| 20 | Schwerer Fehler         |

**TBADD – ISPF DM Service**

Fügt einer Tabelle eine neue Zeile hinzu. Der neue Satz wird entweder hinter dem Satz eingefügt, auf den der CRP aktuell zeigt, oder, sofern die Tabelle sortiert ist, hinter den Satz, der den Sortierkriterien entsprechend der Richtige ist.

Der CRP wird in diesem Falle auf den neu eingefügten Satz gestellt.

Gespeichert werden die Inhalte aller Variablen im Functionpool, deren Namen gleichnamig den Feldern der Tabelle sind.

Bei Tabellen mit Schlüsseln wird gewährleistet, dass eine Eintragung in die Tabelle nur erfolgt, wenn der Schlüssel einmalig ist, andernfalls wird die Einarbeitung des Satzes abgelehnt.

Bei Tabellen ohne Schlüssel kann die Überprüfung auf doppelt vorhandene Sätze in der Tabelle nicht erfolgen.

**Tabname**

benennt die Tabelle, zu der ein Satz hinzugefügt werden soll.

**SAVE(var..**

erlaubt das Hinzufügen von Feldern an einen Datensatz, die bei Erstellung der Tabelle nicht vorgesehen waren. Diese Felder sind Bestandteil nur desjenigen Satzes, für den der TBADD-Befehl gilt, nicht der gesamten Tabelle.

**ORDER**

Sofern die Tabelle sortiert ist, wird der Datensatz den Sortierkriterien entsprechend in der Tabelle eingereiht. Ist die Tabelle nicht sortiert, wird die Angabe ignoriert.

Wird ORDER bei einer sortierten Tabelle nicht angegeben, wird der Eintrag in die Tabelle auch nicht den Sortierkriterien gemäß durchgeführt. Um die Tabellensätze wieder in die gewünschte Reihenfolge zu bringen, ist ein erneuter Sortierbefehl nötig.

**MULT(wert)**

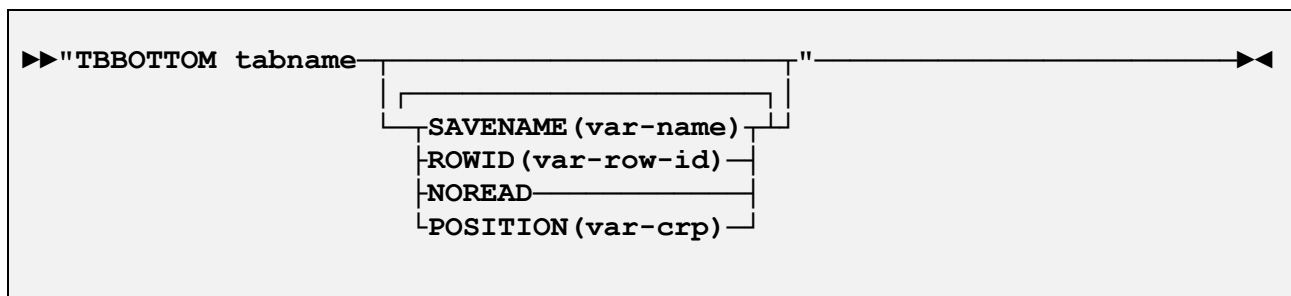
sorgt für höhere Verarbeitungsgeschwindigkeit. Wird über die mitgelieferte Zahl die etwaige Anzahl hinzuzufügender Sätze mitgeteilt, sorgt ISPF bei der ersten Ausführung von TBADD dafür, dass entsprechend mehr Speicherplatz angefordert wird, um die Tabellensätze einzuarbeiten.

Wird TBADD in einer Schleife verarbeitet, wird der Operand MULT() solange nicht berücksichtigt, als noch genügend Speicherplatz zur Einarbeitung von Tabellensätzen vorhanden ist.

Der Standardwert für den benötigten Freiplatz zum Einarbeiten von Tabellensätzen ist 1, sofern dies bei der Installation von ISPF nicht anderweitig definiert wurde.

**Return Codes**

- 0     Normales Ende
- 4     MULT() angegeben aber nur Speicherplatz für einen Satz nötig
- 8     Satz mit gleichen Schlüsselfelder existiert bereits. Der CRP wird auf TOP gesetzt. Dieser RC kann nur bei Tabellen mit Schlüsseln auftreten.
- 12    Tabelle nicht geöffnet
- 16    Fehler (siehe numerische Restriktionen für TBSORT). Dieser RC kann nur bei sortierten Tabellen auftreten
- 20    Schwerer Fehler

**TBBOTTOM – ISPF DM Service**

Setzt den CRP auf den letzten Satz in der Tabelle.

**tablename**

benennt die Tabelle, die von TBBOTTOM betroffen wird.

**SAVENAME(var-name)**

Variable, in der alle Namen von Extension-Variablen stehen, die für den aktuellen Satz gespeichert sind. Die Variablennamen sind durch BLANK voneinander getrennt.

**ROWID(var-rowid)**

legt in der angegebenen Variablen eine eindeutige Identifikationsnummer des Datensatzes ab, auf den der CRP zeigt (vor Befehlsausführung). Dieser Wert kann im Operanden ROW() im TBSKIP-Befehl angegeben werden, um den Datensatz zu einem späteren Zeitpunkt wieder aufsuchen zu können. Der Wert wird bei TBCLOSE oder TBSAVE nicht abgespeichert.

**NOREAD**

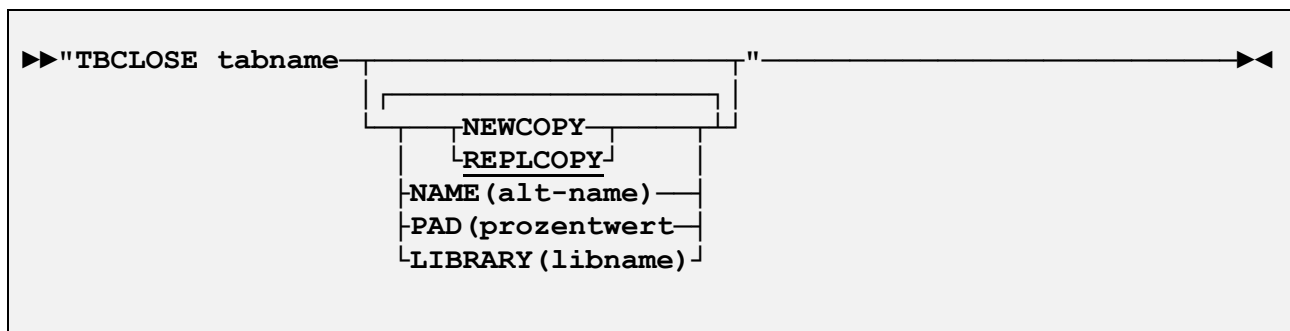
bewirkt, dass, abweichend vom Standard, die Feldinhalte des aktuellen Satzes nicht in gleichnamige Variable übertragen werden.

**POSITION(var-crp)**

die Variable wird mit der Identifikationsnummer des aktuellen Satzes in der Tabelle gefüllt (nach Befehlsausführung).

**Return Codes**

- 0     Normales Ende
- 8     Tabelle leer. CRP steht auf TOP
- 12    Tabelle nicht geöffnet
- 20    Schwerer Fehler

**TBCLOSE – ISPF DM Service**

Schließt eine Tabelle. War sie mit NOWRITE eröffnet, wird lediglich die Kopie der Tabelle im virtuellen Speicher gelöscht. Wurde der OPEN mit WRITE durchgeführt, wird die Tabelle auf Platte zurückgeschrieben.

Wird eine Tabelle im SHARED-Modus geöffnet (Split-Screen), führt ISPF einen Benutzungszähler (use counter). Ist dieser Zähler bei TBCLOSE nicht 0, wird anstelle des TBCLOSE ein TBSAVE durchgeführt. Dadurch bleibt die Tabelle im virtuellen Speicher erhalten, bis ein CLOSE-Befehl ausgeführt wird, zu dessen Zeitpunkt der Zähler auf 0 steht (erst dann wird die Tabelle auf Platte zurückgeschrieben und im virtuellen Speicher gelöscht).

**Tabname**

benennt die Tabelle, die geschlossen werden soll.

**NEWCOPY**

bewirkt, dass die Tabelle an das Ende der Ausgabebibliothek angehängt wird, auch wenn ein Update am alten Ort möglich wäre. Dadurch wird ausgeschlossen, dass die Altversion überschrieben wird.

**REPLCOPY**

die Tabelle wird an ihren alten Platz zurück geschrieben. Ist das bestehende Member kleiner als die Tabelle im virtuellen Speicher, oder in der Ausgabebibliothek nicht vorhanden, wird die Tabelle am Ende der Ausgabedatei angehängt.

**NAME(alt-name)**

Die Tabelle wird in der Ausgabebibliothek unter einem anderen (alternate) Namen abgespeichert. Ist eine gleichnamige Tabelle in der Ausgabebibliothek bereits vorhanden, wird sie überschrieben.

Ist in der Ausgabebibliothek eine Tabelle mit dem Originalnamen vorhanden, bleibt diese von etwaigen Veränderungen unberührt.

**PAD(prozentwert)**

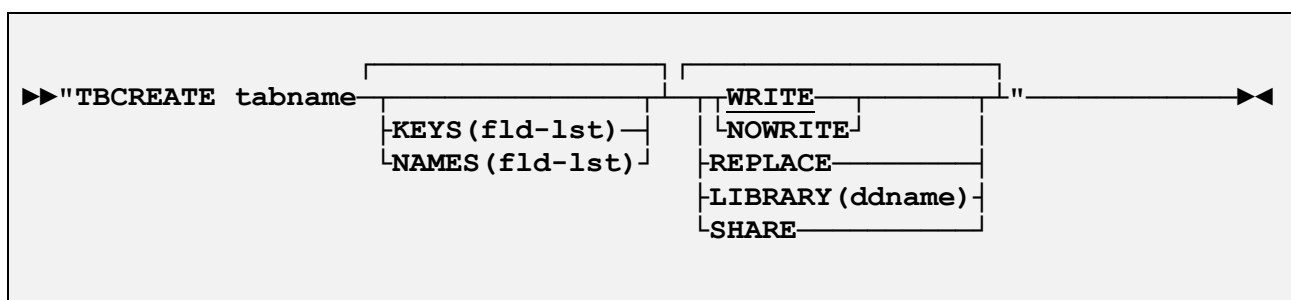
liefert eine prozentuale Größenangabe im Verhältnis zur aktuellen Größe der Tabelle. Diese Größe wird als Vorhalt bereitgestellt, wenn die Tabelle neu an das Ende der Ausgabebibliothek angehängt wird. Der Parameter wird ignoriert, wenn die Tabelle an ihren alten Platz in der Ausgabebibliothek zurück geschrieben werden kann.

**LIBRARY(ddname)**

verweist beim Schreiben auf einen anderen als den standardgemäßen DD-Namen (ISPTABL).

**Return Codes**

- 0      Normales Ende
- 12     Tabelle nicht geöffnet
- 16     Alternative Ausgabebibliothek nicht zugewiesen
- 20     Schwerer Fehler

**TBCREATE – ISPF DM Service**

Erstellt eine Tabelle im Arbeitsspeicher und öffnet sie zur Verarbeitung.

Die Namen aller Felder (die in der Verarbeitung mit gleichnamigen Variablen korrespondieren) werden angegeben.

SAVE-Variable können an einen Tabellensatz angehängt werden, wenn der Satz geschrieben wird.

Ein oder mehrere Felder können als Schlüsselfelder definiert werden. Wird kein Schlüssel definiert, kann für Update-Operationen ausschließlich der CRP genutzt werden.

Der Vollständigkeit halber sei noch erwähnt, dass es auch Tabellen ohne Schlüssel- und Namensfelder gibt. Auf diese Variante wird hier aber nicht näher eingegangen.

**tabname**

ist der Name der Tabelle, die erstellt werden soll.

**KEYS(fld-lst)**

Namen der Felder, die den Schlüssel der Tabelle ergeben. Fehlt diese Angabe werden keine Schlüsselfelder gebildet.

**NAMES(fld-lst)**

Namen aller Felder, die keine Schlüsselfunktion wahrnehmen.

Sind weder Schlüsselfelder noch Namensfelder definiert, können ausschließlich SAVE-Variable in die Tabelle eingetragen werden, wenn ein Tabellensatz geschrieben werden soll.

**WRITE**

Die Tabelle soll permanent gehalten werden. Bei einem TBSAVE- oder TBCLOSE-Befehl wird der aktuelle Zustand der Tabelle aus dem virtuellen Speicher auf Platte zurück geschrieben. WRITE ist Default.

**NOWRITE**

bewirkt, dass die Tabelle nur temporär aufgebaut wird. Bei Verarbeitungsende sollte die Tabelle durch TBEND oder TBCLOSE geschlossen werden.

**REPLACE**

eine gleichnamige vorhandene Tabelle darf überschrieben werden. Ist eine gleichnamige Tabelle im virtuellen Speicher bereits geöffnet, wird sie gelöscht, bevor die neue Tabelle erzeugt wird (RC=4). Das Duplikat der Tabelle wird in der Eingabebibliothek nicht gelöscht.

Ist WRITE angegeben wird ein eventuell vorhandenes gleichnamiges Member in der Ausgabebibliothek überschrieben. Dies ist unter Umständen problematisch, wenn als Eingabe- und Ausgabebibliothek die gleiche PO-Datei zugewiesen ist, was in der Praxis häufig vorkommt.

**LIBRARY(ddname)**

benennt den DD-Namen der Eingabebibliothek (Standard ist ISPTLIB, wenn die Angabe fehlt).

**SHARE**

erlaubt, dass die Tabelle aus allen logischen Bildschirmen (SPLIT-SCREEN-Modus) genutzt werden kann, wobei nur ein Schirm den TBCREATE-SHARE-Befehl absetzen kann. Wird von einem anderen Schirm aus ein TBCREATE oder TBOPEN auf die gleiche Tabelle abgesetzt, wird entsprechend der Nutzungszähler hoch gezählt. Der Zähler wird bei den jeweiligen TBEND- oder TBCLOSE-Befehlen wieder vermindert.

**Return Codes**

- |    |                                                                                                                                                                                                    |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Normales Ende                                                                                                                                                                                      |
| 4  | Normales Ende. Eine gleichnamige Tabelle existiert bereits aber REPLACE war angegeben                                                                                                              |
| 8  | Die Tabelle existiert bereits, aber REPLACE war nicht angegeben oder REPLACE war angegeben aber die Tabelle ist im SHARED Modus geöffnet                                                           |
| 12 | Table in USE                                                                                                                                                                                       |
| 16 | WRITE wurde angegeben und eine alternative Eingabebibliothek war nicht angegeben. TBCREATE kontrolliert die Eingabebibliotheken, um zu prüfen, ob eine gleichnamige Tabelle existiert (siehe RC 8) |
| 20 | Schwerer Fehler                                                                                                                                                                                    |

**TBDELETE – ISPF DM Service**

```
►►"TBDELETE tabname"◄◄
```

Löscht einen Satz aus der Tabelle.

Bei Tabellen mit Schlüssel wird der zu löschende Satz gemäß den Inhalten der gleichnamigen Variablen in der Tabelle gesucht.

Ohne Schlüssel wird der Satz gelöscht, auf den der CRP zeigt.

CRP wird nach erfolgreichem Löschen auf den vorangehenden Satz gestellt.

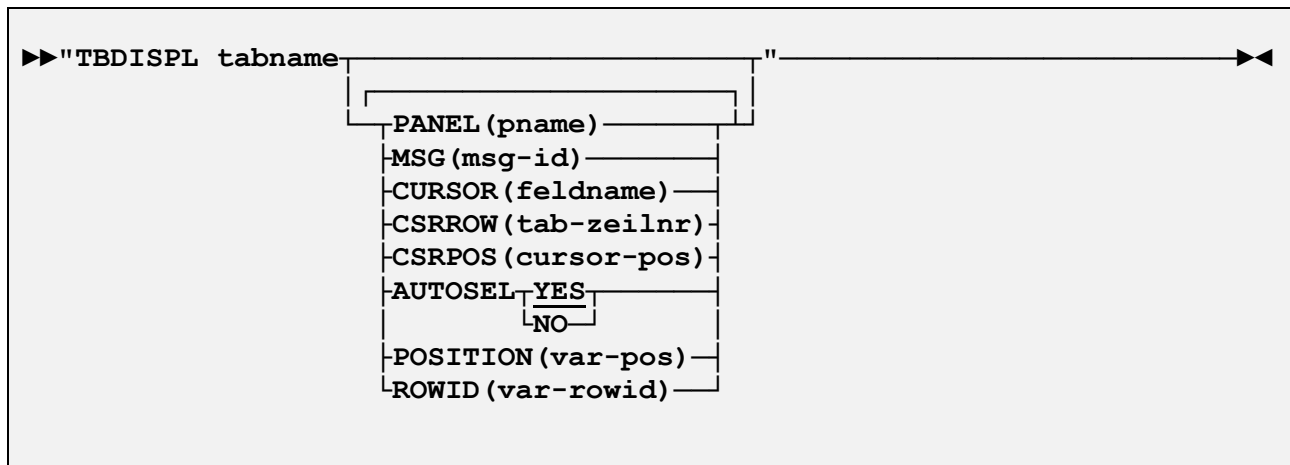
**Tabname**

Name der Tabelle, aus der ein Satz gelöscht werden soll.

**Return Codes**

- |    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 0  | Normales Ende                                                                                                |
| 8  | Mit Schlüssel: Keinen passenden Satz gefunden. CRP wird auf TOP gesetzt<br>Ohne Schlüssel: CRP steht auf TOP |
| 12 | Tabelle nicht geöffnet                                                                                       |
| 20 | Schwerer Fehler                                                                                              |

## TBDISPL – ISPF DM Service



TBDISPL bringt Informationen aus einem Panel mit denen einer Tabelle zusammen. Es zeigt alle, oder bestimmte Zeilen aus einer Tabelle am Bildschirm und erlaubt blättern (SCROLLING).

Sollen nur bestimmte Tabellenzeilen gezeigt werden, muss vor der Anzeige ein TBSARG-Befehl (SearchARGument) die Auswahlkriterien festlegen. In der )MODEL-Zeile muss die Anweisung ROWS(SCAN) kodiert sein.

Das Format der Satzanzeige ist im angegebenen Panel hinterlegt. Dieses enthält einen konstanten (Überschriften, etc.) und einen variablen Teil ( zur Tabellenanzeige). Der variable Teil des Panels wird im )MODEL des Panels beschrieben, innerhalb dessen in bis zu acht Zeilen festgelegt werden kann, wie die Sicht einer Tabellenzeile auszusehen hat.

Es können prinzipiell beliebig viele Tabellenzeilen modifiziert werden. Die Änderungen in der Tabelle werden nicht vom TBDISPL-Service selbst durchgeführt. In der aufrufenden Funktion können aber alle modifizierten Tabellensätze in einer Schleife abgearbeitet werden.

### Tabname

Name der Tabelle, deren Sätze angezeigt werden sollen.

### PANEL(pname)

Bildschirmmaske, die zur Anzeige genutzt werden soll.

### MSG(msg-id)

Zeigt auf eine Nachricht, die im Panel angezeigt werden soll.

### CURSOR(feldname)

benennt das Feld, auf das der Cursor bei der Anzeige gesetzt werden soll. Ist im Panel die Steuervariable :CURSOR gesetzt, hat diese Definition den Vorrang.

### CSRROW(tab-zeilnr)

definiert die Tabellenzeilennummer (CRP-Nummer), auf die positioniert wird.

Ist dieser Parameter angegeben, ohne dass AUTOSEL(NO) gesetzt ist, werden die Inhalte dieser Zeile gezeigt, auch wenn der Benutzer diese Zeile nicht explizit ausgewählt hat. Dieses Vorgehen nennt man automatische Auswahl.

Ist die Steuervariable .CSRROW im Anzeigepanel gesetzt, hat sie Vorrang vor der Angabe im TBDISPL-Service.

### **CSRPOS(csr-pos)**

Position innerhalb eines Feldes, auf die der Cursor positioniert werden soll. Ist dieser Wert nicht angegeben, wird die Stelle 1 des entsprechenden Feldes angenommen.

### **AUTOSEL(YES | NO)**

YES regelt, dass, sofern der CSRROW()-Parameter angegeben oder .CSRROW im )INIT- oder )REINIT-Teil des Panels kodiert ist, die betreffende Zeile gezeigt wird, auch wenn sie vom Benutzer nicht explizit ausgewählt wurde. Dieses Verfahren nennt man automatische Auswahl. Die Angabe YES wird angenommen, wenn AUTOSEL() fehlt.

Bei NO wird, auch wenn CSRROW() oder .CSRROW gesetzt ist, eine Tabellenzeile nur gezeigt, wenn sie vom Benutzer durch Eingabe in mindestens ein Feld der Zeile ausgewählt wurde.

Sind weder CSRROW() noch .CSRROW angegeben, wird die AUTOSEL()-Angabe ignoriert.

Ist im Panel die .AUTOSEL-Steuervariable angegeben, hat sie Vorrang vor dem AUTOSEL()-Parameter im TBDISPL.

### **POSITION(var-crp)**

die Variable wird mit der Identifikationsnummer des aktuellen Satzes in der Tabelle gefüllt (nach Befehlsausführung).

### **ROWID(var-rowid)**

legt in der angegebenen Variablen eine Identifikationsnummer des Datensatzes ab, auf den der CRP zeigt (vor Befehlsausführung). Dieser Wert kann im Operanden ROW() im TBSKIP-Befehl angegeben werden, um den Datensatz zu einem späteren Zeitpunkt wieder aufsuchen zu können.

### **Umsetzung der Parameter**

- Die Angaben PANEL(pname) und MSG(msg-id) sind optional. Werden sie angegeben, zieht dies folgende Verarbeitung nach sich:
- Ist der Panelname angegeben aber keine Meldungsnummer definiert, wird das Panel geladen, Sätze (beginnend mit dem Satz, auf den der CRP zeigt) werden aus der Tabelle gelesen und am Bildschirm angezeigt. Eine Meldungsausgabe von eventuell vorausgegangenen TBDISPL-Befehlen erfolgt nicht.
- Sind Panelname und Meldungsnummer angegeben, wird zusätzlich die Meldung ausgegeben.
- Ist kein Panelname, aber eine Meldung definiert, wird die aktuelle Tabellenanzeige mit der Nachricht überlagert, ohne neuen Bildaufbau oder Nachlesen der Tabelle.
- Sind weder Panelname noch Meldung angegeben, ist die weitere Verarbeitung davon abhängig, ob Tabellensätze zur Verarbeitung ausgewählt wurden.
- Sind keine Sätze ausgewählt passiert folgendes: War die letzte Bedieneraktion:
  - Drücken der ENTER-Taste, werden die entsprechenden Zeilen aus der Tabelle neu gelesen und der Bildschirm erfährt ein Refreshing.



- Die Eingabe eines SCROLL-Befehles, wird die Funktion ausgeführt, die neuen Sätze werden aus der Tabelle gelesen und am Terminal angezeigt.
- Die Eingabe von END oder RETURN, wird der CRP auf Tabellenanfang gesetzt und die Steuerung an die aufrufende Funktion zurückgegeben. Von TBDISPL wird ein RC 8 gemeldet.
- Wurden Sätze zur Verarbeitung ausgewählt, wird der CRP auf die jeweiligen Sätze positioniert. Die Sätze werden in der Tabelle aufgesucht und die Eingabefelder aus dem Panel werden in den entsprechenden Sätzen eingearbeitet.

Über den CONTROL-Service kann das Umfeld eines TBDISPL ohne Panelname in Schleife gesichert und rückgespeichert werden, falls er durch einen anderen TBDISPL, DISPLAY, BROWSE oder EDIT unterbrochen werden sollte.

- - Die Angaben CURSOR und CSRROW sind optional. Werden sie kodiert, sieht die Verarbeitung folgendermaßen aus:
    - Wird auf CURSOR verzichtet aber CSRROW kodiert, wird der Cursor auf das erste Feld der entsprechenden Tabellenzeile gesetzt.
    - Wird CURSOR angegeben, aber CSRROW fehlt, entscheidet der aktuelle Standort des CRP, welche Tabellenzeile ausgewählt wird. Der Cursor wird auf das angegebene Feld positioniert. Steht der CRP auf TOP, wird der Cursor in die Kommandozeile gestellt.
    - Sind weder CURSOR noch CSRROW angegeben, wird der Cursor in die Kommandozeile gesetzt.
    - Sind CURSOR und CSRROW angegeben, wird die in CSRROW angegebene Tabellenzeile ausgewählt und der Cursor auf das angegebene Feld positioniert.

Wenn der CSRROW-Parameter kodiert ist, ohne dass AUTOSEL(NO) angegeben wurde, wird die entsprechende Zeile ausgewählt, auch wenn sie vom Benutzer nicht modifiziert wurde.

Jede Angabe der Steuervariablen .CURSOR und .CSRROW im Panel hat Vorrang vor den gleichbedeutenden Angaben im TBDISPL-Befehl.

### **Erläuterung der TBDISPL-Operationen**

TBDISPL erlaubt dem Benutzer das Vor-/Rückwärtsblättern in der Tabelle und die Eingabe von Daten sowohl in Feldern im )BODY des Panels als auch in den Feldern des Panelteiles, der dem Blättern unterliegt.

Die Verarbeitung des TBDISPL ist abhängig davon, ob im entsprechenden Panel eine )REINIT- oder )PROC-Sektion kodiert ist.

Wenn eine )REINIT- oder )PROC-Sektion kodiert ist, und der Benutzer die ENTER-Taste drückt, ohne Tabellensätze zu modifizieren, wird vom TBDISPL die Steuerung an die aufrufende Funktion zurückgegeben.

Ist weder ein )REINIT- noch ein )PROC-Teil im Panel kodiert, und der Benutzer hat keine Sätze modifiziert, bevor er die ENTER-Taste drückte, wird dies vom TBDISPL als "NO Operation" interpretiert und die Steuerung bleibt beim TBDISPL. Dieses Vorgehen wurde aus Kompatibilitätsgründen zu vorangegangenen Versionen des Dialog Managers so belassen.

Während der Panelanzeige des TBDISPL können die nachfolgenden Aktionen eines Dialog-Anwenders die Rückgabe der Steuerung an die aufrufende Funktion veranlassen:

DAT-FREIG ohne Eingabe und )PROC oder )REINIT ist im Panel kodiert.

Dateneingabe im fixen oder variablen Teil des Panels und drücken der ENTER-Taste.

Dateneingabe im fixen oder variablen Teil des Panels und drücken der UP- oder DOWN-Funktion.

Eingabe des END- oder RETURN-Befehles

Vor- oder rückwärts blättern, wenn nicht genügend Tabellenzeilen vorhanden sind, um die SCROLL-Anforderung zu befriedigen.

**Folgende Aktionen belassen die Steuerung bei der Dialogfunktion:**

Keine Dateneingabe vor ENTER und weder )PROC noch )REINIT im Panel kodiert.

Blättern ohne Dateneingabe, sofern genügend Tabellensätze vorhanden sind, um das Blättern vollständig durchführen zu können.

Eingabe eines Kommandos außer UP, DOWN, END oder RETURN

Eingabe eines Kommandos, das einen neuen Dialog startet.

Nach der Anzeige einer Bildschirmmaske durch TBDISPL, aber vor Rückgabe der Steuerung an die aufrufende Funktion passiert folgendes:

- Die Inhalte aller )BODY-Felder werden in Variablen hinterlegt.
- Ohne Satzselektion wird CRP auf TOP gesetzt. Wenn SCROLL-RETURN-TO-FUNCTION definiert ist, und Rows zur Befriedigung der SCROLL-Anforderung benötigt werden, werden die SCROLL-RETURN-Systemvariablen im Functionpool hinterlegt.
- Wurden Zeilen ausgewählt, steht der CRP auf der ersten dieser Zeilen. Die Feldinhalte stehen in gleichnamigen Variablen. Danach werden alle Felder des )MODEL-Sets in Variable im Function-Pool eingetragen.
- ZTDSELS enthält die Anzahl selektierter Rows.
- Die Nummer der ersten ausgewählten Zeile steht in ZTDTOP. Soll die Tabelle so wieder angezeigt werden, wie der Benutzer sie gesehen hat, muss der CRP auf diese Zeile rückpositioniert werden, bevor TBDISPL wieder mit PANEL() aufgerufen wird. (siehe auch "Dialog Management Guide - Techniken zur Nutzung des TBDISPL-Service").
- Die Variablen ZTDSELS und ZTDTOP stehen im Function Pool. REXX-Programme können unmittelbar darauf zugreifen. Ein Lademodul kann über VDEFINE oder VCOPY an die Informationen gelangen.
- Wird mehr als eine Zeile der Tabelle ausgewählt, enthält ZTDSELS einen Wert von 2 oder mehr. Dies bedeutet, dass Tabellenzeilen zur Verarbeitung anstehen. Diese Zeilen nennt man "offene ausgewählte Zeilen (pending selected rows)". Ein Aufruf an TBDISPL ist nötig, um an die nächste der offenen Tabellenzeilen zu gelangen. Dabei werden die entsprechenden Feldinhalte in gleichnamige Variable im Function Pool übertragen. Wenn CRP positioniert ist, kann die Funktion den jeweiligen Satz verarbeiten. Für diese Abarbeitung der ausgewählten Tabellensätze dürfen im TBDISPL Service weder PANEL() noch MSG() kodiert sein. Die Verarbeitung innerhalb einer Schleife verläuft immer nach dem oben beschriebenen Schema, wobei bei jedem Schleifendurchlauf der nächste ausgewählte Satz bearbeitet werden kann.
- Grundsätzlich kann im Dialog ein TBDISPL mit Angabe eines Panelnamens erforderlich sein, noch bevor alle ausgewählten Tabellensätze abgearbeitet sind. Dabei gehen aber die Informationen zur weiteren Satzverarbeitung verloren. Um dieses Problem zu lösen kann durch den CONTROL-Service das Speichern und Rückspeichern der entsprechenden Umgebung des TBDISPL veranlasst werden.

**Return Codes**

- 0 Wenn im Panel weder eine )PROC- noch eine )REINIT-Section kodiert ist, wurde <ENTER> gedrückt, oder ein Scroll-Befehl abgesetzt. Dabei trat folgende Situation auf:
- Eine Zeile aus der Tabelle wurde ausgewählt. Der CRP wurde auf diese Tabellenzeile gesetzt und die Feldinhalte wurden in gleichnamige Variable im Function Pool überstellt. Anschließend wurden die Eingabefelder des zugehörigen )MODEL-Sets in die entsprechenden Variablen im Function Pool übertragen.
  - Vom Benutzer erfolgte eine Eingabe im konstanten Teil des Panels
- Anderenfalls war der Grund:
- Es wurde SCROLL-RETURN-TO-FUNCTION festgelegt (ZTDRET wurde für UP, DOWN und/oder VERTICAL gesetzt. Und dabei nicht genügend Tabellenzeilen vorhanden waren um die Scroll-Anforderung zu befriedigen oder keine Ausgewählten Zeilen mehr zu verarbeiten waren.
  - Gibt es im Panel eine )PROC- oder )REINIT-Section, wäre auch möglich, dass der Bediener nur auf <ENTER> gedrückt hat, ohne etwas einzugeben
- 4 <ENTER> oder ein Scrollbefehl wurden ausgeführt. Passiert ist eins der folgenden:
- Zwei oder mehr Tabellenzeilen wurden im variablen Teil des Panels ausgewählt. Der CRP wurde auf die erste ausgewählte Zeile gestellt und die Feldinhalte stehen im Function Pool zur Verfügung.
  - Der Benutzer hat Eingaben im festen Teil des Panels durchgeführt. Wenn SCROLL-RETURN-TO-FUNCTION definiert wurde und zwei oder mehr Zeilen zur Verarbeitung ausgewählt wurden, wird vom TBDISPL ein RC 4 ausgegeben, bis alle ausgewählten Zeilen abgearbeitet sind. Es können erst dann neue Tabellenzeilen zur Verarbeitung ausgewählt werden, wenn alle vorher ausgewählten Sätze abgearbeitet wurden (wenn der Wert von ZTDSELS auf 0 gesetzt ist).
  - Für einen TBDISPL in Schleife, bei dem kein Panel und keine Nachricht angegeben sind, wird RC 4 ausgegeben, solange noch zu verarbeitende Tabellensätze verbleiben. Für die letzte zu verarbeitende Zeile wird RC 0 ausgegeben, wobei der Wert von ZTDSELS dabei noch auf 1 zeigt.
- 8 <END> oder <RETURN> wurde gedrückt. Der CRP ist auf die Erste der ausgewählten Tabellenzeilen gesetzt. Die Felder des Tabellensatzes und dem entsprechenden )MODEL-Set des Panels stehen im Function Pool zur Verfügung.  
 Wurden keine Tabellensätze ausgewählt, steht der CRP auf TOP (0)  
 Um alle ausgewählten Tabellensätze verarbeiten zu können, nachdem END oder RETURN vom Bediener abgesetzt wurde, muss der TBDISPL Service ohne Angabe eines Panels oder einer Nachricht in einer Schleife aufgerufen werden, bis der Inhalt aus ZTD-SELS bei 1 angekommen ist.  
 Wenn der END-Befehl bei einem TBDISPL mit Panelname eingegeben wird und eine untergeordnete (nachfolgende) Anzeige (Redisplay) erfolgen soll, endet dies ebenfalls mit einem RC 8.  
 Der Benutzer könnte Eingaben im fixen Teil des Panels gemacht haben.
- 12 Panel oder Nachricht nicht gefunden oder  
 Syntaxfehler oder  
 Die Tabelle ist nicht geöffnet
- 20 Schwerer Fehler

## TBEND – ISPF DM Service

```
▶▶"TBEND tabname"◀◀
```

löscht die Tabelle aus dem virtuellen Speicher. Der aktuelle Zustand der Tabelle wird nicht auf Platte zurückgeschrieben.

Wird der TBEND Service für eine SHARED-Tabelle aufgerufen, wird der Nutzungszähler um 1 vermindert. Ist der Zähler auf Null, wird TBEND ausgeführt, andernfalls erfolgt keine weitere Aktion und die Tabelle bleibt im virtuellen Speicher verfügbar.

### Tabname

liefert den Namen der Tabelle, die betroffen ist.

### Return Codes

- |    |                        |
|----|------------------------|
| 0  | Normales Ende          |
| 12 | Tabelle nicht geöffnet |
| 20 | Schwerer Fehler        |

## TBERASE – ISPF DM Service

```
▶▶"TBERASE tabname"◀◀
 LIBRARY(ddname)
```

Löscht eine Tabelle aus der Ausgabebibliothek. Die Ausgabebibliothek muss zugewiesen sein.

### Tabname

benennt die zu löschende Tabelle

### LIBRARY(ddname)

liefert den DD-Namen, unter dem die Ausgabebibliothek zugewiesen ist, sofern dieser nicht ISPTABL sein soll.

### Return Codes

- |    |                                              |
|----|----------------------------------------------|
| 0  | Normales Ende                                |
| 8  | Tabelle in Ausgabebibliothek nicht vorhanden |
| 12 | Table in USE                                 |
| 16 | Ausgabebibliothek nicht zugewiesen           |
| 20 | Schwerer Fehler                              |

## TBEXIST – ISPF DM Service

```
▶▶"TBEXIST tabname"◀◀
```

Prüft das Vorhandensein eines bestimmten Satzes in einer Tabelle. Dabei werden die aktuellen Inhalte der den Schlüsselfeldern entsprechenden Variablen als Vergleichskriterium bei der Suche innerhalb der Tabelle genutzt.

Dieser Service ist für Tabellen ohne Schlüssel nicht verfügbar.

Der CRP wird auf TOP gesetzt.

### Tabname

Name der zu durchsuchenden Tabelle

### Return Codes

- 0    Normales Ende
- 8    Mit Schlüssel: Keinen passenden Satz gefunden. CRP wird auf TOP gesetzt  
     Ohne Schlüssel: Service nicht verfügbar  
     In beiden Fällen wird der CRP auf TOP gestellt
- 12   Tabelle nicht geöffnet
- 20   Schwerer Fehler

## TBGET – ISPF DM Service

```
▶▶"TBGET tabname"◀◀
 ┌── SAVENAME (var-name) ──┐
 │ ROWID (var-rowid) ───┐
 │ NOREAD ───────────┐
 └── POSITION (var-pos) ─┘
```

Liest eine ROW. Ohne KEYS, ist es der Satz des CRP. Mit Keys erfolgt ein Scan. Suchkriterium sind die Inhalte der auf die KEYS zeigenden Variablen.

### Tabname

benennt die Tabelle, die von TBGET betroffen wird.

### SAVENAME(var-name)

In der Variablen werden als Wortkette die Namen der Extension-Variablen geliefert, die für den aktuellen Satz gespeichert sind.

**ROWID(var-rowid)**

Satznummer des CRP (vor Befehlsausführung). Der Wert kann bei ROW() im TBSKIP angegeben werden, um den Datensatz später wieder aufzusuchen.

**NOREAD**

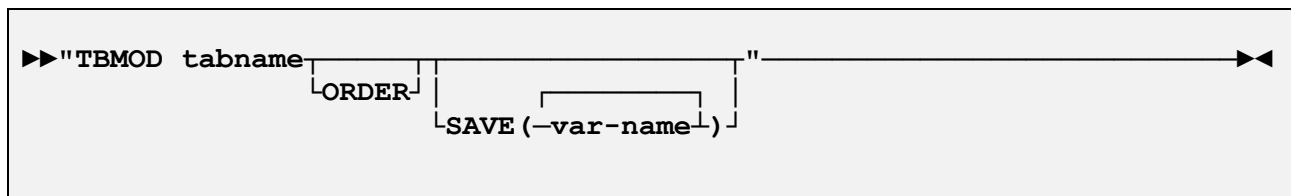
abweichend vom Standard werden die Feldinhalte des aktuellen Satzes nicht in gleichnamige Variable übertragen.

**POSITION(var-crp)**

CRP nach Befehlsausführung.

**Return Codes**

- 0     Normales Ende
- 8     Mit Schlüssel: Keinen passenden Satz gefunden. CRP wird auf TOP gesetzt  
      Ohne Schlüssel: CRP steht auf TOP
- 12    Tabelle nicht geöffnet
- 16    Der Wert einer Variablen wurde abgeschnitten, oder nicht genügend Platz um die Namen aller SAVE-Variablen auszugeben.
- 20    Schwerer Fehler

**TBMOD – ISPF DM Service**

Modifiziert einen Satz in der Tabelle.

Mit Keys wird der zu ändernde Satz über die den Keys entsprechenden Variableninhalte gesucht. Ist das Suchergebnis negativ, erfolgt ein TBADD.

Ohne Keys ist TBMOD = TBADD. Änderungen sind mit TBPOT durchzuführen.

Die Inhalte aller zu den Tabellenfeldern namensgleichen Variablen werden in die Tabelle eingetragen.

**Tabname**

Name der Tabelle, in der eine Modifikation durchgeführt werden soll.

**ORDER**

Vorausgesetzt die Tabelle wurde sortiert, wird der Tabellensatz gemäß der Sortierkriterien in die Tabelle eingearbeitet, nachdem ggfls. die Schlüsselfelder aktualisiert wurden.

Fehlt die Angabe bei einer sortierten Tabelle, wird die Sortierung zerstört und sie muss erneut sortiert werden. Bei unsortierten Tabellen wird ORDER ignoriert.

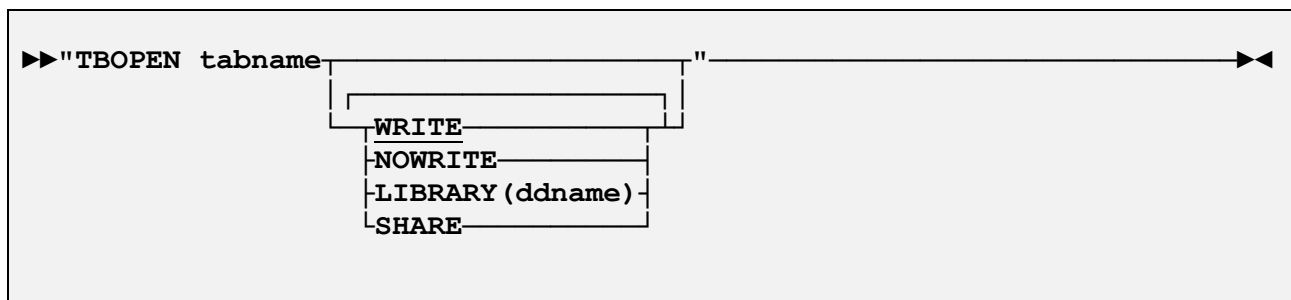
### SAVENAME(var-name)

Enthält als Wortkette die Namen aller SAVE-Variablen des Tabellensatzes.

### Return Codes

- 0 Normales Ende
- 8 Mit Schlüssel: Keinen passenden Satz gefunden. Satz hinzugefügt (wie TBADD).
- 12 Tabelle nicht geöffnet
- 16 Fehler bei numerischer Umwandlung (siehe numerische Restriktionen bei TBSORT)  
Die Meldung ist nur bei sortierten Tabellen möglich.
- 20 Schwere Fehler

## TBOPEN – ISPF DM Service



Öffnet eine Tabelle und überträgt sie von ISPTLIB in den virtuellen Speicher. Ein ENQ gewährleistet, dass zum jeweiligen Zeitpunkt kein anderer Benutzer auf die Tabelle zugreift (betrifft nur das Member, nicht die Datei).

### tabname

Name der Tabelle, die geöffnet werden soll.

### WRITE

Die Tabelle wird für Updates geöffnet. Änderungen können während Verarbeitung durch TBSAVE, oder bei Ende durch TBCLOSE zurück geschrieben werden. Die Tabelle ist für andere Benutzer gesperrt.

### NOWRITE

Die Tabelle kann nach Veränderungen nicht zurück geschrieben werden. Die Tabelle ist für andere Benutzer nur während des Einlesens gesperrt.

**LIBRARY(ddname)**

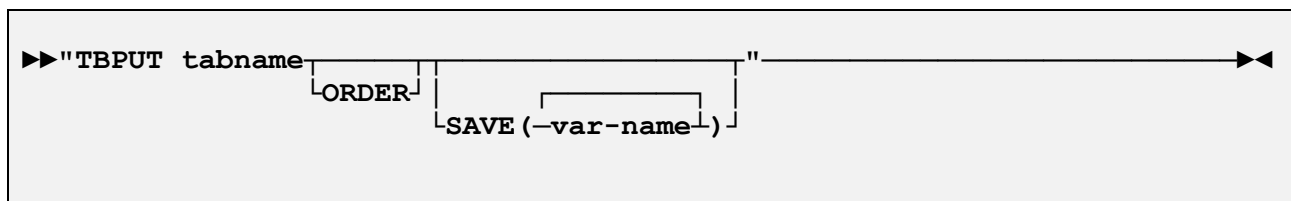
DD-Name der Eingabebibliothek, sofern nicht von ISPTLIB gelesen wird.

**SHARE**

erlaubt den Zugriff von mehreren Screens. Der erste TBOPEN eröffnet die Tabelle. Der TBOPEN des zweiten Schirms benutzt die gleiche virtuelle Tabelle und den gleichen CRP. Dabei wird der Nutzungszähler um eins erhöht.

**Return Codes**

- 0     Normales Ende
- 8     Tabelle existiert nicht
- 12    Tabelle exklusiv belegt (kann der gleiche User im Splitscreen sein).
- 16    Tabelleneingabebibliothek nicht zugewiesen
- 20    Schwerer Fehler

**TBPUT – ISPF DM Service**

Ändert den Inhalt eines Tabellensatzes.

Bei Tabellen mit Schlüssel müssen die Inhalte der mit den Schlüsselfeldern korrespondierenden Variablen mit den Inhalten der Schlüsselfelder übereinstimmen. Andernfalls erfolgt kein Update auf den Datensatz (Schlüsselfelder können nicht durch TBPUT verändert werden, siehe TBMOD).

Bei Tabellen ohne Keys wird immer der Satz geändert, auf den der CRP zeigt.

War der Update erfolgreich, wird die Position des CRP nicht verändert.

**Tabname**

Name der Tabelle, in der eine Modifikation durchgeführt werden soll.

**ORDER**

ist die Tabelle sortiert, wird der Tabellensatz gemäß der Sortierkriterien eingearbeitet. Ohne diese Angabe geht die Sortierung verloren. TBSORT ist erforderlich, um die Tabelle wieder in die gewünschte Reihenfolge zu bringen.

Wird ORDER bei unsortierten Tabellen angegeben, wird die Angabe ignoriert.

**SAVENAME(var-name)**

Liefert als Wortkette die Namen aller SAVE-Variablen des Tabellensatzes.





**NAMENUM(var-namnum)**

enthält die Anzahl der Schlüssel-, Namensfelder und Save-Variablen.

**SORTFLDS(var-fields)**

Liefert die Parameter des letzten TBSORT

**SARGLIST(var-arglist)**

Liefert die Angaben von ARGLIST() beim letzten TBSARG

**SARGCOND(var-condlist)**

Liefert die Angaben von CONDLIST() beim letzten TBSARG

**SARGDIR(var-dir)**

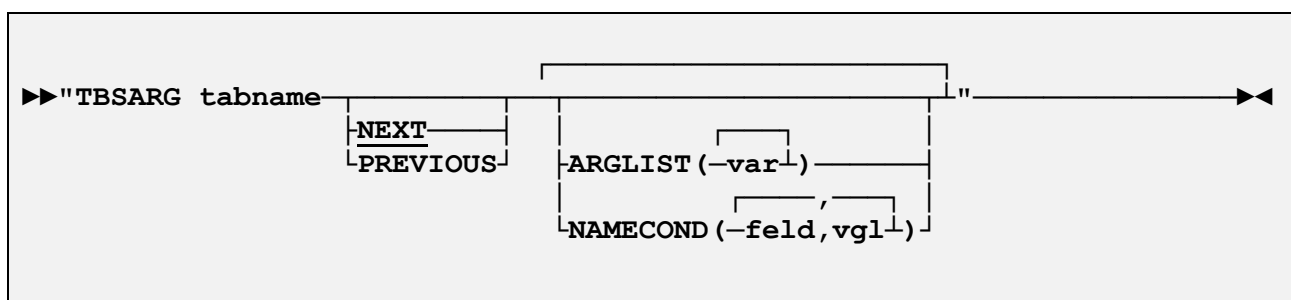
Zeigt die Richtungsangabe vom letzten TBSCAN (NEXT oder PREVIOUS)

**POSITION(var-crp)**

CRP des Tabellensatzes. Steht der CRP auf TOP, enthält die Variable eine 0.

**Return Codes**

- 0      Normales Ende
- 12     Tabelle nicht geöffnet
- 16     Wegen Platzmangels konnten nicht alle KEY- oder NAMES-Namen in die entsprechende Variable retourniert werden.
- 20     Schwerer Fehler

**TBSARG – ISPF DM Service**

Definiert ein Suchargument (TaBle Search ARGument), über das die Tabelle gefiltert werden kann. Befehle, die sich auf dieses Suchargument beziehen sind TBSCAN und TBDISPL.

Wird das Suchargument in Verbindung mit TBDISPL genutzt, muss im )MODEL-Teil des Anzeigepanels ROWS(SCAN) gesetzt sein.

Das Suchargument wird in Form namensgleicher Variabler zu den Tabellenfeldern hin festgelegt. Enthält eine Variable keinen Wert (NULLSTRING), wird sie beim Scannen nicht berücksichtigt.

Bevor die Variablen gefüllt werden, die das Suchargument bilden, ist es sinnvoll, durch TBVCLEAR alle Variablen inhaltlich zu löschen, für die es gleichnamige Tabellenfelder gibt. So kann ausgeschlossen werden, dass eine noch gefüllte Variable ungewollt zur Bildung des Suchargumentes herangezogen wird.

SAVE-Variable können ebenfalls in das Suchargument aufgenommen werden, sofern ihre Namen in die Namensliste im Operanden ARGLIST() eingetragen werden. NULLSTRING in einer SAVE-Variablen ist dabei ein gültiges Suchargument.

Generische Suche ist möglich. Soll die Suche nach Postleitzahlen alle Sätze, deren PLZ-Feld mit einer 8 beginnt, finden, ist die entsprechende Variable mit "8\*" zu füllen. Der Vergleich endet positiv, wenn die erste Stelle des Feldes eine 8 enthält.

Wird ein Suchargument festgelegt, wird ein gegebenenfalls zu einem früheren Zeitpunkt definiertes überschrieben.

### **Tabname**

liefert den Namen der Tabelle, für die das Suchargument gebildet werden soll.

### **NEXT**

Der Scanvorgang beginnt hinter der Tabellenzeile, auf die der CRP zeigt. Standard, wenn keine Richtung angegeben ist.

### **PREVIOUS**

Der Scanvorgang beginnt vor der Tabellenzeile vor der aktuellen Position des CRP liegt. Um hier ab Tabellenende in Richtung Anfang vergleichen zu können, muss der CRP auf TOP stehen.

### **ARGLIST(var)**

benennt eine oder mehrere SAVE-Variable, die in den Scanvorgang einbezogen werden sollen.

### **NAMECOND(feld,vgl)**

liefert eine Liste jeweils paariger Angaben von Namensfeldern und Vergleichsoperanden, über die die Feldinhalte mit den entsprechend gleichnamigen Variablen verglichen werden sollen. Die Schreibweise sieht folgendermaßen aus:

**NAMECOND (feld1 ,vgl1 ,feld2 ,vgl2    ···    feldn ,vgln)**

#### **feld**

muss der Name eines Schlüsselfeldes, Namensfeldes oder einer SAVE-Variablen sein. Ist der Name nicht vorhanden wird RC 20 ausgegeben.

#### **vgl**

liefert den Vergleichsoperanden, über den das Feld mit der gleichnamigen Variableninhalten verglichen wird. Die möglichen Vergleichsoperanden sind:

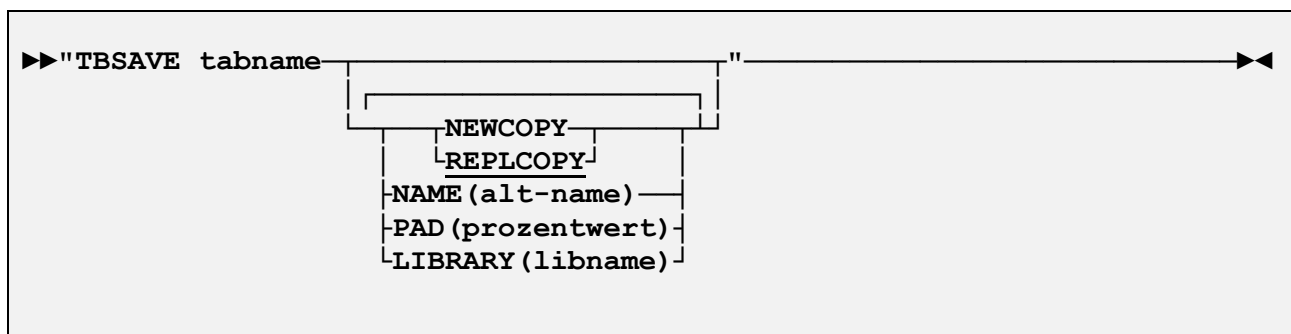
- |    |                                                                      |
|----|----------------------------------------------------------------------|
| EQ | Feldinhalt und Variableninhalt müssen gleich sein (Default).         |
| NE | Feldinhalt und Variableninhalt müssen unterschiedlich sein.          |
| LE | Feldinhalt muss kleiner oder gleich des Inhaltes der Variablen sein. |
| LT | Feldinhalt muss kleiner dem Inhalt der Variablen sein.               |

- GE Feldinhalt muss größer oder gleich des Inhaltes der Variablen sein.  
GT Feldinhalt muss größer als der Inhalt der Variablen sein.

### Return Codes

- 0 Normales Ende  
8 Alle Variablen sind NULL und ARGVLIST() wurde nicht angegeben  
12 Tabelle nicht geöffnet  
20 Schwerer Fehler

## TBSAVE – ISPF DM Service



Schreibt den aktuellen Stand einer Tabelle aus dem virtuellen Speicher unter ISPTABL auf Platte zurück. Die Tabelle muss im WRITE-Modus geöffnet worden sein. Das Abbild der Tabelle im virtuellen Speicher wird nicht gelöscht. Die Verarbeitung kann unmittelbar fortgesetzt werden.

### tabname

benennt die Tabelle, die gesichert werden soll.

### NEWCOPY

regelt, dass die Tabelle an das Ende der Tabellenbibliothek geschrieben werden soll, auch wenn ein Update am alten Platz möglich wäre. Damit wird erreicht, dass der jeweils alte Zustand vor Verarbeitung nicht überschrieben wird. Die Restauration einer alten Tabellenversion ist aber für den Anwender im Allgemeinen nicht möglich (Systemprogrammierung).

### REPLCOPY

steuert, dass die Tabelle an ihren alten Platz in der Ausgabebibliothek zurück geschrieben werden soll. Dazu wird zunächst ein Vergleich der Membergröße in der PO-Datei mit der Größe der Tabelle im virtuellen Speicher durchgeführt. Ist das Member zu klein, oder ein gleichnamiges Member nicht vorhanden, wird die Tabelle an das Ende der PO-Datei angefügt.

### NAME(alt-name)

gibt für die SAVE-Option einen alternativen Namen bekannt, unter dem die Tabelle in der Ausgabebibliothek abgestellt wird. Ist ein gleichnamiges Member bereits vorhanden, wird es überschrieben. So existiert die Tabelle nach dem TBSAVE sowohl unter ihrem alten Namen (noch unverändert) als auch (mit allen aktuellen Veränderungen) unter dem neuen Namen.

**PAD(prozentwert)**

liefert, basierend auf der aktuellen Größe der Tabelle, einen Prozentualwert, der als Größenvorhalt reserviert wird, wenn die Tabelle an das Ende der Ausgabebibliothek geschrieben wird. Kann das Rückschreiben an den alten Platz in der Ausgabebibliothek erfolgen, wird diese Angabe ignoriert.

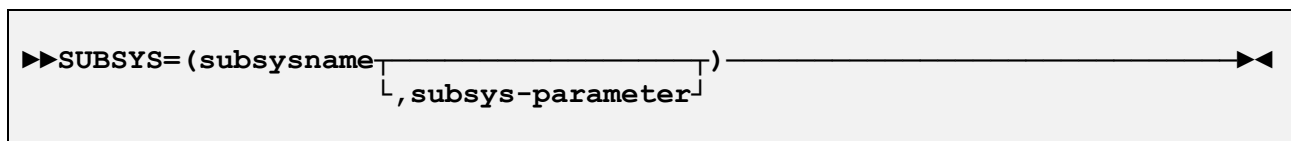
Wird beispielsweise eine Tabelle mit 100 Sätzen und der Angabe PAD(20) an das Ende der PO-Datei angehängt, wird zu den 100 Sätzen noch Platz für weitere 20 Sätze angehängt. Fehlt dieser Parameter wird PAD(0) angenommen.

**LIBRARY(ddname)**

übergibt den DD-Namen, unter dem die Tabelle abweichend zu ISPTABL gesichert werden soll.

**Return Codes**

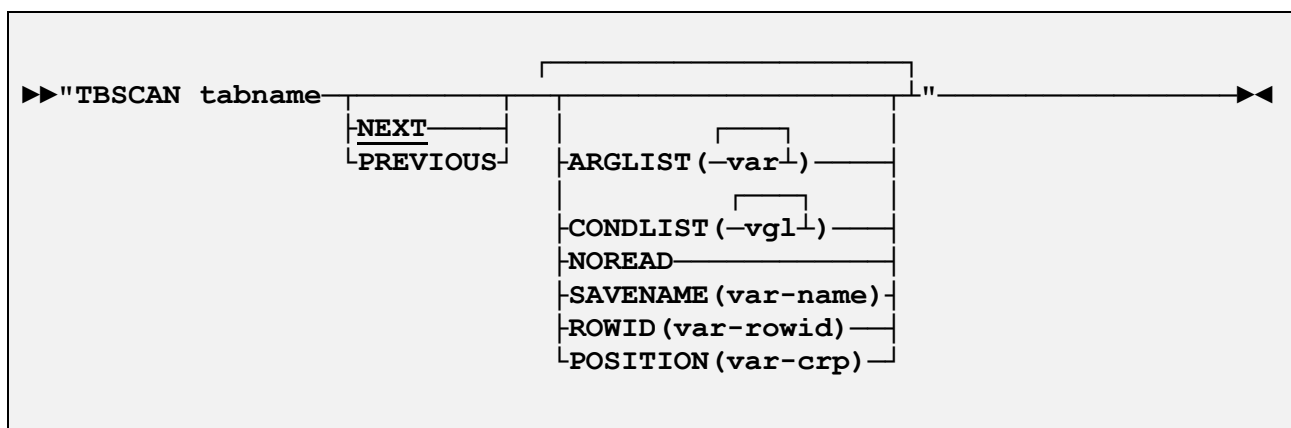
- 0      Normales Ende
- 12     Tabelle nicht geöffnet
- 16     Alternative Ausgabebibliothek nicht zugewiesen
- 20     Schwerer Fehler

**SUBSYS – JCL DD Parameter**

Legt fest, dass sich nicht z/OS, sondern ein installiertes Subsystem (beispielsweise CSM) um Dateioperationen kümmern soll. So können gegebenenfalls Dateizugriffe über die Plexgrenze erfolgen.

**Beispiel**

```
//name DD DISP=SHR,DSN=dateiname,SUBSYS=(CSM,'PLEX=APLEX')
```

**TBSCAN – ISPF DM Service**

Sucht Sätze, die mit einem Suchargument übereinstimmen. Die Suchargumente können über TBSARG oder im TBSCAN selbst definiert werden. Gesucht werden kann vor- oder rückwärts. Der erste zu vergleichende Satz ist immer der nach dem aktuellen CRP.

Wird ein Suchkriterium in TBSCAN angegeben, wird TBSARG ignoriert.

Generische Suchargumente (Me\*) sind möglich. Wird nach "DATA\*12\*" gesucht, ist der nur letzte Stern Merkmal der generischen Suche.

**tabname**

Name der Tabelle, die durchsucht werden soll.

**NEXT**

Der Scanvorgang beginnt hinter der Zeile, auf die der CRP zeigt. Dieses Vorgehen ist Standard, wenn keine Richtung angegeben ist.

**PREVIOUS**

Der Scan beginnt bei der Zeile vor dem CRP Richtung TOP. Um ab Tabellenende in Richtung Anfang suchen zu können, muss der CRP auf TOP stehen.

**ARGLIST(var)**

benennt ein oder mehrere Felder (KEYs, NAMES oder SAVE-Variable), die für den Scanvorgang herangezogen werden.

**CONDLIST(vgl)**

Liste von Vergleichsoperanden, über die die in ARGLIST() angegebenen Felder mit den entsprechenden Variablen verglichen werden sollen.

```
ARGLIST(feld1,feld2, ... feldn)
CONDLIST(vgl1,vg12, ... vgl n)
```

**feld**

muss der Name eines Schlüsselfeldes, Namensfeldes oder einer SAVE-Variablen sein.

**vgl**

Vergleichsoperand, über den das Feld mit der gleichnamigen Variableninhalt verglichen wird. Die einzelnen Angaben in ARGLIST() und CONDLIST() werden 1:1 miteinander in Verbindung gebracht. Die Vergleichsoperanden sind:

| Vergleich | Ende SCAN                                               |
|-----------|---------------------------------------------------------|
| EQ        | Wenn Inhalt aus Feld und Variable gleich sind (Default) |
| NE        | Wenn Inhalt aus Feld und Variable ungleich sind         |
| LE        | Wenn der Feldinhalt kleiner/gleich der Variablen ist    |
| LT        | Wenn der Feldinhalt kleiner der Variablen ist           |
| GE        | Wenn der Feldinhalt größer/gleich der Variablen ist     |
| GT        | Wenn der Feldinhalt größer der Variablen ist            |

**NOREAD**

Die Feldinhalte werden nicht in Variablen im Function Pool übertragen.

**SAVENAME(var-name)**

benennt die Variable, in die die Namen aller SAVE-Variablen übertragen werden. Die Namensliste wird von runden Klammern eingeschlossen. Die einzelnen Variablennamen werden durch BLANK voneinander getrennt.

**ROWID(var-rowid)**

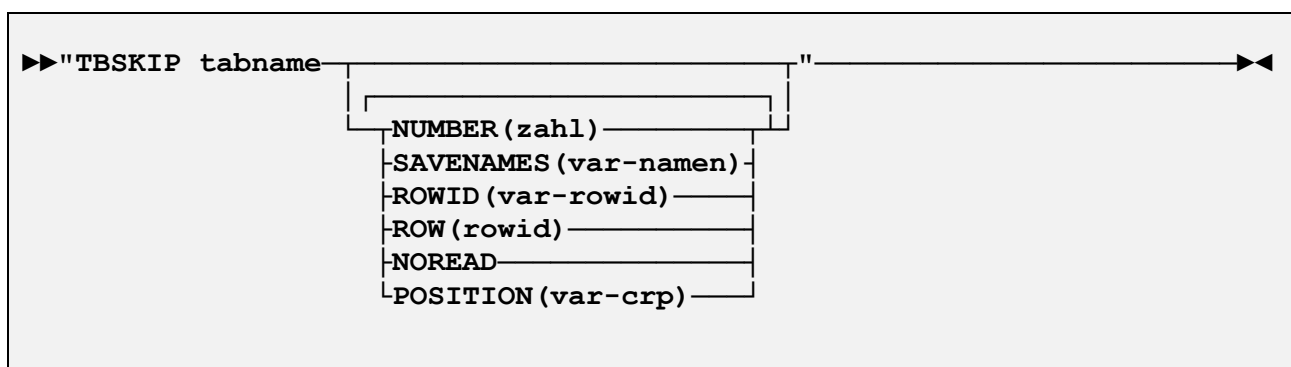
liefert den Wert des CRP vor Verarbeitung des Befehles.

**POSITION(var-crp)**

überträgt den Wert des CRP nach Verarbeitung des Befehles.

**Return Codes**

- 0     Normales Ende
- 8     Mit Schlüssel: Schlüsselfelder und Variablen stimmen inhaltlich nicht überein. CRP auf TOP gesetzt.  
      Ohne Schlüssel: CRP steht auf TOP
- 12    Tabelle existiert nicht
- 20    Schwerer Fehler

**TBSKIP – ISPF DM Service**

Bewegt den CRP um die angegebene Anzahl Zeilen.

**tabname**

Name der Tabelle, in der der CRP positioniert werden soll.

**NUMBER(zahl)**

Verschiebt den CRP nach unten oder nach oben. 0 entspricht TBGET. Ohne ROW() und NUMBER() wird CRP einen Satz nach unten bewegt.

**SAVENAMES(var-namen)**

enthält die Namen aller SAVE-Variablen. Die Namensliste wird von runden Klammern eingeschlossen.

**ROWID(var-rowid)**

liefert in der Variablen den Wert des CRP vor Verarbeitung ab.

**ROW(rowid)**

Positioniert auf einen Tabellensatz unmittelbar durch seine Satznummer. Der Wert wird über die Variable in ROWID() zur Verfügung gestellt.

**NOREAD**

die Inhalte der aktiven Zeile werden nicht im Function Pool abgebildet.

**POSITION(var-crp)**

hinterlegt in einer Variablen den Wert des CRP nach Befehlsausführung.

**Return Codes**

- 0 Normales Ende
- 8 CRP falsch positioniert. CRP auf TOP gesetzt.
- 12 Tabelle nicht geöffnet
- 16 Variablenwert abgeschnitten oder nicht genügend Platz um alle SAVE-Variablen zu melden
- 20 Schwerer Fehler

**TBSORT – ISPF DM Service**

```
►►"TBSORT tabname FIELDS (—feld,fmt,order↓) "◄◄
```

Sortiert eine geöffnete Tabelle und speichert die Informationen in einem Sort-Informationssatz. Der Sortbefehl kann auch in einer noch leeren Tabelle angewandt werden. Nachdem der Sort durchgeführt ist, steht der CRP auf TOP.

Wird die Ordnung innerhalb der Tabelle verändert (TBADD, TBMOD und TBPUT), wird der Informationssatz nur dann aktualisiert, wenn bei der jeweiligen Operation ORDER angegeben wurde. Andernfalls ist ein neuer Sortierlauf nötig.

Die Sortinformationen werden mit der Tabelle auf Platte gespeichert und sind nach TBOPEN wieder verfügbar.



**tabname**

liefert den Namen der Tabelle, die sortiert werden soll.

**feld**

ist der Name eines Schlüssel- oder Namensfeldes innerhalb der Tabelle. Das erstgenannte Feld innerhalb der Sortierfolge hat in der Sortierung die meiste Gewichtung. Die weitere Reihenfolge in der Sortierung entscheidet die Schreibweise der Sortierfelder von links nach rechts.

**fmt**

beschreibt den Sortierfeldtyp. Die möglichen Angaben sind

B      Binary  
C      Character (EBCDI-Code)  
N      Numeric

Im englischen Sprachraum (Sort nach EBCDI-Code) liegt zwischen B oder C kein Unterschied. In anderen Sprachen, wo CHARACTER-Formate abweichend zum EBCDI-Code vorliegen können, kann nur B für binäre Folge genutzt werden.

**order**

liefert die Sortierrichtung. Mögliche Angaben sind

A      Ascending (aufsteigend) oder  
D      Descending (absteigend)

Feldformat und Sortierrichtung können für das jeweils letztgenannte Feld entfallen. In diesem Fall wird für das Format "C" und für die Sortierrichtung "A" angenommen.

**Besonderheiten**

In manchen Sprachen wird der Dezimalpunkt durch Komma dargestellt. Um die nationalen Besonderheiten berücksichtigen zu können, werden drei verschiedene numerische Darstellungsformen unterstützt:

| Konvention | Beispiele |       | Wird benutzt in             |
|------------|-----------|-------|-----------------------------|
| Period     | 1,234.56  | 0.789 | Japan, Mexiko, England, USA |
| Comma      | 1.234,56  | 0,789 | Den meisten anderen Ländern |
| French     | 1234,56   | 0,789 | Frankreich, Südafrika       |

Welche benutzt wird, ist abhängig von der eingesetzten Sprache, die über ZLANG aus dem System-Profile-Pool ermittelt werden kann. Englisch erlaubt nur den Punkt. Die Sortierung ist abhängig von den Landeskonventionen.

**Regeln für das Sortieren numerischer Felder**

Ein Feld muss aus Ziffern bestehen und kann ein Vorzeichen Plus(+) oder Minus(-), den Dezimalpunkt (kann Punkt(.) oder Komma(,) je nach Länderdefinition sein) und führende Leerzeichen enthalten.

Ein numerisches Feld kann inklusive Vorzeichen und Dezimalpunkt maximal 16 Stellen lang sein.

Der größte Wert eines numerischen Feldes ist positiv oder negativ  $2^{31}-1$  (2.147.483.647).

### Hinweise zur Performance

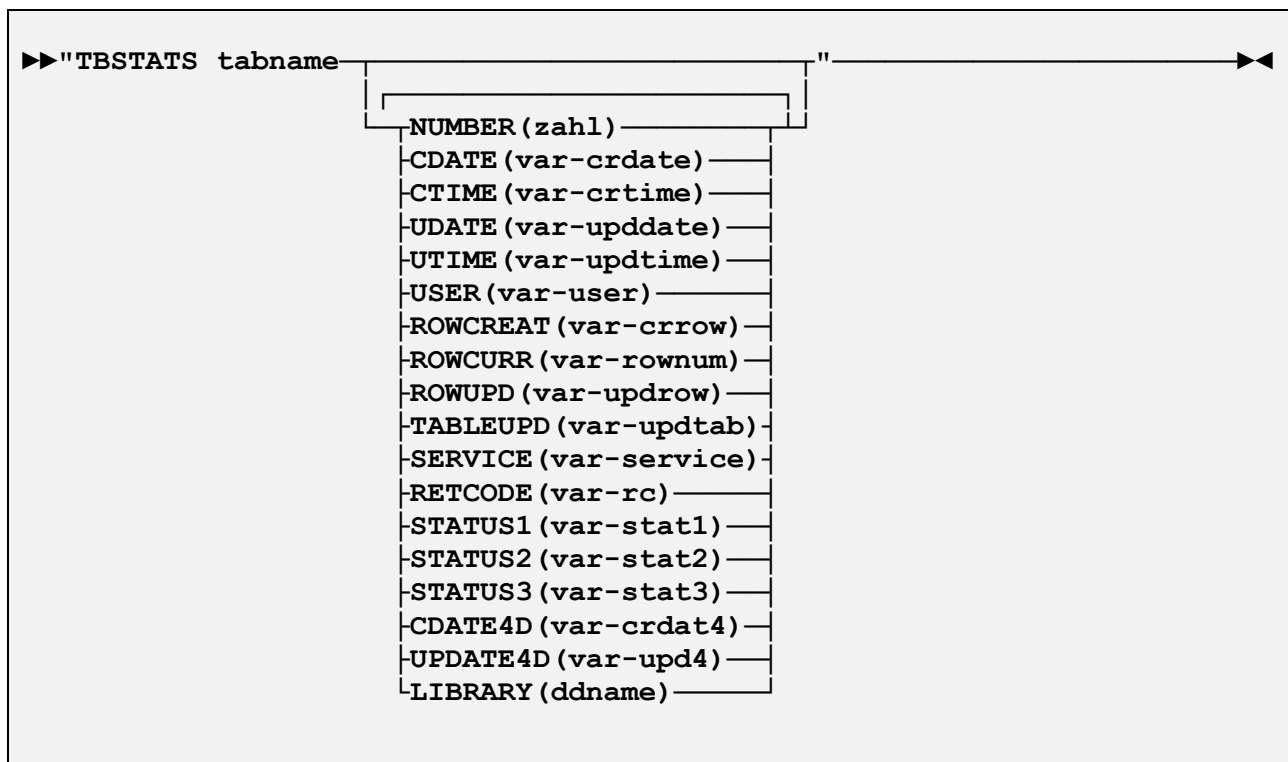
Der beste Zeitpunkt eine Tabelle zu sortieren ist, solange sie noch leer ist.

Das Sortieren numerischer Felder wirkt sich ebenfalls negativ auf die Performance aus, weil für jeden Vergleich zwei Ziffern umgewandelt werden müssen.

### Return Codes

- 0      Normales Ende
- 12     Tabelle ist nicht geöffnet
- 16     Fehler bei numerischer Umwandlung (siehe numerische Restriktionen bei TBSORT)
- 20     Schwerer Fehler

## TBSTATS – ISPF DM Service



Informationen über eine Tabelle. Diese Statistiken werden für jede permanente Tabelle geführt. Sie sind verfügbar, auch wenn die Tabelle nicht geöffnet ist, sofern sie unter ISPTLIB vorhanden ist. Ist sie im logischen Screen geöffnet, werden deren Infos ausgegeben.

### tablename

Name der Tabelle, über die Informationen eingeholt werden sollen.

### CDATE(var-crdate)

### CTIME(var-crtime)

Erstellungsdatum in der Form YY/MM/DD und Uhrzeit in der Form HH.MM.SS

**UPDATE(var-upddate)****UTIME(var-updtime)**

Datum letzter Update in der Form YY/MM/DD, Uhrzeit Updates HH.MM.SS

**CRDATE4D(var-crdat4)****UPDATE4D(var-upd4)**

Creation- und Update-Datum mit 4-stelligem Jahr

**USER(var-user)**

TSO-Benutzerkennung des letzten Änderers.

**ROWCREAT(var-crow)****ROWCURR(var-rownum)**

Liefert die Anzahl der Tabellensätze bei Erstellung der Tabelle (TBCREATE), beziehungsweise die aktuelle Anzahl der Tabellenzeilen.

**ROWUPD(var-updrow)****TABLEUPD(var-updtab)**

liefert entweder die Anzahl geänderter Tabellenzeilen, oder wie oft die Tabelle seit ihrer Erstellung insgesamt verändert wurde.

**SERVICE(var-service)**

Sofern die Tabelle im logischen Bildschirm geöffnet ist, wird der Name des Services gemeldet, der die Tabelle zuletzt bearbeitet hat.

**RETCODE(var-rc)**

gibt den RC des Services bekannt, der zuletzt die Tabelle bearbeitet hat (nur wenn die Tabelle im aktiven Bildschirm geöffnet ist).

**STATUS1(var-stat1)**

- 1 Tabelle existiert in der Input-Bibliotheksverknüpfung
- 2 Tabelle existiert nicht in der Input-Bibliotheksverknüpfung
- 3 Eingabebibliothek nicht zugewiesen

**STATUS2(var-stat2)**

- 1 Tabelle im logischen Schirm nicht geöffnet
- 2 Tabelle im logischen Schirm mit NOWRITE eröffnet
- 3 Tabelle im logischen Schirm mit WRITE eröffnet
- 4 Tabelle ist im logischen Schirm mit SHARED NOWRITE geöffnet
- 5 Tabelle ist im logischen Schirm mit SHARED WRITE geöffnet

**STATUS3(var-stat3)**

- 1 Tabelle zum Schreiben verfügbar
- 2 Tabelle nicht zum Schreiben verfügbar

**LIBRARY(ddname)**

bestimmt den DD-Namen der Eingabebibliothek, sofern von ISPTLIB abgewichen werden soll.

**Return Codes**

- |    |                                                |
|----|------------------------------------------------|
| 0  | Normales Ende                                  |
| 16 | TRUNCATION-Fehler beim Speichern von Variablen |
| 20 | Schwerer Fehler                                |

**TBTOP – ISPF DM Service**

```
▶▶"TBTOP tabname"_____◀◀
```

Setzt den CRP auf Tabellenanfang (liegt vor dem ersten Tabellensatz)

**tabname**

gibt den Namen der Tabelle bekannt, bei der der CRP auf Tabellenanfang positioniert werden soll.

**Return Codes**

- |    |                        |
|----|------------------------|
| 0  | Normales Ende          |
| 12 | Tabelle nicht geöffnet |
| 20 | Schwerer Fehler        |

**TBVCLEAR – ISPF DM Service**

```
▶▶"TBVCLEAR tabname"_____◀◀
```

Setzt alle Variablen, die namensgleich zu den Feldern der angegebenen Tabelle sind, auf NULL-STRING.

**tabname**

Name der Tabelle, deren Felder Grundlage des TBVCLEAR-Service sind.

**Return Codes**

- |    |                        |
|----|------------------------|
| 0  | Normales Ende          |
| 12 | Tabelle nicht geöffnet |
| 20 | Schwerer Fehler        |

## TENTER – ISPF Edit Macro

```
►►"TENTER zz└─┘"◄◄
 └zahl┘
```

Startet den TEXT-ENTRY-Modus. TEXT-ENTRY-Input wird innerhalb der Boundaries ausgerichtet.

### zz

Zeigt auf die Zeile, hinter der in den TEXT-ENTRY-Modus verzweigt werden soll. Zeilenzeiger kann ein Label oder eine relative Zeilennummer sein.

### zahl

Anzahl Zeilen für den TEXT-ENTRY-Modus. Default ist der restliche Bildschirm.

### Beispiel

```
"locate .zlast"
"tenter 3"
```

Sucht die letzte Zeile auf und stellt dahinter drei Eingabezeilen im TEXT-ENTRY-Modus bereit.

### Return Codes

|    |                      |
|----|----------------------|
| 0  | Normales Ende        |
| 12 | Falsche Zeilennummer |
| 20 | Schwerer Fehler      |

## TFLOW – ISPF Edit Macro

```
►►"TFLOW zz└─┘"◄◄
 └spalte┘
```

Verändert die Zeilenumbrüche bis Absatz. Absatz wird erkannt durch eine Leerzeile, oder eines der Sonderzeichen(. : &) am Zeilenanfang.

### zz

Benennt die Zeile über ein Label oder eine relative Zeilennummer, von der ab der Befehl wirken soll.

### spalte

Rechte Grenze für den Zeilenumbruch. Default ist die rechte Boundarygrenze.

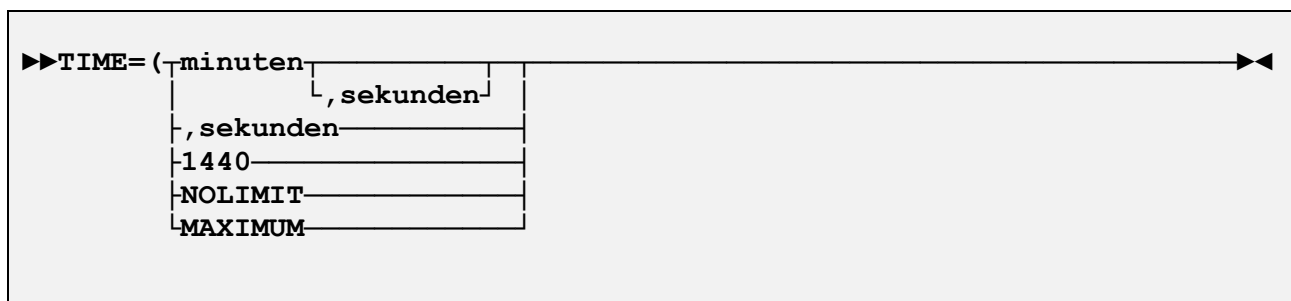
**Beispiel**

```
"(,rechts) = display_cols"
"tflow .xyz" rechts
```

Die rechte sichtbare Bildschirmspalte wird ermittelt und der Text ab Label .XYZ bis Absatzende auf diese Breite neu umgebrochen.

**Return Codes**

|    |                      |
|----|----------------------|
| 0  | Normales Ende        |
| 12 | Falsche Zeilennummer |
| 20 | Schwerer Fehler      |

**TIME - JOB-Anweisung**

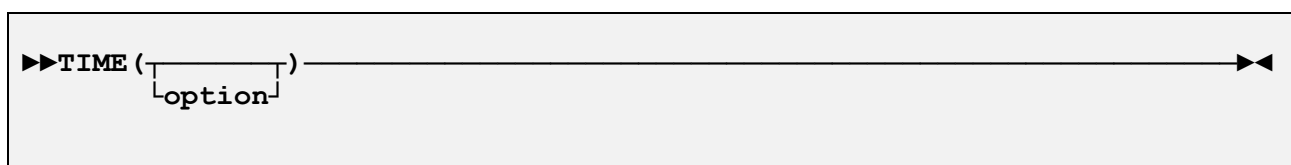
Legt die maximale CPU-Zeit für einen Job fest. Wenn nur Minuten angegeben werden, können die Klammern entfallen. Wird diese Zeit überschritten, wird der Job abgebrochen.

**1440 oder NOLIMIT**

Erlaubt die zeitlich unbeschränkte Nutzung der CPU.

**MAXIMUM**

Ermöglicht die als Maximum unter z/OS bestimmte Maximallaufzeit eines Jobs (357912 Minuten).

**TIME() – REXX Built-in-Function**

Übergibt die Zeit im 24-Stunden-Format in der Form std:min:sek (zum Beispiel 09:27:56). Durch nachfolgende Optionen kann das Ausgabeformat der Zeit beeinflusst oder eine Stoppuhr-Funktion benutzt werden.

| Option | Ausgabe                                           |
|--------|---------------------------------------------------|
| C      | CIVIL hh:mmxx in 12-Stunden-Format mit AM/PM      |
| E      | Elapsed, aufgelaufene Zeit seit dem letzten Reset |
| H      | Hours, Anzahl Stunden seit Mitternacht            |
| L      | Long, hh:mm:ss.aaaaaa (aaaaaa=Microsekunden)      |
| M      | Minutes, Anzahl Minuten seit Mitternacht          |
| N      | Normal, hh:mm:ss                                  |
| R      | Reset, setzt den Timer auf 0                      |
| S      | Seconds, Anzahl Sekunden seit Mitternacht         |

### Beispiele

```
say TIME() '16:25:43'
say TIME('L') '16:25:43.123456' /*möglich*/
say TIME('H') '16'
say TIME('M') '985'
say TIME('S') '1028'
say TIME('N') '16:25:43'
say TIME('C') '4:25pm'
```

### Stoppuhr-Funktion

```
say TIME('R') '0' /* Erster Aufruf */
/* Nach einer Pause von zirka einer Sekunde */
say TIME('E') '1.001063' /* oder ähnliches */
/* Nach einer Pause von zirka einer Sekunde */
say TIME('E') '2.003703' /* oder ähnliches */
```

### Hinweis

Wird die Stoppuhr-Funktion in einem Ausdruck mehrfach benutzt, führt sie zu keinen unterschiedlichen Zeitergebnissen.

## TRACE – REXX Instruction



Durch das Trace-Kommando wird unter anderem die Ablaufverfolgung eines REXX-Programmes erlaubt.

**zahl**

Im interaktiven TRACE wird für die angegebene Anzahl von Befehlen kein Haltepunkt gesetzt. Danach wird normal mit der Ablaufverfolgung fortgefahren.

**?**

Interaktiver Trace. Eine Zeile des Quellcodes wird gelesen, angezeigt, ausgeführt und anschließend wird die Steuerung an den Bediener übergeben.

**!**

Unterdrückung der Befehlsausführung. Für jeden unterdrückten Befehl wird ein RC 0 angenommen. Wird im interaktiven TRACE-Modus die Befehlsausführung unterdrückt, kann dies durch Eingabe einer beliebigen TRACE-Option mit einem führenden Rufzeichen (!) aufgehoben werden. Mehrfache !-Angaben innerhalb einer Prozedur bewirken immer das Gegenteil der aktuellen Situation. Generelles Ausschalten der Befehlsunterdrückung kann durch TRACE O oder TRACE erreicht werden. Diese Instruktion kann auch im interaktiven TRACE während des Ablaufes abgesetzt werden.

**Die Buchstabenkürzel bewirken im Einzelnen:**

| Kürzel | Bedeutung                                              |
|--------|--------------------------------------------------------|
| A      | Alle Befehle werden vor Ausführung gezeigt             |
| C      | Commands vor Ausführung werden getraced                |
| I      | Zwischenergebnisse bei der Auslösung von Ausdrücken    |
| N      | Protokolliert Befehle mit negativem Return Code        |
| O      | Der Ablaufverfolger wird ausgeschaltet                 |
| R      | Ergebnisanzeige des Befehls (wird sehr häufig benutzt) |
| S      | Syntaxcheck ohne Befehlsausführung                     |

**Ausgabeformat des TRACE**

Die unterschiedlichen TRACE-Optionen werden beim Debugging durch entsprechende Identifikationen gekennzeichnet. Jedes gemeldete Source-Statement wird mit seiner Zeilennummer ausgegeben. Ist die Zeilennummer größer 99999, wird links abgeschnitten und ein Fragezeichen vorangestellt (z.B.: bei Zeilennummer 100487 erscheint ?00487).

Alle gezeigten Sourcezeilen bekommen während des TRACE einen drei Byte langen Vorspann, um die Art des TRACE auszudrücken. Dieser kann sein:

**\*-\***

Befehlszeile im Sourcecode

**+++**

TRACE-Message. Kann ein Return Code ungleich 0 eines Kommandos oder ein Syntaxfehler in einer Instruktion sein.

**>>>**

Kennzeichnet das Resultat eines Ausdruckes (bei TRACE R), oder einer Wertzuordnung für eine Variable durch PARSE, oder einen zurückgegebenen Wert von einer Subroutine.



## TRACE() – REXX Built-in-Function

►►TRACE ( option ) —————►◄◄

Übergibt den aktuellen Modus der TRACE-Option (Ablaufverfolgung). Als Option kann ? oder !, beziehungsweise einer der Buchstaben A,C, E, F, I, L, N, O, R oder S sein.

Detaillierte Beschreibung kann unter dem TRACE-Befehl nachgelesen werden.

### Beispiele

```
say TRACE () '?R' /* möglich */
say TRACE ('O') '?R'
say TRACE ('?I') 'O'

:
alt_trace=trace()
trace '?r'
/* diverse Instruktionen */
trace alt_trace
:
```

## TRANS – ISPF DM Service

►►"TRANS FROMCCSID(fromccsid) TOCCSID(toccsid) FROMVAR(fromvar) —————►

—►" TOVAR(tovar) LENGTH(length) —————►◄◄

Konvertiert von einer Codepage in eine andere.

### FROMCCSID(fromccsid)

5-stellige Angabe eines Coded Character Sets Identifiers als Source

### TOCCSID(toccsid)

5-stellige Angabe eines Coded Character Sets Identifiers als Target

### FROMVAR(fromvar)

Sende-Variable

**TOVAR(tovar)**

Empfangs-Variable

**LENGTH(length)**

Länge der Sende-Variablen ( max: 32.767)

**Hinweis**

Folgende CCSID-Angaben sind möglich:

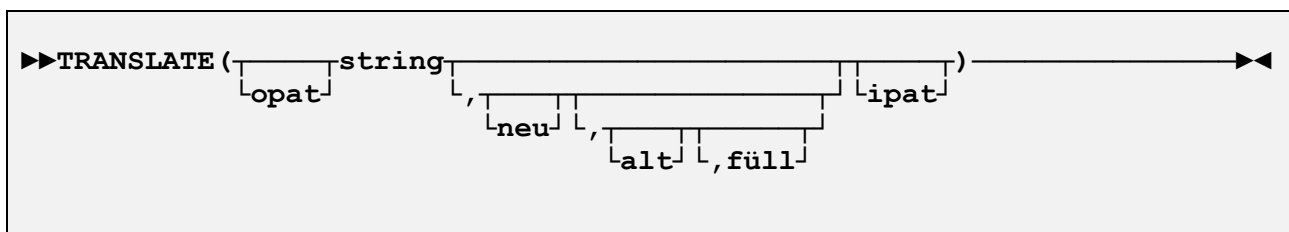
| CCSID | Character-Set | Code-page | Extended CCSID1 Unterstützung für Länder                              |
|-------|---------------|-----------|-----------------------------------------------------------------------|
| 00037 | 679           | 37        | USA, Kanada, Niederlande, Portugal, Brasilien, Australien, Neuseeland |
| 00273 | 697           | 273       | Deutschland, Österreich                                               |
| 00277 | 697           | 277       | Dänemark, Norwegen                                                    |
| 00278 | 697           | 278       | Finnland, Schweden                                                    |
| 00280 | 697           | 280       | Italien                                                               |
| 00284 | 697           | 284       | Spanien, Latein Amerika (spanisch)                                    |
| 00285 | 697           | 285       | Vereinigtes Königreich                                                |
| 00297 | 697           | 297       | Frankreich                                                            |
| 00420 | 235           | 420       | Arabisch                                                              |
| 00424 | 941           | 424       | Hebräisch                                                             |
| 00500 | 697           | 500       | Schweiz, Belgien                                                      |
| 00838 | 1176          | 838       | Thailand                                                              |
| 00870 | 959           | 870       | Latin-2                                                               |
| 00871 | 697           | 872       | Island                                                                |
| 00875 | 923           | 875       | Griechenland                                                          |
| 00880 | 960           | 880       | Kyrillisch                                                            |
| 01025 | 1150          | 1025      | Kyrillisch                                                            |
| 01026 | 1026          | 1026      | Türkei                                                                |
| 01047 | 697           | 1047      | Latin-1                                                               |
| 01123 | 1326          | 1123      | Ukraine                                                               |

| CCSID | Character-Set | Code-page | Extended CCSID2 Unterstützung (EURO) für Länder                       |
|-------|---------------|-----------|-----------------------------------------------------------------------|
| 00924 | 1353          | 0924      | Latin9                                                                |
| 01140 | 659           | 1140      | USA, Kanada, Niederlande, Portugal, Brasilien, Australien, Neuseeland |
| 01141 | 659           | 1141      | Österreich, Deutschland                                               |
| 01142 | 695           | 1142      | Dänemark, Norwegen                                                    |
| 01143 | 695           | 1143      | Finnland, Schweden                                                    |
| 01144 | 695           | 1144      | Italien                                                               |
| 01145 | 659           | 1145      | Spanien, Latein Amerika (spanisch)                                    |
| 01146 | 659           | 1146      | Vereinigtes Königreich                                                |
| 01147 | 659           | 1147      | Frankreich                                                            |
| 01148 | 659           | 1148      | Schweiz, Belgien                                                      |
| 01149 | 659           | 1149      | Island                                                                |
| 01153 | 1375          | 1153      | Latin2                                                                |
| 01154 | 1381          | 1154      | Kyrillisch                                                            |

|       |      |      |              |
|-------|------|------|--------------|
| 01155 | 1378 | 1155 | Türkei       |
| 01158 | 1388 | 1158 | Ukraine      |
| 01160 | 1395 | 1160 | Thailand     |
| 04899 | 1356 | 0803 | Hebräisch    |
| 04971 | 1371 | 0875 | Griechenland |
| 12712 | 1357 | 0424 | Hebräisch    |
| 16804 | 1461 | 0420 | Arabisch     |

Auf die Angaben für traditionell Chinesisch und dergleichen möchte ich verzichten

## TRANSLATE() – REXX Built-in-Function



Tauscht Zeichen in einem String gegen andere aus. Wird in der Funktion nur der String angegeben, wird in UPPERCASES (Großbuchstaben) umgewandelt.

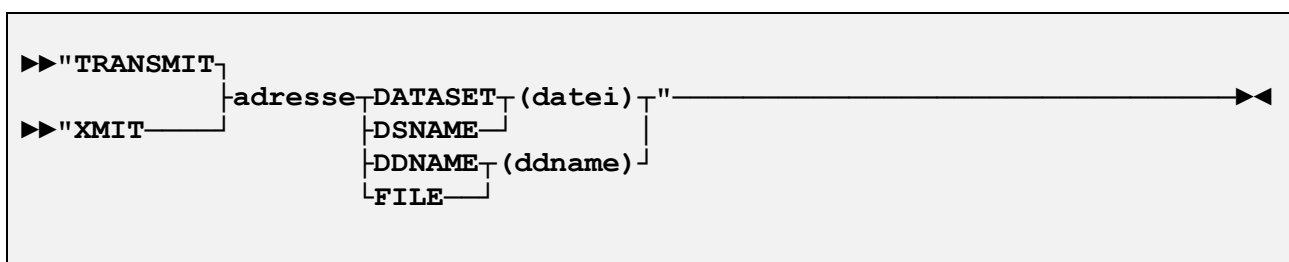
mit opat/ipat können Schablonen angegeben werden, welche regeln, wie gegebenenfalls Daten umzustellen sind. Die Angaben sind case-sensitiv.

### Beispiele

```
say TRANSLATE('klein') 'KLEIN'
say TRANSLATE('ADDA', 'B', 'D') 'ABBA'
say TRANSLATE('aba', 'z', 'a') 'zbz'
say TRANSLATE('abcde', '12', 'bd') 'a1c2e'
say TRANSLATE('abcde', '12', 'db') 'a2c1e'
say TRANSLATE('abcde', '12', 'abc', '.') '12.de'
say TRANSLATE('4231', 'abcd', '1234') 'dbca'

say TRANSLATE('Tt.Mm.CcJj', "19540917", "CcJjMmTt") 17.09.1954
```

## TRANSMIT – TSO Command (privilegiert)



TRANSMIT gehört zu den privilegierten TSO-Kommandos. Das heißt: je nach Definitionen im Unternehmen, kann der Befehl nicht von jedem ausgeführt werden. (siehe auch TSOEXEC-Kommando).

Der Befehl kann benutzt werden, um eine Datei zu einem Empfänger über das Spoolsystem zu senden, oder eine Ausgabedatei im XMIT-Format zu erstellen, welche dann auf den PC geladen auch per Mail verschickt werden kann. Es ist darauf zu achten, dass der Transfer auf den PC im Binärformat durchgeführt wird. Dies kann zwar ohne Hilfsmittel auf dem PC nicht gelesen werden, aber es gibt im Internet kostenlose XMIT-Manager, welche die EBCDIC-Files bearbeiten können.

Von den vielen möglichen Parametern betrachten wir hier die am häufigsten benutzten. Ansonsten sei auf das Manual "TSO/E Command Reference" verwiesen.

### **adresse**

Setzt sich im Allgemeinen aus dem Knotennamen und der Empfänger-UserID zusammen. Die beiden Angaben werden mit Punkt(.) getrennt.

### **DATASET(datei) | DSNAME(datei)**

Benennt die Datei, welche gesendet werden soll über ihren Namen.

### **DDNAME(ddname) | FILE(ddname)**

Liefert den DD-Namen einer preallocated Datei, die gesendet werden soll.

## **TRUNC() – REXX Built-in-Function**

```
►►TRUNC (zahl [,kommastellen])◄◄
```

Übergibt den integeren Teil einer Dezimalzahl und, sofern gewünscht, eine bestimmte Anzahl Nachkommastellen.

### **Beispiele**

```
say TRUNC (34.5) 34
say TRUNC (5.2564 ,2) 5.25
```

## **TSOEXEC – TSO Command**

```
►►"TSOEXEC tso-command"◄◄
```

Der Befehl kann benutzt werden, um in einem NICHT-autorisierten Umfeld wie ISPF autorisierte Kommandos abzusetzen wie TRANSMIT oder RECEIVE.

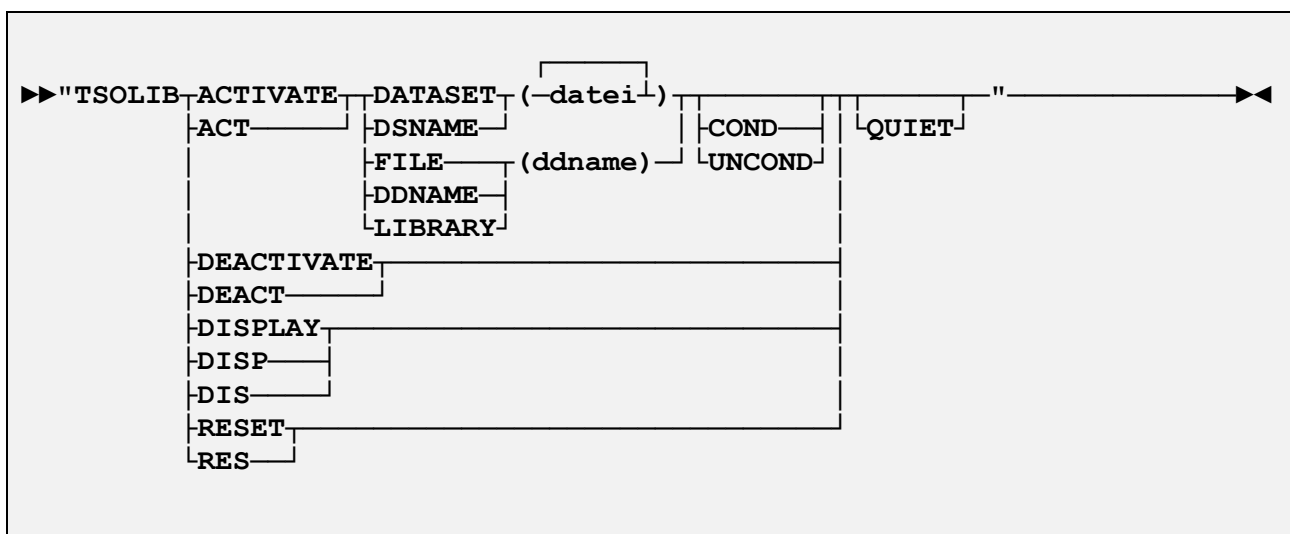
Durch TSOEXEC werden drei Variable mit folgender Bedeutung gefüllt:

- **SYSABNCD** Enthält den ABEND-Code
- **SYSABNRC** enthält den Reason Code
- **SYSCMDRC** enthält den Return Code des via TSOEXEC ausgeführten Befehls

## Beispiele

```
/* REXX
*/
"TSOEXEC XMIT NODE25A.SP1 DA(EDU.DATA)"
```

## TSOLIB – TSO Command



TSOLIB unterstützt einen zusätzlichen Search-Level, den TSO für die Suche nach Commands und Programmen benutzt. Die zugewiesenen Dateien werden in der Suchreihenfolge vorne eingeklinkt.

Die Standard-Suchreihenfolge unter TSO ist (von oben nach unten):

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| JOB LIB oder STEPLIB | Entsprechend der Definitionen in der LOGON-Routine            |
| Link Pack Area       | Entsprechend der Bibliotheken im LPALSTxx Member der Parmlib  |
| Linklist-Verkettung  | Entsprechend der Bibliotheken im LNKLISTxx Member der Parmlib |

TSOLIB kann sowohl für nicht-autorisierte als auch für autorisierte Bibliotheken benutzt werden.

## ACTIVATE

Die angegebenen Bibliotheken werden in der Suchreihenfolge vorne eingehängt.

**DEACTIVATE**

Entfernt die per ACTIVATE eingehängten Bibliotheken aus der Suchreihenfolge und stellt den alten Zustand wieder her.

**DATASET(datei) | DSNAME(datei)**

Liefert die Namen der Lademodulbibliothek(en), die bei der Suche nach Programmen zusätzlich berücksichtigt werden soll(en).

**FILE(ddname) | DDNAME(ddname) | LIBRARY(ddname)**

Liefert den DD-Namen, dessen dahinterstehende Dateien bei der Suche nach Programmen zusätzlich berücksichtigt werden sollen.

**COND | UNCOND**

Steuert, ob die Definition bedingungslos durchgeführt wird. Im Falle COND, erfolgt die Definition nur, wenn noch kein anderweitiger TSOLIB Befehl aktiv ist. Anderenfalls wird ein RC gesetzt.

**RESET | RES**

Löst die Zuweisung aller mit ACTIVATE definierten Bibliotheken wieder auf und kehrt zur Ursprungssituation zurück.

**DISPLAY | DISP | DIS**

Liefert Informationen über den aktuellen Stand der LoadLib-Zuweisungen. Auf diese Weise kann auch erkannt werden, wenn beispielsweise ISPF eine Erweiterung per LIBDEF für den DD-Namen ISPLLIB durchführt.

**QUIET**

Unterdrückt die Meldungen von TSOLIB. Ein Trapping durch OUTTRAP() kann nicht erfolgen. Ist ISPF aktiv, werden diverse Variable im Shared Pool abgelegt. Siehe hierzu das folgende Beispiel

**Beispiele**

```
/* REXX
*/
ADDRESS TSO "TSOLIB DISPLAY QUIET"
ADDRESS ISPEXEC "VGET (IKJTSM IKJTSM1 IKJTSM2 IKJTSM3...) SHARED"
SAY "IKJTSM " IKJADM
SAY "IKJTSM1" IKJTSM1
SAY "IKJTSM2" IKJTSM2
SAY "IKJTSM3" IKJTSM3
```

IKJTSM enthält die Anzahl der erzeugten Meldungen. IKJTSM1 und folgende enthalten die Meldungen.

```
TSOLIB ACT DSNAME('XWITT.DEMO-LOAD')
TSOLIB DISPLAY
IDY0022I Search order (by DDNAME) is:
IDY0023I DDNAME = SYS00101
```

```
ALLOC DD(MYLOAD) DA(MY.LOAD)
```

```
TSOLIB ACT DDNAME(MYLOAD)
TSOLIB DISPLAY
IDY0022I Search order (by DDNAME) is:
IDY0023I DDNAME = MYLOAD

/* REXX
*/
"ALLOC DD(MYLOAD) DA(MY.LOAD) SHR"
"TSOLIB ACT DDNAME(MYLOAD) "
:
Exit
```

### Hinweis:

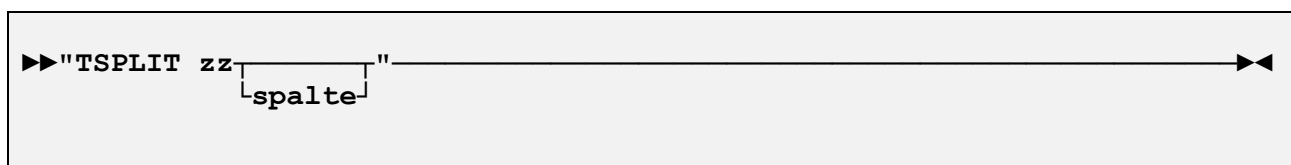
Generell ist die Nutzung eher fragwürdig. Der große Nachteil ist, dass das TSOLIB-Command nur native unter TSO (nicht aus einer REXX und schon gar nicht aus einer ISF-Umgebung) benutzt werden darf. Das würde man gegebenenfalls noch innerhalb eines LOGON-Ablaufs hinbekommen, für die reguläre Nutzung im laufenden Betrieb um vernünftig mit gekapselten Anwendungen arbeiten zu können, scheidet dies aber aus.

Leider bietet die IBM im IKJEFTxx keine Unterstützung für Dynamic Steplib (es gibt Software-Anbieter, welche das können). Die Lösung aus dieser Misere ist das Assembler-Programm \$TEPLIB. Es kann kostenlos als Quellcode von [www.cbttape.org](http://www.cbttape.org) heruntergeladen werden. Man muss es nur übersetzen, und in eine APF-Bibliothek legen. Das \$TEPLIB kann alles, was man sich in dieser Richtung von der IBM schon seit 40 Jahren wünscht. Ich arbeite damit seit beinahe 20 Jahren und hatte noch nie Probleme damit.

### Return Codes

- |    |                                                                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Verarbeitung erfolgreich                                                                                                                              |
| 4  | Es existiert keine entsprechende Bibliothek (bei DEACT)                                                                                               |
| 8  | Es gibt bereits eine aktive Zuweisung (im Falle von COND)                                                                                             |
| 16 | Unter dem angegebenen DD-Namen ist keine Datei zugewiesen                                                                                             |
| 20 | Schwerer Fehler. Wird von einer Fehlermeldung begleitet                                                                                               |
| 24 | Interner Fehler. TSOLIB ist nicht in der Lage, ein Recovery-Environment aufzubauen oder es ist ein während eines TSOLIB-Installation-Exit aufgetreten |
| 28 | Environment-Fehler                                                                                                                                    |
| 32 | Environment-Fehler. TSOLIB wurde als Commandprozessor aufgerufen                                                                                      |

### TSPLIT – ISPF Edit Macro



Trennt eine Zeile und schafft Platz, um eine neue Zeile einzufügen. Die Boundarygrenzen werden berücksichtigt. Daten außerhalb von Boundary werden beim Zeilenumbruch nicht mitgenommen.

**zz**

Zeigt über ein Label oder eine relative Zeilennummer auf die Zeile, in der der Befehl ausgeführt werden soll.

**spalte**

Spalte innerhalb der Zeile, an der die Trennung durchgeführt werden soll.

Werden zeilenzeiger und spalte nicht angegeben, wird für die Trennung die aktuelle Cursorposition genutzt.

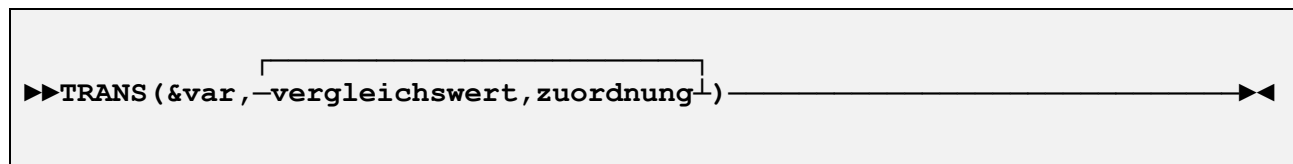
**Beispiel**

```
"tsplit .anfang 35"
```

Textsplit wird in der Zeile mit dem Label .ANFANG in Spalte 35 durchgeführt.

**Return Codes**

|    |                      |
|----|----------------------|
| 0  | Normales Ende        |
| 12 | Falsche Zeilennummer |
| 20 | Schwerer Fehler      |

**TRANS – ISPF Panel Function**

Kann für Wertzuweisungen genutzt werden, deren Ergebnis abhängig vom Inhalt anderer Variabler sein soll.

**&var**

benennt die Variable, die verglichen werden soll.

**vergleichswert**

ist die rechte Seite der Gleichung

**zuordnung**

ist der Wert, der einer Variablen zugeordnet wird, sofern der Vergleich positiv war.



**Beispiel**

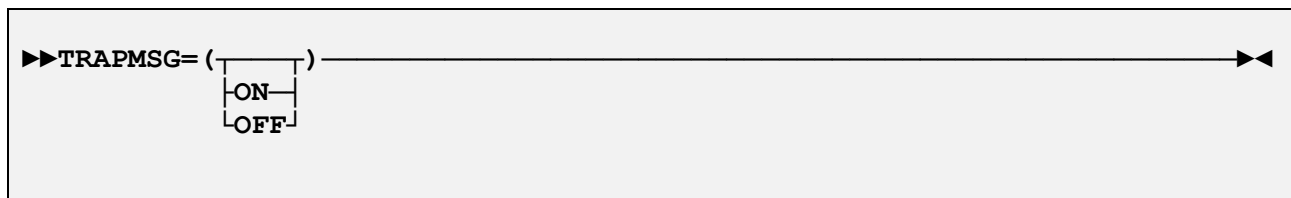
```

&RABATT=TRANS (&KDGRU ,
 010,0
 020,5
 030,10
 040,20
 050,30
 *, &RABATT)

```

Die Variable &KDGRU (Kundengruppe) wird auf ihren Inhalt überprüft. Wird einer der Vergleiche als richtig erkannt, wird der Variablen &RABATT der entsprechende Prozentwert zugewiesen. Trifft keiner der Vergleiche zu, bleibt die Variable &RABATT unverändert.

Dieses Vorgehen kann in der Praxis genutzt werden, um Felder abhängig zu den Inhalten bestimmter anderer Felder mit Werten zu füllen, ohne Eingaben von einem Benutzer zu fordern.

**TRAPMSG() – TSO extended Function**

Wenn das Message Trapping ausgeschaltet ist (und das ist Default), werden Commandoutputs generell auf den Bildschirm ausgegeben und können auch mit OUTTRAP() nicht eingefangen werden.

Ohne Parameter liefert die Funktion ihren aktuellen Status zurück (ON oder OFF).

**OFF**

Zurückgemeldet wird der aktuelle Status. Das Message Trapping wird ausgeschaltet.

**ON**

Zurückgemeldet wird der aktuelle Status. Das Message Trapping wird eingeschaltet.

**Beispiel**

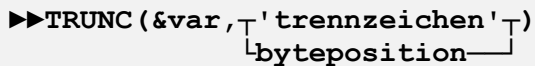
```

/* REXX
*/
OldTrapSettings = trapmsg("ON")
Call outtrap "EXECIO."
"alloc dd(sample) da(...) ..."
"execio 1 disk sample ..."
If rc > 2 then do
 Do $=1 to execio.0
 Say "MSG."$ ">>" execio.$
 end
end
else do

```

```
 call trapmsg "OFF"
 call outtrap "OFF"
end
```

## TRUNC – ISPF Panel Function



```
▶▶TRUNC(&var, 'trennzeichen',
 byteposition)
```

Abschneiden einer Zeichenfolge auf zwei verschiedene Arten. Zur Trennung kann das Vorkommen eines bestimmten Zeichens, oder eine Byteposition benutzt werden.

### 'trennzeichen'

ein oder mehrere Zeichen, die zur Trennung genutzt werden. Die Trennzeichen selbst werden weder in der zu füllenden Variablen, noch in .TRAIL hinterlegt.

### byteposition

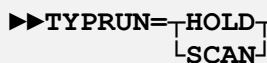
liefert die Anzahl Byte, die zum Füllen der Variablen genutzt werden. Der Rest wird in die Variable .TRAIL überführt.

### Beispiel

```
&STD=TRUNC(&ZTIME, ' : ')
```

Zur Bildung der Variablen &STD wird die Systemzeit ausgelesen und alles übernommen, bis ein Doppelpunkt erkannt wird.

## TYPRUN – JOB-Anweisung



```
▶▶TYPRUN=HOLD,
 SCAN
```

TYPRUN wird benutzt, um die Ausführung eines Jobs nach Übergabe an den Internal Reader zunächst nicht sofort durchführen zu lassen.

### HOLD

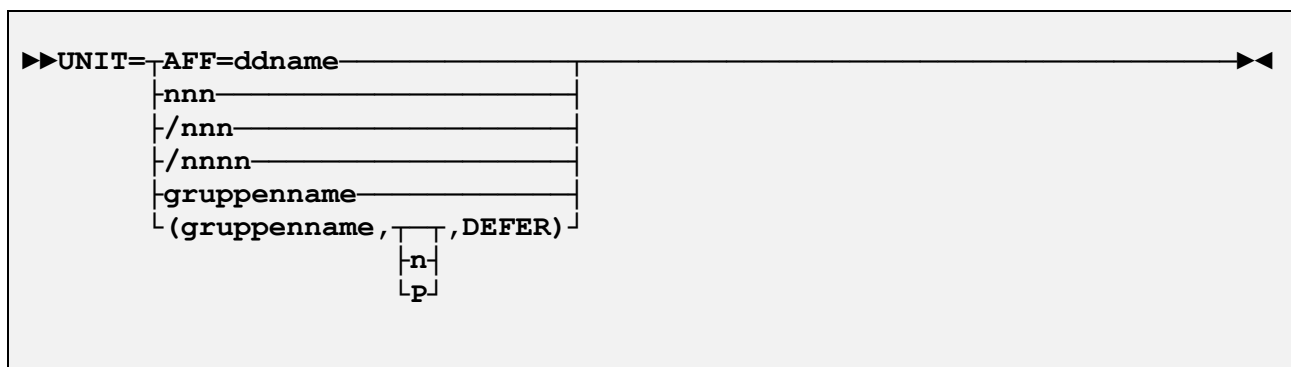
Führt zum Einlesen des Jobs und zur Übergabe in die INPUT-HOLD- QUEUE. Eine Verarbeitung erfolgt erst nach Freigabe durch den Operator (JES-Kommando RELEASE - \$A).

**SCAN**

zieht nach sich, dass der Job nur auf syntaktische Fehler geprüft, aber nicht ausgeführt wird (sinnvoll bei Neuerstellung).

**Hinweis**

Der SCAN-Parameter ist heute nicht mehr zeitgemäß. Die meisten Unternehmen verfügen über Tools wie beispielsweise JCLPLUS, welche zur Syntaxprüfung vor SUBMIT herangezogen werden können. Diese Tools sind in der Regel auch in der Lage, DISP-Parameter logisch zu prüfen und arbeiten auch mit IWS zusammen.

**UNIT – JCL DD Parameter**

Der Parameter muss für neu einzurichtende Dateien zwingend angegeben werden (ebenso bei bestehenden, nicht katalogisierten Dateien).

**AFF=ddname**

bezieht sich auf eine Gerätezuordnung innerhalb des angegebenen DD-Namens.

**gruppenname**

Der Gerätegruppenname muss bei der Systemgenerierung mit den zugehörigen Geräten beschrieben sein (z.B.: SYSDA=Platten allgemein, VIO für virtuelle Ein-/Ausgabe (Datei wird im Hauptspeicher abgebildet)). Darüber hinaus können kundenspezifische Erweiterungen stattfinden. Am besten zu klären mit der für Plattenverwaltung zuständigen Abteilung.

**nnn | /nnn | /nnnn**

Ist die HEX-Adresse des Gerätes. Dreistellige Gerätenummern dürfen, vierstellige müssen mit Schrägstrich angegeben werden.

**n**

Mit n kann die Anzahl der angeforderten Geräte, speziell bei Band-Einheiten, angegeben werden.

**P (parallel mounting)**

Die Anzahl der Geräte wird dem VOL-Parameter entnommen. Ist dies im UNIT- und VOL-Parameter angegeben, wird der höhere Wert genommen.

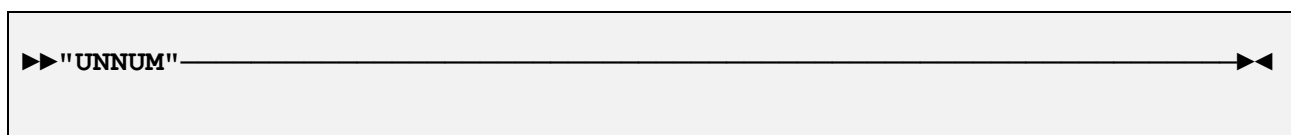
**DEFER**

Legt fest, dass der Datenträger erst angefordert wird, wenn eine Datei eröffnet wird. Wird DEFER verwendet, aber keine Angabe über die Anzahl der angeforderten Geräte gemacht, so sind nach dem Gruppennamen zwei Komata(,,) zu setzen.

**Beispiel**

```
//ddname DD UNIT=(TAPE,,DEFER)
```

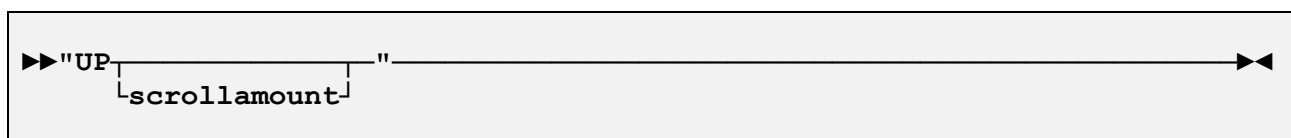
Die Montieranweisung für das Bandgerät erfolgt erst beim OPEN auf die Datei. Wird kein Satz ausgegeben, wird das Band nicht angefordert.

**UNNUM – ISPF Edit Macro**

Entfernt die Zeilensequenznummern und schaltet den Nummernmodus aus.

**Return Codes**

|    |                                  |
|----|----------------------------------|
| 0  | Normales Ende                    |
| 12 | Nummernmodus nicht eingeschaltet |
| 20 | Schwerer Fehler                  |

**UP – ISPF Edit Macro**

UP wird benutzt, um den Bildschirmausschnitt nach oben zu verschieben.

**scrollamount**

|        |                                       |
|--------|---------------------------------------|
| zahl   | Numerischer Wert zwischen 0 und 9999  |
| MAX    | Zum Datenende                         |
| HALF   | um eine halbe Seite                   |
| PAGE   | um eine ganze Seite                   |
| CURSOR | Die Cursor-Zeile wird die unterste.   |
| DATA   | Die oberste Zeile wird zur untersten. |

**Beispiele****"up max"**

Zeigt den Anfang der Datei

**Return Codes**

|    |                              |
|----|------------------------------|
| 0  | Normales Ende                |
| 2  | UP nicht mehr möglich        |
| 4  | Keine sichtbaren Datenzeilen |
| 8  | Keine Daten anzuzeigen       |
| 12 | Kein Scrollamount angegeben  |
| 20 | Schwerer Fehler              |

**UPDATE – DB2 Instruction**

```

▶▶"UPDATE table SET zuweisungen WHERE ausdrück—————▶◀

```

Führt Änderungen in DB2-Datensätzen durch. Die UPDATE-Instruktion ist sehr vielfältig. Wir betrachten hier nur eine von vielen Möglichkeiten.

**Table**

Name der Tabelle, in welcher die Änderung erfolgen soll

**SET zuweisungen**

Befüllt die Feld-Variablen zunächst mit den Werten, welche für den Update benutzt werden sollen

**WHERE ausdrück**

Stellt die Bedingung auf, welche erfüllt sein muss, damit es zum Update kommen kann.

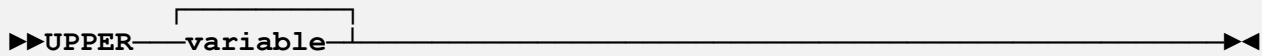
**Beispiel**

```

SQLSTMT = "UPDATE" Schema"."Table ,
 "SET VORNAME" !! "=" !! Vn !! "' , " ,
 "NACHNAME" !! "=" !! Nn !! "' , " ,
 "PLZ" !! "=" !! Plz !! "' , " ,
 "ORT" !! "=" !! Ort !! "' , " ,
 "STRASSE" !! "=" !! Str !! "' , " ,
 "GEBURTS_DATUM" !! "=" !! GebDat !! "' , " ,
 "PSEUDONYM" !! "=" !! Pseudo !! "' , " ,
 "HOBBY" !! "=" !! Hobby !! "' , " ,
 "ABTEILUNG" !! "=" !! Abt !! "' , " ,
 "TIMESTAMP" !! "=" !! NewTStmp !! "' , " ,
 "WHERE LASTUPD=' "LastUpdated" ' "

```

## UPPER – REXX Instruction



UPPER wird genutzt, um den Inhalt einer oder mehrerer Variabler in Großbuchstaben umzuwandeln. Prinzipiell gibt es hierfür auch eine Built-in-Funktion. Sollen aber mehrere Variable gewandelt werden, ist die Angabe aller Variablennamen in einer UPPER-Instruktion effizienter, als die Umwandlung jeder einzelnen durch die Built-in-Funktion TRANSLATE().

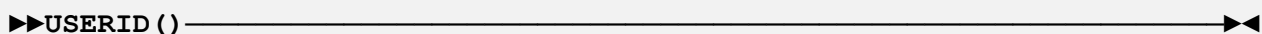
### Beispiele

```
a='Hallo, '
b='wie '
c='geht '
d='es '
e='euch '
say a b c d e /* Hallo, wie geht es euch */

say translate(a b c d e) /* HALLO, WIE GEHT ES EUCH */

upper a b c d e
say a b c d e /* HALLO, WIE GEHT ES EUCH */
```

## USERID() – TSO external



Vorausgesetzt, die Prozedur läuft unter einem TSO-Adressraum, wird die TSO-Benutzerkennung des Aufrufers übergeben.

### Beispiel

```
say USERID () 'SP1'
```

### Hinweise:

Läuft die Prozedur nicht in einem TSO-Adressraum, wird durch die Funktion USERID() einer der folgenden Werte gemeldet:

- Benutzerkennung
- Stepname
- Jobname

Dabei gilt: Ist die Benutzerkennung nicht bekannt, wird der Stepname ausgegeben. Ist beides nicht bekannt, erfolgt als Rückmeldung der Jobname.

## USER\_STATE – ISPF Edit Macro

```
►►" (variable) = USER_STATE"—————►◄

►►"USER_STATE = (variable)"—————►◄
```

Durch USER\_STATE können benutzerspezifische Informationen erfragt, oder gesetzt werden. Wird die aktuelle Information in einer Variablen hinterlegt, erfolgt dies in einem internen Format des Editors. Vor Manipulationen ist dringend abzuraten. Informationen, die übergeben werden sind die jeweiligen Zustände von

|          |             |         |          |
|----------|-------------|---------|----------|
| AUTOLIST | CURSOR      | NOTE    | RECOVERY |
| AUTONUM  | HEX         | NULLS   | STATS    |
| AUTOSAVE | IMACRO      | NUMBER  | TABS     |
| BOUNDS   | MASKLINE    | PACK    | TABSLINE |
| CAPS     | MODEL CLASS | PROFILE |          |

### =USER\_STATE

Der Benutzerstatus wird in einer Variablen hinterlegt.

### USER\_STATE=

Der Benutzerstatus wird gesetzt.

### Return Codes

|    |                 |
|----|-----------------|
| 0  | Normales Ende   |
| 20 | Schwerer Fehler |

## VALUE() – REXX Built-in-Function

```
►►VALUE (variable)—————►◄
```

Gibt Informationen über die angegebene Variable aus.

### Beispiele

**Angenommen:** DROP A3;A33=7;J=3;F='J'

```

say VALUE('F') 'J' /* Inhalt von "F" */
say VALUE(F) '3' /* Zeigt auf "J" */
say VALUE('a'j) 'A3'
say VALUE('a'j!!j) '7'

/* REXX
 Erzeugt aus einem Stem ISPF-taugliche Variablen
*/
V.1 = "Hund"
V.2 = "Katze"
V.3 ="Maus"
Do $=1 to 3
 Call value "V"$,V.$
end

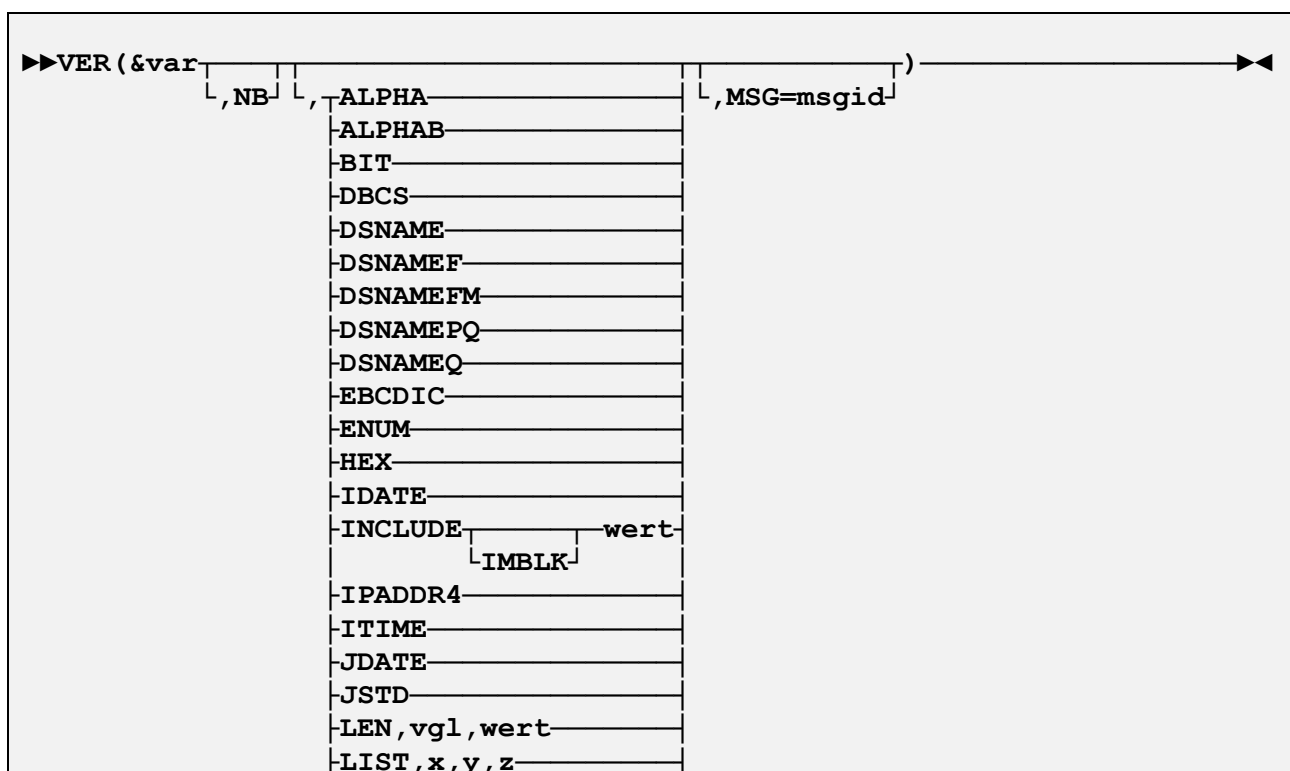
```

### Hinweise

Wird die Variable innerhalb der Funktion zwischen Apostrophe gesetzt, wird der ihr zugewiesene Wert übergeben. Enthält eine Variable den Namen einer anderen (usw.) und werden keine Apostrophe benutzt, erfolgt eine entsprechende Auflösung.

Für vieles, was mit VALUE erledigt werden kann, wird INTERPRET benutzt, obwohl es nicht nötig wäre. Manche Unternehmen untersagen die Nutzung von INTERPRET. Wer sich EINMAL mit VALUE auseinander gesetzt hat, vermisst INTERPRET nicht (natürlich gibt es für alles eine Ausnahme).

## VER – ISPF Panel Function





|                                      |
|--------------------------------------|
| <code>-LISTV, &amp;var</code>        |
| <code>-LISTVX, &amp;var</code>       |
| <code>-LISTX, x, y, z</code>         |
| <code>-NAME</code>                   |
| <code>-NAMEF</code>                  |
| <code>-NUM</code>                    |
| <code>-PICT, 'maske'</code>          |
| <code>-PICTCN, mz, fm, string</code> |
| <code>-RANGE, x, y</code>            |
| <code>-STDDATE</code>                |
| <code>-STDTIME</code>                |

Prüft den Inhalt eines Feldes. Die Prüfungen erfolgen der Reihe nach. Endet eine Prüfung negativ, wird ein Standardtext, oder eine vorgesehene Meldung ausgegeben. Anschließend wird (gegebenenfalls über den )REINIT-Teil) in den )BODY-Teil verzweigt und der Cursor auf das reklamierte Feld positioniert.

**&var**

ist die Variable, die überprüft werden soll.

**NB**

Prüft den Variableninhalt auf NONBLANK (ungleich Blank). Diese Angabe kann mit allen anderen Prüfungen kombiniert werden, da Prüfungen auf numerisch oder Länge nur durchgeführt werden, sofern der Variableninhalt nicht BLANK ist.

**ALPHA**

Erlaubt die Buchstaben in Groß- oder Kleinschreibung, sowie die Sonderzeichen HASH(#), AT(@) und Right Parenthesis()).

**ALPHAB**

Erlaubt die Buchstaben in Groß- oder Kleinschreibung, ohne irgendwelche Sonderzeichen.

**BIT**

Der Inhalt der Variablen muss binär sein (nur 0 und 1).

**DBCS**

Die Variable darf nur gültige DBCS-Characters enthalten.

**DSNAME**

Überprüft den Inhalt der Variablen auf einen gültigen Dateinamen hin.

**DSNAMEF**

**Überprüft den Inhalt der Variablen auf einen gültigen Dateinamen hin.**

**DSNAMEFM**

Wie DSNAMEF. Die Zeichen Asterisk(\*) und Percent(%) dürfen aber nur für Member-Angaben benutzt werden.

**DSNAMEPQ**

Wie DSNAMEQ. Beginnt die Datei mit einer öffnenden Klammer und endet sie nicht mit einer schließenden, wird diese hinzugefügt.

**DSNAMEQ**

Wie DSNAME. Falls der Dateiname mit einem Apostroph beginnt und kein endendes Apostroph vorhanden ist, wird dieses hinzugefügt.

**EBCDIC**

Der Feldinhalt muss aus EBCDIC-Zeichen bestehen.

**ENUM**

Der Feldinhalt muss numerisch sein, Vorzeichen ( $\pm$ ), Dezimalsymbol und Tausender-Trennzeichen (je nach Land '.' oder ',') sind erlaubt.

Siehe Details im Manual "ISPF Dialog Developers Guide and Reference".

**HEX**

Es dürfen nur HEX-Zeichen benutzt werden (0-9,A-F,a-f).

**IDATE**

International Date im Format YY/MM/DD. Die Trennzeichen werden nach den nationalen Gegebenheiten geprüft.

**INCLUDE**

Zwei Feldprüfungen können gemischt werden. Zusätzlich bewirkt IMBLK, dass eingebettete BLANKS erlaubt werden. Als wert kann ALPHA, ALPHAB und/oder NUM kodiert sein. Mindestens eine Angabe ist Pflicht.

**VER (&var , NB , INCLUDE , IMBLK , ALPHA , NUM)**

var ist ein Pflichtfeld. Inhaltlich darf sie Buchstaben (ALPHA), Ziffern (NUM) und eingeschlossene Blanks (IMBLK) führen.

**IPADDR4**

Muss eine IP-Adresse im Version-4-Format sein. Der erste Wert kann zwischen 0 und 223 liegen. Die restlichen drei zwischen 0 und 255. IPADDR4 unterstützt kein CIDR (Classless Inter Domain Routing).

**ITIME**

International Time im Format HH:MM (5-stellig).

**JDATE**

Julian Date 6-stellig im Format YY.DDD.

**JSTD**

Julian Standard Date 8-stellig im Format YYYY.DDD.

**LEN,vgl,wert**

Die inhaltliche Länge wird überprüft. Als Vergleichsoperanden stehen die üblichen Symbole wie >, <, = etc. zur Verfügung.

Werden die Sonderzeichen als Vergleichsoperatoren benutzt, müssen diese zwischen Apostrophe eingeschlossen werden.

```
VER (&VAR,LEN,'>=' ,5)
```

```
VER (&VAR,LEN,GE,5)
```

**LIST,x,y,z**

Die Variable muss einem der angegebenen Werte entsprechen.

**LISTV**

Die zu prüfenden Werte werden in einer Variablen hinterlegt.

**LISTVX**

Die Werte in der Variablen dürfen nicht vorkommen.

**LISTX,x,y,z**

Die Variable darf keiner der folgenden Werte sein.

**NAME**

Prüft auf einen gültigen Membernamen.

**NAMEF**

Wie NAME, jedoch dürfen Asterisk(\*) und Percent(%) enthalten sein.

**NUM**

Der Inhalt der Variablen muss numerisch sein.

**RANGE,x,y**

Der Inhalt muss numerisch sein und im Bereich x und y liegen.

**PICT,'maske'**

Der Inhalt der Variablen muss mit der angegebenen Maske übereinstimmen. Als Zeichen sind erlaubt (Großschreibung!):

|          |                                         |
|----------|-----------------------------------------|
| 9 oder N | Ziffern                                 |
| A        | Groß- oder Kleinbuchstaben, #, @ oder ) |
| X        | HEX-Werte                               |
| C        | beliebige Zeichen                       |

Alle übrigen Zeichen müssen so in der Variablen stehen, wie sie in der Maske festgelegt sind.

### **PICTCN,mz,fm,string**

Erlaubt die Überprüfung des Inhalts auf bestimmte Konstanten.

#### **mz (Masken-Zeichen)**

Jedes Zeichen repräsentiert sich selbst. Wird eines der folgenden Sonderzeichen als Maskenzeichen benutzt, muss es zwischen Apostrophe eingeschlossen werden: ^, =, , >, <, ), (, '

Als Mask Charakter darf keines der folgenden Zeichen benutzt werden: C, A, N, 9, X, c, a, n, x

#### **fm (Feld Maske)**

Eine Kombination von Konstanten und Maskenzeichen

#### **string**

Eine Kombination aus Konstanten und Picture-Strings.

|   |                                |
|---|--------------------------------|
| C | beliebig                       |
| A | Alphabetic (A-Z, a-z, #, \$, @ |
| N | Numerisch                      |
| 9 | Numerisch (wie N)              |
| X | Hexwert                        |

Maskenzeichen, Feldmaske und String können auch über Variablen geliefert werden.

### **Beispiele**

```
VER (&V1,PICTCN,'^','VV^^LL^^',VVNNLLNN)
```

Enthält &V1 den Wert VV01LL11, verläuft die Prüfung positiv. Wäre der Inhalt von &V1 aber VV03LLA5, wäre die Prüfung negativ, weil das A an Stelle 7 nicht numerisch ist.

```
VER (&V1,PICTCN,*,OS*****,OSNNNAN)
```

Maskenzeichen ist Asterisk(\*), Konstante sind O und S, Picture-String-Characters sind N für numeric und A für Alphabetic. Der Inhalt OS39027 würde reklamiert werden, weil die 2 kein Alphazeichen ist.

### **STDDATE**

Standard Datum 10-stellig. US-amerikanisch wäre das Format YYYY/MM/DD.

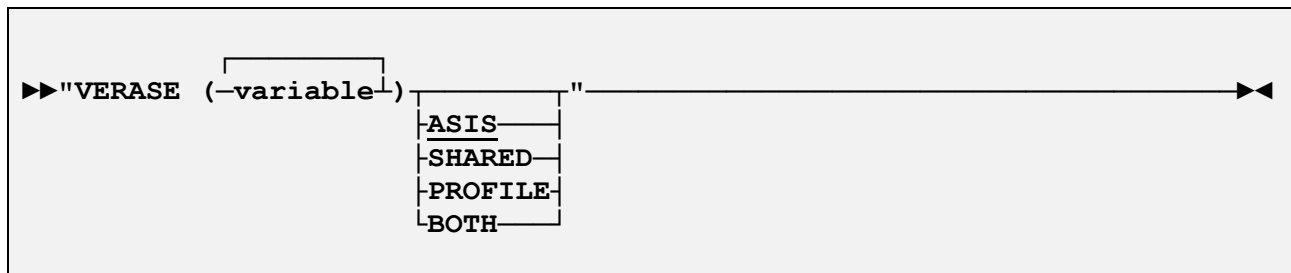
### **STDTIME**

Standardzeit. US-amerikanisch wäre das Format HH:MM:SS

### **MSG=msgid**

Identifikation der Nachricht, welche im Negativfall ausgegeben werden soll.

## VERASE – ISPF DM Service



Löscht eine Variable aus dem Pool. Systemvariablen, oder nicht veränderbare können nicht gelöscht werden.

### variable

benennt die Variable(n), die aus dem SHARED oder PROFILE Pool oder beiden gelöscht werden soll(en).

### ASIS

sucht zunächst im SHARED Pool. Ist die Variable dort nicht vorhanden, wird im PROFILE Pool gelöscht.

### SHARED

Die Variablen werden nur aus dem SHARED Pool ausgetragen.

### PROFILE

Die Variablen werden aus dem PROFILE Pool der aktuellen Applikation ausgetragen.

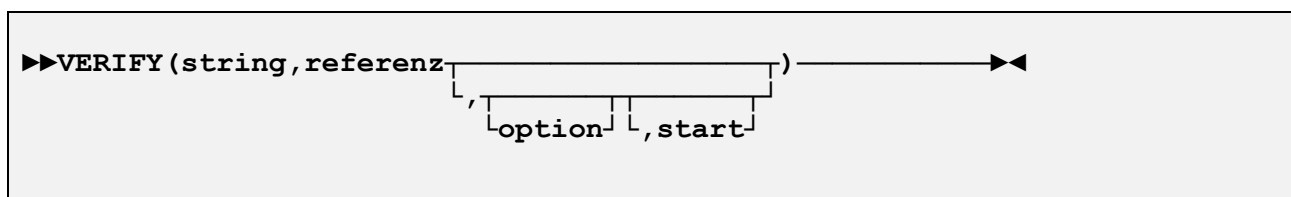
### BOTH

Die Variablen werden sowohl aus dem SHARED, als auch aus dem PROFILE Pool ausgetragen.

### Return Codes

- 0     Normales Ende
- 12    Eine oder mehrere Variablen nicht gefunden
- 20    Schwerer Fehler

## VERIFY() – REXX Built-in-Function



Überprüft, ob eine Zeichenfolge ausschließlich aus bestimmten Zeichen gebildet wurde. Durch die Option N(omatch) werden Abweichungen gemeldet (Standard), durch M(atch) Übereinstimmungen. Rückgemeldet wird die Position im String, bei der Übereinstimmung oder Abweichung erkannt wurde.

**Option**

M für Match (erste Übereinstimmung)

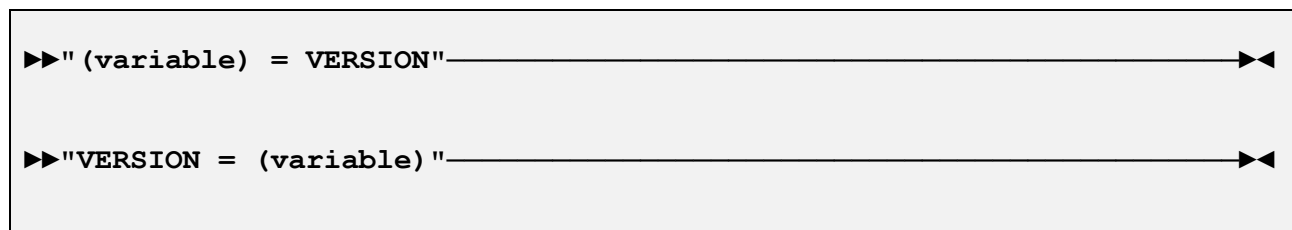
N für NoMatch (erste Abweichung)

**Start**

Byteposition, ab der die Überprüfung stattfinden soll.

**Beispiele**

|                                                    |   |
|----------------------------------------------------|---|
| <code>say VERIFY('123', '1234567890')</code>       | 0 |
| <code>say VERIFY('1W3', '1234567890')</code>       | 2 |
| <code>say VERIFY('A2B', '1234567890')</code>       | 1 |
| <code>say VERIFY('1a34', '1234567890', 'M')</code> | 1 |
| <code>say VERIFY('1a34', '1234567890', 'N')</code> | 2 |

**VERSION – ISPF Edit Macro**

```
>>'(variable) = VERSION'>>
>>'VERSION = (variable) '>>
```

Setzt oder übergibt die aktuelle Versionsnummer eines Members.

**=VERSION**

Die aktuelle Versionsnummer wird in einer Variablen hinterlegt.

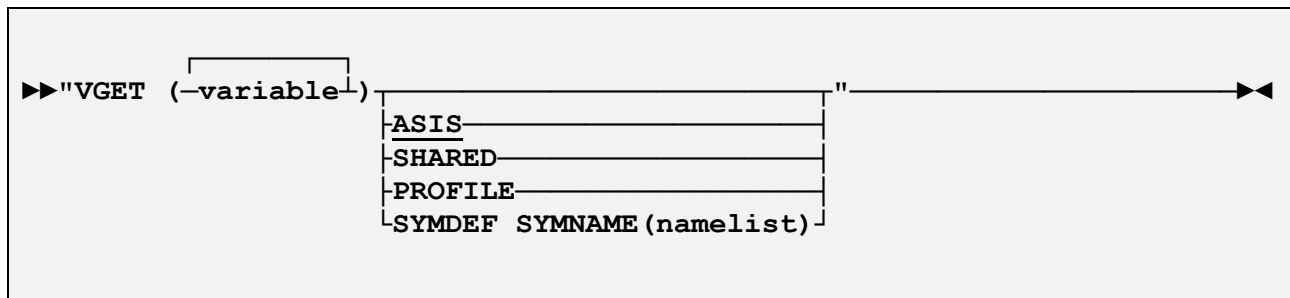
**VERSION=**

Die Versionsnummer wird entsprechend in den Memberstatistiken hinterlegt.

**Return Codes**

- |    |                                                         |
|----|---------------------------------------------------------|
| 0  | Normales Ende                                           |
| 8  | Stats-Modus ist ausgeschaltet, Befehl wird ignoriert    |
| 12 | Falscher Wert, Versionsnummer muss >= 00 und <= 99 sein |
| 20 | Schwerer Fehler                                         |

## VGET – ISPF DM Service



Kopiert eine Variable aus dem SHARED oder PROFILE Pool in den FUNCTION Pool der Applikation. Existiert dort eine gleichnamige Variable bereits, wird sie überschrieben.

### variable

benennt eine oder mehrere Variable, die kopiert werden sollen.

### ASIS

kopiert die Variable aus dem SHARED Pool, oder, sofern sie dort nicht vorhanden ist, aus dem PROFILE Pool. Dieser Operand wird angenommen, wenn keine Angabe erfolgt.

### SHARED

Die Variable wird aus dem SHARED Pool kopiert.

### PROFILE

Die Variable wird aus dem PROFILE Pool kopiert. Eine gleichnamige Variable im SHARED Pool wird gelöscht, auch wenn sie im PROFILE Pool nicht gefunden wird.

### SYMDEF SYMNAME(namelist)

Die Liste der Namen aus SYMNAME wird aus den System Symbols gefüllt.

Ein Grund für SYMNAME kann darin liegen, dass ein Symbol namensgleich zu einem reservierten Wort oder einer Read-Only-Variablen ist. In diesem Fall muss eine andere Variable benutzt werden:

```
VGET (CLONE) SYMDEF SYMNAME(SYSCClone)
```

### Hinweise

Die Länge eines VGET-Statements ist auf 255 Byte begrenzt.

Soll eine Variable aus einem Pool (SHARED oder PROFILE) kopiert werden, die dort nicht existiert, wird der Inhalt einer bereits existierenden Variablen trotzdem verändert.

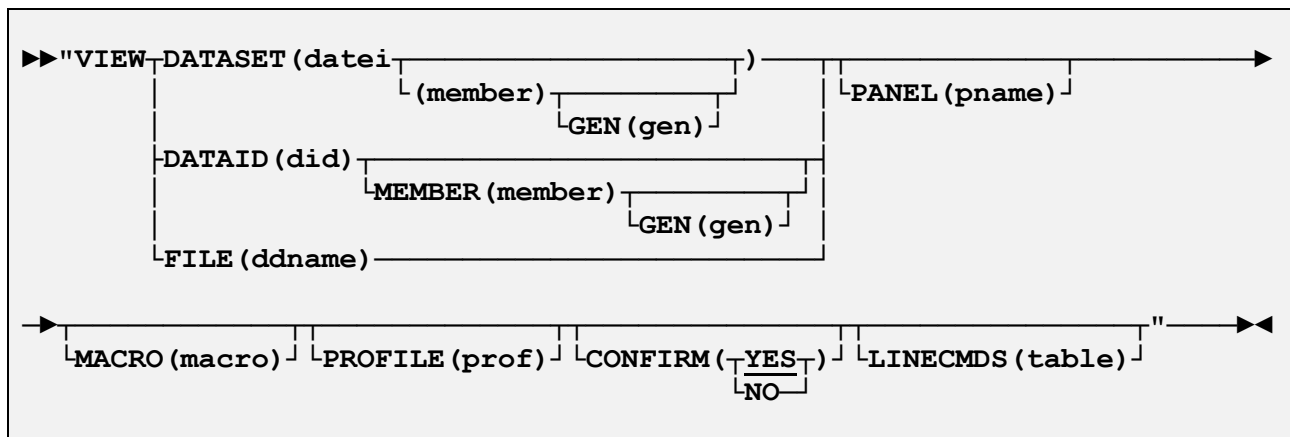
Bei CHARACTER-Variablen wird der Inhalt auf BLANK gesetzt, binäre oder HEX-Variable werden auf 0 gesetzt.

### Return Codes

0      Normales Ende

- 8 Variable nicht gefunden
- 16 Kopierfehler
- 20 Schwerer Fehler

## VIEW – ISPF DM Service



ruft den ISPF-Editor und präsentiert gleichzeitig die hinter dem Schlüsselwort DATASET angegebene Datei. Wird EDIT aus einer Funktion (REXX etc.) aufgerufen, kann der Anwender nicht wie gewohnt durch =3.4 im PDF-Dialog verzweigen. Der Editor kann durch END, RETURN oder CANCEL verlassen werden. Die Ablaufsteuerung wird immer an die Funktion zurückgegeben, die entscheidet, was weiter zu tun ist.

Während der EDIT-Session sind alle Edit-Befehle verfügbar.

### Datei | datei(member)

Name der anzuzeigenden Datei. Wird der Name einer sequentiellen Datei angegeben, zeigt BROWSE die Daten.

Wird der Name einer PO-Datei ohne Member angegeben, liefert BROWSE das MembERVERZEICHNIS. Wird das Member mit angegeben, werden die Memberdaten gezeigt.

### DATAID(did)

DatenID welche per LMINIT zugewiesen wurde

### FILE(ddname)

DD-Name, unter dem das Dataset allokiert ist.

### GEN(gen)

Generation eines PO/E-Type2-Datasets mit Angabe der relativen Generationsnummer (0, -1, -2, etc.).

### PANEL(pname)

Panelname, sofern ein anderes als das Standardpanel der IBM benutzt werden soll.



**MACRO(macro)**

Name des Initialisierungsmacros.

**PROFILE(prof)**

Name des zu benutzenden Profils

**CONFIRM YES | NO**

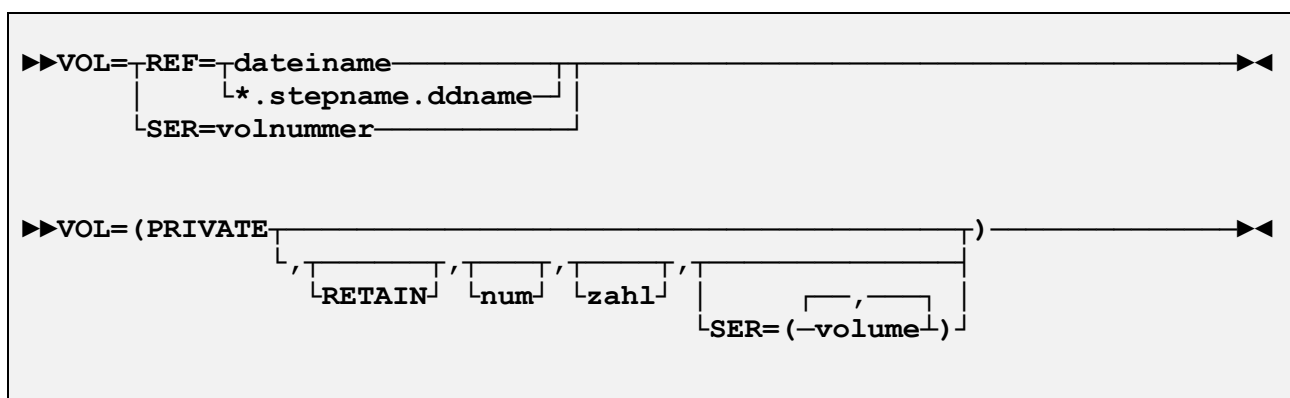
Bestätigung für REPLACE etc.

**LINECMDS(table)**

Name der Linecommand-Table sofern eine benutzt werden soll.

**Return Codes**

- 0     Normales Ende.
- 4     Normales Ende. Daten wurden nicht gesichert.
- 10    Generationismember nicht gefunden
- 12    LOCK war auf YES gesetzt
- 14    Datei, Member oder Unix File in USE
- 16    Kein Member matching entsprechend dem Pattern oder keine Member vorhanden
- 18    Es wurde eine VSAM-Datei angegeben und in der ISPF Configuration Table ist dies nicht erlaubt
- 20    Schwerer Fehler

**VOL – JCL DD-Parameter**

Bei katalogisierten oder temporären Dateien kann dieser Parameter entfallen.

**VOL=REF=dateiname**

Bezieht sich auf die Volume-Nummer der Datei. Sie muss katalogisiert, oder durch PASS von einem vorangegangenen Step übergeben worden sein.

**VOL=REF=\*.stepname.ddname**

Bezieht sich auf die VOLUME in der unter *ddname* angegebenen DD-Anweisung. Innerhalb des Steps kann stepname entfallen. Wenn VOL=REF= angegeben wird, kann der UNIT-Parameter entfallen.

**PRIVATE**

Dem Datenträger darf keine Ausgabe zugeordnet werden, außer, er wird ausdrücklich angefordert.

**RETAIN**

Bei Tapes werden diese bei Stepende nicht automatisch entladen.

**num**

Gibt an, die wievielte Kassette einer MULTI-VOL-Datei angesprochen wird.

**zahl**

Maximale Anzahl Volumes, die einer Ausgabedatei zugeordnet werden.

**SER=(..**

Gibt die VOL-Nummer an, auf der sich die Datei befindet, oder eingerichtet werden soll. Wird nur eine Volumenummer angegeben, können die Klammern entfallen.

**VOLUME – ISPF Edit Macro**

**►►" (var1,var2,var3) = VOLUME"\_\_\_\_\_◄◄**

Liefert Informationen über den Datenträger, auf dem die Datei liegt, welche der Editor im Zugriff hält.

**Var1**

VOLSER des Datenträgers. Bei Multi-Volume-Datasets ist dies der Name des ersten Datenträgers.

**Var2**

Anzahl der Datenträger, auf denen die Datei liegt (2 Byte).

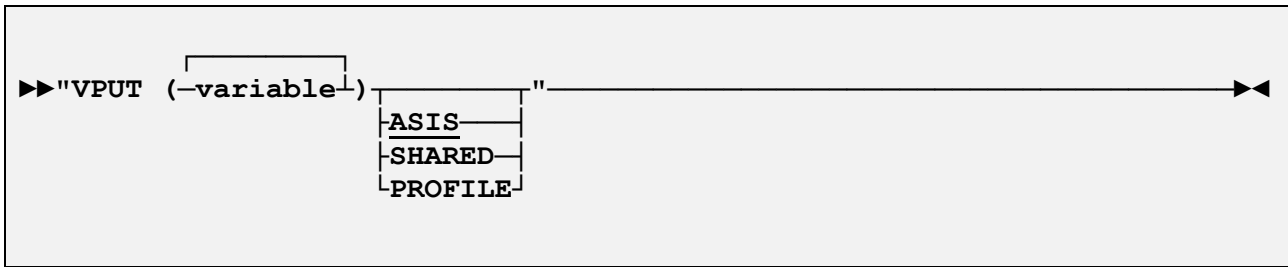
**Var3**

Name des Volumes der Originaldatei

**Return Codes**

- |    |                                                                                             |
|----|---------------------------------------------------------------------------------------------|
| 0  | Normales Ende                                                                               |
| 4  | Die Datei ist ein Multi-Volume-Dataset. ZEDMVOL enthält die Namen aller Volumes (maximal 6) |
| 20 | Schwerer Fehler                                                                             |

## VPUT – ISPF DM Service



Kopiert eine Variable aus dem FUNCTION Pool in den SHARED oder PROFILE Pool der Applikation. Existiert dort eine gleichnamige Variable bereits, wird sie überschrieben.

### variable

benennt eine oder mehrere Variable, die kopiert werden sollen.

### ASIS

kopiert die Variable in den Pool, in dem sie vorhanden ist, oder in den SHARED Pool, sofern sie in keinem der beiden Pools gefunden wird. Ist die Variable sowohl im SHARED als auch im PROFILE Pool vorhanden, wird sie nur in den SHARED Pool kopiert.

### SHARED

Die Variable wird in den SHARED Pool kopiert.

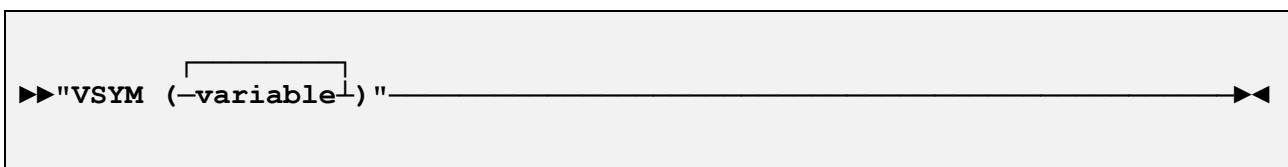
### PROFILE

Die Variable wird in den PROFILE Pool kopiert. Eine gleichnamige Variable im SHARED Pool wird gelöscht.

### Return Codes

- |    |                                                |
|----|------------------------------------------------|
| 0  | Normales Ende                                  |
| 8  | Variable nicht gefunden                        |
| 16 | Kopierfehler der Variablen in den Profile Pool |
| 20 | Schwerer Fehler                                |

## VSYM – ISPF DM Service



Interpretiert eine oder mehrere Variable und substituiert die System Symbols, welche darin enthalten sind.

**variable**

benennt die Variable, die substituiert werden soll.

**Beispiel**

```
/* REXX
*/
DSNHLQ = "SYS9.&SYSPLEX"
Address ispxexec "VSYM (DSNHLQ) "
Say DSNHLQ "SYS9.SYSPLEX2" /* möglich */
```

**Return Codes**

- 0    Normales Ende
- 8    Variable nicht gefunden
- 16  Kopierfehler der Variablen in den Profile Pool
- 20  Schwerer Fehler

**WORD() – REXX Built-in-Function**

►►WORD(string,zahl)◄◄

Übergibt das durch zahl bestimmte Wort aus einer Zeichenfolge. Ist zahl größer als die Anzahl von Worten im String, wird ein Nullstring übergeben. Diese Funktion ist gleichzusetzen mit der Funktion SUBWORD(string,zahl,1).

**Beispiele**

```
say WORD('Bitte noch ein Bier',1) 'Bitte'
say WORD('Bitte noch ein Bier',3) 'ein'
```

**WORDINDEX() – REXX Built-in-Function**

►►WORDINDEX(string,zahl)◄◄

Meldet die Beginnposition des entsprechenden Wortes im String.

**Beispiele**

```
say WORDINDEX('Bitte noch ein Bier',2) 7
say WORDINDEX('Bitte noch ein Bier',7) 0
```

## WORDLENGTH() – REXX Built-in-Function

►►WORDLENGTH(string,zahl)◄◄

Übergibt die Länge des betreffenden Wortes. Ist zahl größer als die gesamt vorhandenen Worte im String, wird 0 gemeldet.

### Beispiele

```
say WORDLENGTH('Bitte noch ein Bier',2) 4
say WORDLENGTH('Bitte noch ein Bier',3) 3
say WORDLENGTH('Bitte noch ein Bier',6) 0
```

## WORDPOS() – REXX Built-in-Function

►►WORDPOS(suchbegriff,string)◄◄  
[suchbeginn]

Durchsucht eine Zeichenfolge nach Vorhandensein einer Wortfolge und gibt die Position des ersten betreffenden Wortes im String zurück.

### Beispiele

```
say WORDPOS('es','jetzt ist es Zeit') 3
say WORDPOS('ES','jetzt ist es Zeit') 0
say WORDPOS('ist es','jetzt ist es Zeit') 2
say WORDPOS('jetzt es','jetzt ist es Zeit') 0
say WORDPOS('be','to be or not to be') 2
say WORDPOS('be','to be or not to be',3) 6
```

## WORDS() – REXX Built-in-Function

►►WORDS(string)◄◄

Meldet die Anzahl Worte im String.

### Beispiele

```
say WORDS('Es ist vollbracht') 3
```

```
say WORDS(' ')
```

0

## XEQ – JCL JES-Statement

```
/*XEQ systemname
```

Bestimmt ein von der Eingabe des Jobs abweichendes System für die Verarbeitung. Die Angabe ist gleichbedeutend mit /\*ROUTE XEQ.

### Hinweis

Sind mehrere XEQ-Statements im Job, zählt nur das letzte.

## XRANGE() – REXX Built-in-Function

```
►►XRANGE (beginn ende)
```

Erzeugt einen String mit allen EIN-BYTE-CODES zwischen beginn und ende. Die Standardannahme bei fehlendem beginn ist '00'x. Wenn der ende-Wert größer dem beginn-Wert ist erfolgt die Zusammensetzung des Strings zunächst bis 'ff'x und über '00'x zum angegebenen ende-Wert. Beide Werte (beginn und ende) müssen, wenn angegeben, ein Byte langer Begriff sein.

### Beispiele

```
say XRANGE('a','f') 'abcdef'
say XRANGE('03'x,'07'x) '0304050607'x
say XRANGE(, '04'x) '0001020304'x
say XRANGE('i','j') '898A8B8C8D8E8F9091' /*EBCDIC*/
say XRANGE('FE'x,'02'x) 'FEFF000102'
```

## XSTATUS – ISPF Edit Macro

```
►►"(variable) = XSTATUS"
```

```
►►"XSTATUS zz = variable"
```

Setzt oder übergibt den Status einer Zeile (ausgeblendet oder nicht ausgeblendet). Um ganz generell Zeilen auszublenden, sollte EXCLUDE genutzt werden. Um in ausgeblendeten Bereichen Zeilen sichtbar zu machen, ist FIND der bessere Weg. XSTATUS ist sinnvoll, um den Status einer Zeile vor Verarbeitung festzustellen, und diesen Status wiederherzustellen, sofern er durch die Verarbeitung verändert wurde.

### **=XSTATUS**

Der Zeilenstatus (X für eXcluded, NX für Not eXcluded) wird in einer Variablen hinterlegt.

### **XSTATUS zz=**

Der Zeilenstatus wird gesetzt.

### **Return Codes**

- |    |                                                                                                                               |
|----|-------------------------------------------------------------------------------------------------------------------------------|
| 0  | Normales Ende                                                                                                                 |
| 8  | Zuweisung des Zeilenstatus NX nicht möglich Die Zeile enthält ein interpretiertes aber noch nicht ausgeführtes Zeilenkommando |
| 12 | Falsche Zeilennummer                                                                                                          |
| 20 | Schwerer Fehler                                                                                                               |

## **X2B() – REXX Built-in-Function**

►►X2B(hexstring)◄◄

Konvertiert einen Hexstring nach Binär.

### **Beispiele**

```
say X2B('C3') '11000011'
say X2B('F9') '1001'
```

## X2C() – REXX Built-in-Function

►►X2C(hexstring)◄◄

Konvertiert einen Hexstring nach Character.

### Beispiele

```
say X2C('F8F3C1') '83A'
say X2C('F8 F3 C1') '83A'
say X2C('F') ' ' /* 0F ist nicht darstellbar */
```

## X2D() – REXX Built-in-Function

►►X2D(hexstring)◄◄

Konvertiert einen Hexstring nach Decimal.

### Beispiele

```
say X2D('0F') '15'
say X2D('F0') '240'
```

## Z-Variablen des Dialog Managers

Der besseren Übersicht wegen sind die Systemvariablen zu logischen Gruppen zusammengefasst. Der Aufbau der Tabellen sieht folgendermaßen aus:

|          |                                                                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Spalte 1 | Name der Systemvariablen                                                                                                                                        |
| Spalte 2 | Benennt den Pool, in dem sich die Variable befindet.<br>F Function Pool<br>S Shared Pool<br>P Profile Pool<br>A in irgendeinem der Pools                        |
| Spalte 3 | Variablentyp. Hier sind folgende Abkürzungen möglich:<br>E Eingabevariable<br>A Ausgabevariable<br>EA Ein-/Ausgabevariable<br>UV Unveränderbare Ausgabevariable |
| Spalte 4 | Länge der Variablen in Byte                                                                                                                                     |
| Spalte 5 | Beschreibung der Variablen                                                                                                                                      |



## Datum und Zeit

| Name     | Pool | Typ | Länge | Beschreibung                                         |
|----------|------|-----|-------|------------------------------------------------------|
| ZDATE    | S    | UV  | 8     | Tagesdatum, Format sprachabhängig                    |
| ZDATEF   | S    | UV  | 8     | Tagesdatum: DD/MM/YY                                 |
| ZDATEFD  | S    | UV  | 8     | Wie ZDATEF aber sprachabhängig                       |
| ZDATESTD | S    | UV  | 8     | Tagesdatum, Format sprachabhängig mit Jahr 4-stellig |
| ZDAYOFWK | S    | UV  | 8     | Wochentag                                            |
| ZDAY     | S    | UV  | 2     | Tagesdatum: DD                                       |
| ZJDATE   | S    | UV  | 6     | Tagesdatum julianisch: YY.DDD                        |
| ZJ4DATE  | S    | UV  | 8     | Tagesdatum julianisch (YYYY.DDD                      |
| ZMONTH   | S    | UV  | 2     | Tagesdatum: MM                                       |
| ZTIME    | S    | UV  | 5     | Uhrzeit: HH:MM                                       |
| ZTIMEL   | S    | UV  | 10    | Uhrzeit mit Hundertstel Sekunde                      |
| ZYEAR    | S    | UV  | 2     | Tagesdatum: YY                                       |

## Generelle Informationen

| Name     | Pool | Typ | Länge | Beschreibung                                                                                                                 |
|----------|------|-----|-------|------------------------------------------------------------------------------------------------------------------------------|
| Z        | S    | UV  | 0     | Platzhalter                                                                                                                  |
| ZAPLCNT  | S    | UV  | 4     | Zahl Aufrufe von APL in einem Screen                                                                                         |
| ZAPPLID  | S    | UV  | 8     | Applikationskennung                                                                                                          |
| ZENVIR   | S    | UV  | 32    | Umgebungsbeschreibung<br>1-8 = Produktname<br>9-16 = Betriebssystem<br>17-24 = Systemumgebung<br>25-32 = Blanks (reserviert) |
| ZISPFRC  | S    | E   | 8     | Return Code von ISPSTART                                                                                                     |
| ZLANG    | P    | UV  | 8     | Dialogsprache (installationsabhängig)                                                                                        |
| ZLOGON   | S    | UV  | 8     | Name der Logon-Prozedur                                                                                                      |
| ZPLACE   | P    | EA  | 7     | Lage der Befehlszeile (ASIS/BOTTOM)                                                                                          |
| ZPREFIX  | S    | UC  | 8     | Prefix aus dem TSO-Benutzerprofil                                                                                            |
| ZPROFAPP | P    | E   | 8     | Name des Profile Pools                                                                                                       |
| ZRXMSG   | S    | EA  | 8     | Message-ID aus Panel-REXX                                                                                                    |
| ZRXRC    | S    | EA  |       | Return Code aus Panel-REXX                                                                                                   |
| ZTEMPF   | S    | UV  | 8     | Temporäre Ausgabe File Tailoring                                                                                             |
| ZTEMPN   | S    | UV  | 8     | DD-Name von ZTEMPF                                                                                                           |
| ZUSER    | S    | UV  | 8     | TSO-Benutzerkennung                                                                                                          |
| ZVERB    | S    | A   | 8     | SetVerb-Instruktion aus CMD-Table                                                                                            |
| ZWINTTL  | F    | E   |       | Titelzeile für PopUp-Windows                                                                                                 |

## Bildschirm und PF-Tasten

| Name    | Pool | Typ | Länge | Beschreibung                          |
|---------|------|-----|-------|---------------------------------------|
| Z       | S    | UV  | 0     | Platzhalter                           |
| ZCOLOR  | S    | UV  | 4     | Anzahl möglicher Farben (1 oder 7)    |
| ZDBCS   | S    | AV  | 3     | DBCS-Fähigkeit des Terminals (YES/NO) |
| ZHILITE | S    | UV  | 3     | Hilite möglich (YES/NO)               |

|          |   |    |       |                                                                                        |
|----------|---|----|-------|----------------------------------------------------------------------------------------|
| ZKEYS    | P | A  | 4     | Anzahl PF-Tasten                                                                       |
| ZPFCTL   | P | EA | 5     | PFSHOW-Befehl möglich (YES/NO)                                                         |
| ZPFFMT   | P | EA | 4     | Anzahl PF-Tastenanzeige je Zeile                                                       |
| ZPFSET   | P | EA | 4     | Angezeigtes PFK-Tasten-Set                                                             |
| ZPFSHOW  | P | EA | 4     | Befehlsstatus PFSHOW                                                                   |
| ZPFxx    | P | EA | <=255 | PF-Tastenbelegung                                                                      |
| ZPFLxx   | P | EA | 8     | PFK-Label-Inhalt                                                                       |
| ZSCREEN  | S | UV | 2     | Logischer Bildschirm                                                                   |
| ZSCREENI | S | UV |       | Bildschirmpuffer                                                                       |
| ZSCREENC | S | UV |       | Cursorposition in ZSCREENI (Achtung, es ist ein Offset, für die Byteposition immer +1) |
| ZSCREEND | S | UV | 4     | Bildschirmtiefe (Anzahl Zeilen)                                                        |
| ZSCREENW | S | UV | 4     | Bildschirmbreite (Anzahl Spalten)                                                      |
| ZSCRMXDX | S | UV | 4     | Maximale Bildschirmtiefe im Dialog                                                     |
| ZSCRMXW  | S | UV | 4     | Maximale Bildschirmbreite im Dialog                                                    |
| ZSPLIT   | S | UV | 3     | Split Screen aktiv (YES/NO)                                                            |
| ZTERM    | P | A  | 8     | Terminaltyp wie in 0.1 definiert                                                       |

## Scrolling

| Name     | Pool | Typ | Länge | Beschreibung                                              |
|----------|------|-----|-------|-----------------------------------------------------------|
| ZSCBR    | P    | EA  | 4     | Scrollamount für Browse                                   |
| ZSCED    | P    | EA  | 4     | Scrollamount für Edit                                     |
| ZSCML    | P    | EA  | 4     | Scrollamount für Memberliste                              |
| ZSCROLLA | S    | A   | 4     | Wert aus dem Scrollamount-Feld                            |
| ZSCROLLN | S    | A   | 4     | Scrollnumber errechnet aus ZSCROLLA                       |
| ZSCROLLD | A    | E   | 4     | Standardwert für Scroll dynamischer Bereiche und Tabellen |

## Table Display Service

| Name     | Pool | Typ | Länge | Beschreibung                                                                                    |
|----------|------|-----|-------|-------------------------------------------------------------------------------------------------|
| ZTDMARK  | F    | A   | 4     | Ersatzweise zu "Bottom of Data"                                                                 |
| ZTDMSG   | A    | E   | 8     | Benutzerdefinierter TOP/DOWN-Indikator                                                          |
| ZTDROWS  | F    | A   | 6     | Ausgewählte Sätze nach TBDISPL                                                                  |
| ZTDLROWS | F    | A   | 6     | Sätze in der logischen Tabelle                                                                  |
| ZTDSELS  | F    | A   | 4     | Ausgewählte Sätze nach TBDISPL                                                                  |
| ZTDTOP   | F    | A   | 6     | CRP des obersten sichtbaren Satzes                                                              |
| ZTDLTOP  | F    | A   | 6     | Position der TOP-Zeile der physischen Tabelle in Relation zur Position in der logischen Tabelle |
| ZTDRET   | F    | E   | 8     | Bestimmt, ob Scroll-Return benutzt wird                                                         |
| ZTDADD   | F    | A   | 3     | Mehr Sätze für Scrolling nötig (YES/NO)                                                         |
| ZTDSCRIP | F    | EA  | 6     | CRP der TOP-Zeile nach Scrollvorgang                                                            |
| ZTDSRID  | F    | A   | 6     | RowID des Satzes, auf den ZTDSCRIP zeigt                                                        |
| ZTDAMT   | F    | A   | 4     | Anzahl Sätze, die für Scrolling nötig sind                                                      |

**Hilite-Informationen**

| Name     | Pool | Typ | Länge | Beschreibung                               |
|----------|------|-----|-------|--------------------------------------------|
| ZHIAUTO  | S    | A   | 3     | ON oder OFF                                |
| ZHILANG  | S    | A   | 8     | Eingestellte Sprache für HILITE            |
| ZHICOLOR | S    | A   | 8     | Was wird coloriert                         |
| ZHIPAREN | S    | A   | 3     | ON oder OFF, HILIE für Klammern            |
| ZHIFND   | S    | A   | 3     | ON oder OFF, HILITE für den FIND-Befehl    |
| ZHICURSR | S    | A   | 3     | ON oder OFF, HILITE für die Cursorposition |