

Natural für Windows

Statements

Version 6.3.8 für Windows

Februar 2010

Dieses Dokument gilt für Natural ab Version 6.3.8 für Windows.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

Copyright © 1992-2010 Software AG, Darmstadt, Deutschland und/oder Software AG USA, Inc., Reston, VA, Vereinigte Staaten von Amerika, und/oder ihre Lizenzgeber..

Der Name Software AG, webMethods und alle Software AG Produktnamen sind entweder Warenzeichen oder eingetragene Warenzeichen der Software AG und/oder der Software AG USA, Inc und/oder ihrer Lizenzgeber. Andere hier erwähnte Unternehmens- und Produktnamen können Warenzeichen ihrer jeweiligen Eigentümer sein.

Die Nutzung dieser Software unterliegt den Lizenzbedingungen der Software AG. Diese Bedingungen sind Bestandteil der Produktdokumentation und befinden sich unter <http://documentation.softwareag.com/legal/> und/oder im Wurzelverzeichnis des lizenzierten Produkts.

Diese Software kann Teile von Drittanbieterprodukten enthalten. Die Hinweise zu den Urheberrechten und Lizenzbedingungen der Drittanbieter entnehmen Sie bitte den "License Texts, Copyright Notices and Disclaimers of Third Party Products". Dieses Dokument

ist Bestandteil der Produktdokumentation und befindet sich unter <http://documentation.softwareag.com/legal/> und/oder im Wurzelverzeichnis des lizenzierten Produkts.

Inhaltsverzeichnis

1 Statements	1
2 Syntax-Symbole und Operandentabellen	3
Syntax-Symbole	4
Operandentabelle	6
3 Statements nach Funktionen	9
Datenbankzugriffe und Datenbankänderungen	10
Arithmetische Funktionen und Datenzuweisungen	12
Schleifenverarbeitung	12
Erstellen von Ausgabe-Reports	13
Bildschirmgenerierung für interaktive Verarbeitung	13
Verarbeitung logischer Bedingungen	14
Aufrufen von Programmen und Unterprogrammen	14
Beenden von Programmen und Sessions	15
Verarbeitung von Arbeitsdateien	15
Komponentenbasierte Programmierung	15
Ereignisgesteuerte Programmierung	16
Speicherverwaltung für dynamische Variablen/X-Arrays	16
Natural Remote Procedure Call	16
Internet und XML	17
Sonstige Statements	17
Reporting Mode-Statements	18
4 ACCEPT/REJECT	21
Funktion	22
Syntax-Beschreibung	23
Verarbeitung mehrerer ACCEPT/REJECT-Statements	23
Limit-Notation	24
Beispiele	24
5 ADD	27
Funktion	28
Syntax-Beschreibung	28
Beispiel	30
6 ASSIGN	33
7 AT BREAK	35
Funktion	36
Syntax-Beschreibung	37
Gruppenwechsel auf mehreren Ebenen	39
Beispiele	40
8 AT END OF DATA	45
Funktion	46
Einschränkungen	47
Syntax-Beschreibung	47
Beispiel	48
9 AT END OF PAGE	51

Funktion	52
Syntax-Beschreibung	54
Beispiele	54
10 AT START OF DATA	57
Funktion	58
Syntax-Beschreibung	59
Beispiel	59
11 AT TOP OF PAGE	63
Funktion	64
Einschränkung	65
Syntax-Beschreibung	65
Beispiel	66
12 BACKOUT TRANSACTION	69
Funktion	70
Einschränkung	71
Datenbank-spezifische Anmerkungen	71
Beispiel	71
13 BEFORE BREAK PROCESSING	73
Funktion	74
Einschränkungen	75
Syntax-Beschreibung	75
Beispiel	76
14 CALL	77
Funktion	78
Syntax-Beschreibung	78
Return Code	79
User Exits	80
INTERFACE4	80
15 CALL FILE	93
Funktion	94
Einschränkung	94
Syntax-Beschreibung	95
Beispiel	95
16 CALL LOOP	99
Funktion	100
Einschränkung	100
Syntax-Beschreibung	101
Beispiel	101
17 CALLNAT	103
Funktion	104
Syntax-Beschreibung	105
Übertragung von Parametern mit dynamischen Variablen	107
Beispiele	108
18 CLOSE CONVERSATION	111
Funktion	112

Syntax-Beschreibung	112
Weitere Informationen und Beispiele	113
19 CLOSE DIALOG	115
Funktion	116
Syntax-Beschreibung	116
Weitere Informationen und Beispiele	117
20 CLOSE PRINTER	119
Funktion	120
Syntax-Beschreibung	120
Beispiel	121
21 CLOSE WORK FILE	123
Funktion	124
Syntax-Beschreibung	124
Beispiel	124
22 COMPRESS	127
Funktion	128
Syntax-Beschreibung	128
Verarbeitung	132
Beispiele	132
23 COMPUTE	137
Funktion	138
Syntax-Beschreibung	140
Ergebnisgenauigkeit einer Division	142
SUBSTRING-Option	143
Beispiele	143
24 CREATE OBJECT	147
Funktion	148
Syntax-Beschreibung	148
25 DECIDE FOR	151
Funktion	152
Syntax-Beschreibung	152
Beispiele	153
26 DECIDE ON	155
Funktion	156
Syntax-Beschreibung	157
Beispiele	158
27 DEFINE CLASS	161
Funktion	162
Syntax-Beschreibung	163
28 DEFINE DATA	165
29 Syntax-Übersicht	167
Allgemeine Syntax	168
Basis-Syntaxelemente	168
30 DEFINE DATA - Allgemeines	173
Funktion	174

Regeln	174
Programmiermodi	174
Weitere Informationen	175
31 Definition von Local Data	177
Funktion	178
Einschränkung	178
Syntax-Beschreibung	178
32 Definition von Global Data	181
Funktion	182
Syntax-Beschreibung	182
33 Definition von Parameter Data	183
Funktion	184
Einschränkungen	184
Syntax-Beschreibung	184
34 Definition von anwendungsunabhängigen Variablen	189
Funktion	190
Syntax-Beschreibung	191
35 Definition von Kontext-Variablen für den Natural RPC	193
Funktion	194
Einschränkungen	195
Syntax-Beschreibung	195
36 Definition von NaturalX-Objekten	197
Funktion	198
Syntax-Beschreibung	198
37 Definition von Variablen	201
Funktion	202
Syntax-Beschreibung	203
38 View-Definition	205
Funktion	206
Syntax-Beschreibung	206
39 Redefinition	211
Funktion	212
Einschränkungen	212
Syntax-Beschreibung	213
40 Handle-Definition	215
Funktion	216
Syntax-Beschreibung	217
41 Definition von Array-Dimensionen	219
Funktion	220
Syntax-Beschreibung	220
42 Definition eines Ausgangswerts	225
Funktion	226
Einschränkung	226
Syntax-Beschreibung	227
43 Ausgangswerte/Konstanten-Werte für ein Array	229

Funktion	230
Einschränkung	230
Syntax-Beschreibung	231
44 Parameter EM, HD, PM für Feld/Variable	233
Funktion	234
Syntax-Beschreibung	234
45 Beispiele für die Benutzung des DEFINE DATA-Statements	235
Beispiel 1 – DEFINE DATA LOCAL (Direkte Daten-Definition)	236
Beispiel 2 – DEFINE DATA LOCAL (Array-Definition/Initialisierung)	236
Beispiel 3 – DEFINE DATA (View-Definition, Array-Redefinition)	237
Beispiel 4 – DEFINE DATA (Global, Parameter und Local Data Areas)	239
Beispiel 5 – DEFINE DATA (Initialisierung)	240
Beispiel 6 – DEFINE DATA (Variables Array mit (1:V))	240
46 DEFINE FUNCTION	243
Funktion	244
Syntax-Beschreibung	244
Beispiel	246
47 DEFINE PRINTER	247
Funktion	248
Syntax-Beschreibung	248
Beispiele	250
48 DEFINE PROTOTYPE	253
Funktion	254
Syntax-Beschreibung	255
Beispiel	257
49 DEFINE SUBROUTINE	259
Funktion	260
Einschränkungen	260
Syntax-Beschreibung	262
Welche Daten einer Subroutine zur Verfügung stehen	262
Beispiele	263
50 DEFINE WINDOW	267
Funktion	268
Syntax-Beschreibung	269
Schutz von Eingabefeldern in einem Fenster	273
Aufrufen unterschiedlicher Fenster	273
Beispiel	274
51 DEFINE WORK FILE	277
Funktion	278
Syntax-Beschreibung	278
52 DELETE	283
Funktion	284
Einschränkung	284
Syntax-Beschreibung	284
Datenbank-spezifische Anmerkungen	285

Beispiele	285
53 DISPLAY	287
Funktion	288
Syntax-Beschreibung	288
Standardwerte	301
Beispiele	302
54 DIVIDE	309
Funktion	310
Syntax-Beschreibung	310
Beispiel	314
55 DO/DOEND	315
Funktion	316
Einschränkungen	316
Beispiel	317
56 EJECT	319
Funktion	320
Syntax-Beschreibung	320
Verarbeitung	322
Beispiel	322
57 END	325
Funktion	326
Syntax-Beschreibung	326
Beispiele	327
58 END TRANSACTION	329
Funktion	330
Einschränkung	331
Syntax-Beschreibung	331
Betroffene Datenbanken	331
Datenbank-spezifische Anmerkungen	332
Beispiele	332
59 ESCAPE	335
Funktion	336
Syntax-Beschreibung	337
Beispiel	338
60 EXAMINE	341
Syntax 1 – EXAMINE	342
Syntax 2 – EXAMINE TRANSLATE	351
Syntax 3 – EXAMINE für Unicode-Grapheme	353
Beispiele	356
61 EXPAND	363
Funktion	364
Syntax-Beschreibung	364
62 FETCH	369
Funktion	370
Syntax-Beschreibung	371

Beispiel	372
63 FIND	375
Funktion	376
Einschränkungen	378
Syntax-Beschreibung	378
Beispiele	400
64 FOR	411
Funktion	412
Syntax-Beschreibung	413
Beispiel — FOR-Statement	414
65 FORMAT	415
Funktion	416
Syntax-Beschreibung	417
Parameter	417
Beispiel	418
66 GET	421
Funktion	422
Einschränkungen	423
Syntax-Beschreibung	423
Beispiel	424
67 GET SAME	427
Funktion	428
Einschränkungen	428
Syntax-Beschreibung	429
Beispiel	429
68 GET TRANSACTION DATA	431
Funktion	432
Einschränkung	433
Syntax-Beschreibung	433
Beispiel	433
69 HISTOGRAM	435
Funktion	436
Einschränkungen	437
Syntax-Beschreibung	437
Beispiele	444
70 IF	449
Funktion	450
Syntax-Beschreibung	451
Beispiel — IF-Statement	451
71 IF SELECTION	453
Funktion	454
Syntax-Beschreibung	455
Beispiel — IF SELECTION-Statement	456
72 IGNORE	457
Funktion	458

Beispiel – IGNORE-Statement	458
73 INCLUDE	459
Funktion	460
Syntax-Beschreibung	460
Beispiele	462
74 INPUT	467
Funktion	468
Eingabe-Modi	468
Eingabe von Daten als Reaktion auf ein INPUT-Statement	470
SB – Auswahlfenster (Selection Box)	472
Eingabefehler	473
Geteilter Schirm (Split Screen)	473
Systemvariablen beim INPUT-Statement	473
75 INPUT-Syntax 1 – Dynamisch generierter Eingabeschirm	475
INPUT Syntax 1 – Beschreibung	476
Beispiele – Verwendung von Syntax 1	487
76 INPUT-Syntax 2 – Verwendung einer vordefinierten Eingabemaske	491
INPUT USING MAP ohne Parameterliste	492
Im Programm definierte Eingabefelder	493
INPUT Syntax 2 – Beschreibung	493
INPUT-Statement unter Nicht-Screen-Modi	495
Eingabedaten aus dem Natural-Stack	496
77 INTERFACE	497
Funktion	498
Syntax-Beschreibung	499
78 LIMIT	505
Funktion	506
Syntax-Beschreibung	507
Beispiele	507
79 LOOP	509
Funktion	510
Einschränkung	510
Syntax-Beschreibung	511
Beispiele	511
80 METHOD	513
Funktion	514
Syntax-Beschreibung	514
Beispiel	515
81 MOVE	519
Funktion	520
Syntax-Beschreibung	521
Beispiele	534
82 MOVE ALL	539
Funktion	540
Syntax-Beschreibung	540

Beispiel	541
83 MOVE INDEXED	543
84 MULTIPLY	545
Funktion	546
Syntax-Beschreibung	546
Beispiel	548
85 NEWPAGE	551
Funktion	552
Syntax-Beschreibung	553
Beispiel	554
86 OBTAIN	557
Funktion	558
Einschränkung	558
Syntax-Beschreibung	559
Beispiele	563
87 ON ERROR	567
Funktion	568
Einschränkung	568
Syntax-Beschreibung	569
ON ERROR-Verarbeitung in Unterprogrammen	569
Systemvariablen *ERROR-NR und *ERROR-LINE	569
Beispiel	570
88 OPEN CONVERSATION	571
Funktion	572
Syntax-Beschreibung	572
Weitere Informationen und Beispiele	573
89 OPEN DIALOG	575
Funktion	576
Syntax-Beschreibung	576
Weitere Informationen und Beispiele	578
90 OPTIONS	579
Funktion	580
91 PARSE XML	581
Funktion	582
Syntax-Beschreibung	583
Beispiele	586
92 PASSW	591
Funktion	592
Syntax-Beschreibung	592
93 PERFORM	595
Funktion	596
Syntax-Beschreibung	597
Beispiele	599
94 PERFORM BREAK PROCESSING	603
Funktion	604

Syntax-Beschreibung	604
Beispiel	605
95 PRINT	607
Funktion	608
Syntax-Beschreibung	609
Beispiel	614
96 PROCESS	617
Funktion	618
Einschränkung	618
Syntax-Beschreibung	618
97 PROCESS COMMAND	621
Funktion	623
Syntax-Beschreibung	623
Das DDM COMMAND	635
Beispiele	636
98 PROCESS GUI	639
Funktion	640
Syntax-Beschreibung	640
99 PROCESS PAGE	643
Funktion	644
Syntax 1 – PROCESS PAGE	644
Syntax 2 – PROCESS PAGE USING	647
Syntax 3 – PROCESS PAGE UPDATE	650
Syntax 4 – PROCESS PAGE MODAL	653
Beispiele	655
100 PROCESS REPORTER	657
Funktion	658
Syntax-Beschreibung	659
Beispiele	663
101 PROPERTY	667
Funktion	668
Syntax-Beschreibung	668
Beispiel	669
102 READ	671
Funktion	672
Syntax-Beschreibung	673
Bei READ verfügbare Systemvariablen	682
Beispiele	683
103 READ WORK FILE	691
Funktion	692
Feldlängen	697
Verarbeitung dynamischer Variablen	697
Syntax-Beschreibung	694
Feldlängen	697
Verarbeitung dynamischer Variablen	697

Beispiel	698
104 REDEFINE	701
Funktion	702
Einschränkung	702
Syntax-Beschreibung	702
Beispiele	703
105 REDUCE	707
Funktion	708
Syntax-Beschreibung	708
106 REINPUT	713
Funktion	714
Syntax-Beschreibung	715
Beispiele	721
107 REJECT	725
108 RELEASE	727
Funktion	728
Syntax-Beschreibung	728
Beispiel	729
109 REPEAT	731
Funktion	732
Syntax-Beschreibung	732
Beispiele	733
110 REQUEST DOCUMENT	737
Funktion	738
Syntax-Beschreibung	739
Kodierung von eingehenden/ausgehenden Daten	747
Beispiele	748
111 RESET	751
Funktion	752
Syntax-Beschreibung	753
Beispiel	754
112 RESIZE	757
Funktion	758
Syntax-Beschreibung	758
113 RETRY	763
Funktion	764
Einschränkung	764
Beispiel	764
114 RUN	767
Funktion	768
Syntax-Beschreibung	768
Dynamische Sourcecode-Generierung und -Ausführung	769
Beispiel	770
115 SEND EVENT	773
Funktion	774

Syntax-Beschreibung	774
Weitere Informationen und Beispiele	776
116 SEND METHOD	777
Funktion	778
Syntax-Beschreibung	778
Beispiel	781
117 SEPARATE	789
Funktion	790
Syntax-Beschreibung	790
Beispiele	794
118 SET CONTROL	799
Funktion	800
Syntax-Beschreibung	800
Beispiele	800
119 SET GLOBALS	803
Funktion	804
Parameter	804
Beispiel	805
120 SET KEY	807
Funktion	808
Syntax-Beschreibung	808
Tasten programm-sensitiv machen und deaktivieren	809
Kommandos/Programme einer Taste zuweisen	811
Eingabedaten einer Taste zuweisen (DATA)	811
Tastenfunktion vorübergehend deaktivieren	812
Helproutine zuweisen (HELP)	813
Dynamische Funktionszuweisung (DYNAMIC)	813
GUI-Element-Zuweisung deaktivieren (DISABLED)	814
SET KEY-Statements auf verschiedenen Programmebenen	814
Namen zuweisen	816
Beispiel	817
121 SET TIME	819
Funktion	820
Beispiel	820
122 SET WINDOW	823
Funktion	824
Syntax-Beschreibung	824
Beispiel	825
123 SKIP	827
Funktion	828
Syntax-Beschreibung	828
Beispiel	829
124 SORT	831
Funktion	832
Einschränkungen	833






Syntax-Beschreibung	833
Phasen der SORT-Verarbeitung	836
Beispiel	837
125 STACK	841
Funktion	842
Syntax-Beschreibung	842
Beispiel	845
126 STOP	847
Funktion	848
Beispiel	848
127 STORE	851
Funktion	852
Datenbankspezifische Anmerkungen	853
Syntax-Beschreibung	853
Beispiel	855
128 SUBTRACT	859
Funktion	860
Syntax-Beschreibung	860
Beispiel	862
129 SUSPEND IDENTICAL SUPPRESS	863
Funktion	864
Syntax-Beschreibung	864
Beispiele	865
130 TERMINATE	869
Funktion	870
Syntax-Beschreibung	870
Kontrollübergabe nach Abbruch	871
Beispiel	871
131 UPDATE	873
Funktion	874
Einschränkungen	875
Datenbankspezifische Anmerkungen	875
Syntax-Beschreibung	875
Beispiel	877
132 WRITE	879
Funktion	880
Syntax 1 – Dynamische Formatierung	881
Syntax 1 – Beschreibung	881
Syntax 2 – Vordefinierte Form/Map benutzen	889
Syntax 2 – Beschreibung	890
Beispiele	891
133 WRITE TITLE	897
Funktion	898
Einschränkungen	899
Syntax-Beschreibung	899

Beispiel	903
134 WRITE TRAILER	905
Funktion	906
Einschränkungen	907
Syntax-Beschreibung	907
Beispiel	911
135 WRITE WORK FILE	913
Funktion	914
Syntax-Beschreibung	914
Externe Darstellung der Felder	915
Verarbeitung großer und dynamischer Variablen	916
Beispiel	917
136 SQL Statements	919
137 Common Set und Extended Set	921
138 Grundlegende Syntaxbestandteile	923
Konstanten	924
Namen	924
Parameter	927
Natural-Formate und SQL-Datentypen	931
139 Das Natural-View-Konzept	933
140 Skalar-Ausdrücke	935
scalar-expression	936
scalar-operator	936
factor	937
141 Suchbedingungen	941
search-condition	942
predicate	942
142 Select Expressions	949
selection	950
table-expression	951
143 Flexible SQL	957
Flexible SQL benutzen	958
Textvariablen in Flexible SQL angeben	959
144 CALLDBPROC - SQL	963
Funktion	964
Syntax-Beschreibung	965
Beispiel	966
145 COMMIT - SQL	969
Funktion	970
Beispiel	970
146 DELETE - SQL	971
Funktion	972
Syntax-Beschreibung	972
147 INSERT - SQL	975
Funktion	976

Syntax-Beschreibung	976
Beispiel	982
148 PROCESS SQL	983
Funktion	984
Syntax-Beschreibung	984
Entire Access-Optionen	985
Beispiele	985
149 READ RESULT SET - SQL	987
Funktion	988
Syntax-Beschreibung	988
Beispiel	989
150 ROLLBACK - SQL	991
Funktion	992
Hinweis für Nicht-Natural-Programme	992
Beispiel	992
151 SELECT - SQL	993
Funktion	994
Syntax-Beschreibung	994
Join-Abfragen	1008
SELECT – Cursor-orientierte Auswahl	1009
152 UPDATE - SQL	1015
Funktion	1016
Syntax-Beschreibung	1016
Beispiele	1019
153 Referenzierte Beispielprogramme	1021
ASSIGN	1022
AT BREAK	1023
AT END OF DATA	1025
AT END OF PAGE	1026
AT START OF DATA	1027
AT TOP OF PAGE	1028
DEFINE SUBROUTINE	1029
FIND	1030
FOR	1032
HISTOGRAM	1033
IF	1034
PERFORM BREAK PROCESSING	1035
READ	1036
REPEAT	1037
SORT	1039
STORE	1040
UPDATE	1042
Beispielprogramme für Systemvariablen	1043

1 Statements

Diese Dokumentation beschreibt die Statements, aus denen die Natural-Programmiersprache besteht. Sie ist in die folgenden Abschnitte untergliedert:

 Syntax-Symbole und Operandentabellen	Informationen zu den verwendeten Syntax-Symbolen und Operandentabellen.
 Statements nach Funktionen	Liefert eine Übersicht über die nach Funktionen eingeteilten Statements.
 Statements in alphabetischer Reihenfolge	Beschreibungen der Natural-Statements in alphabetischer Reihenfolge (außer SQL-Statements).
 Natural-SQL-Statements	Neben den „eigentlichen“ Natural-Statements bietet Natural SQL-Statements, so dass Sie in Natural-Programmen SQL direkt benutzen können.
 Referenzierte Beispielprogramme	Enthält zusätzliche Beispielprogramme, die in der Natural-Statements- und Systemvariablen-Dokumentation referenziert werden.

Informationen zur grundsätzlichen Benutzung bestimmter Statements finden Sie im *Leitfaden zur Programmierung*. Dort werden u.a. folgende Themen behandelt: *Benutzervariablen* | *X-Arrays* | *Dynamische und große Variablen/Felder* | *Dynamische und große Variablen benutzen* | *Benutzerkonstanten* | *Report-Spezifikation – (rep)-Notation* | *Text-Notation* | *Benutzerkommentare* | *Logische Bedingungen* | *Regeln für arithmetische Operationen* | *User-Defined Functions*

2 Syntax-Symbole und Operandentabellen

- Syntax-Symbole 4
- Operandentabelle 6

Dieses Kapitel behandelt folgende Themen:

Syntax-Symbole

In den Diagrammen, die die Syntax der Natural-Statements darstellen, werden folgende Symbole verwendet:

Syntax-Symbol	Erläuterung
ABCDEF	Elemente, die in Großbuchstaben dargestellt sind, sind Natural-Schlüsselwörter bzw. reservierte Wörter, die genauso eingegeben werden müssen wie angegeben.
<u>ABCDEF</u>	Ist von mehreren wahlweise verwendbaren Elementen, die in Großbuchstaben dargestellt sind, eins unterstrichen (kein Hyperlink!), handelt es sich um das jeweils gültige Standardelement. Lassen Sie das Element weg, gilt der unterstrichene Wert.
<u>ABCDEF</u>	Ist ein Teil eines Wortes in Großbuchstaben unterstrichen (kein Hyperlink!), kann der unterstrichene Teil als Abkürzung für das jeweilige Wort verwendet werden.
<i>abcdef</i>	Elemente, die in Kleinbuchstaben und kursiv dargestellt sind, sind variable Informationen, an deren Stelle Sie die gewünschten Angaben machen. Anmerkung: Anstelle von <i>statement</i> oder <i>statements</i> müssen Sie je nach Situation eines oder mehrere passende Statements angeben. Wenn Sie kein bestimmtes Statement angeben möchten, können Sie das Statement IGNORE einfügen.
[]	Elemente, die in eckigen Klammern untereinander stehen, müssen nicht unbedingt angegeben werden. Von mehreren Elementen, die in einer eckigen Klammer untereinander stehen, kann nur jeweils eines angegeben werden.
{ }	Von mehreren Elementen, die in einer geschweiften Klammer untereinander stehen, muss eines angegeben werden.
	Eines der durch diesen senkrechten Strich voneinander getrennten Elemente kann eingegeben werden.
...	Auslassungspunkte nach einem Element bedeuten, dass das Element mehrmals angegeben werden darf. Gegebenenfalls gibt eine Zahl nach den Punkten an, wie oft das Element angegeben werden darf. Ist das Element vor den Auslassungspunkten in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gelten die Auslassungspunkte für den gesamten in Klammern stehenden Ausdruck.
, ...	Ein Komma und Auslassungspunkte nach einem Element bedeuten, dass das Element mehrmals angegeben werden darf, wobei die einzelnen Angaben durch Kommas voneinander getrennt werden müssen. Gegebenenfalls gibt eine Zahl nach dem Komma und den Auslassungspunkten an, wie oft das Element angegeben werden darf.

Syntax-Symbol	Erläuterung
	Ist das Element vor dem Komma und den Auslassungspunkte ein in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gelten das Komma und die Auslassungspunkte für den gesamten in Klammern stehenden Ausdruck.
: . . .	Ein Doppelpunkt und Auslassungspunkte nach einem Element bedeuten, dass das Element mehrmals angegeben werden darf, wobei die einzelnen Angaben durch Doppelpunkte voneinander getrennt werden müssen. Gegebenenfalls gibt eine Zahl nach dem Doppelpunkt und den Auslassungspunkte an, wie oft das Element angegeben werden darf. Ist das Element vor dem Doppelpunkt und den Auslassungspunkten ein in eckige oder geschweifte Klammern eingeschlossener Ausdruck, gilt der Doppelpunkt und die Auslassungspunkte für den gesamten in Klammern stehenden Ausdruck.
Other symbols (except [] { } . . . , . . . : . . .)	Alle anderen Symbole außer den in dieser Tabelle definierten müssen genauso eingegeben werden wie angegeben. Ausnahme: Der skalare SQL-Verkettungsoperator wird durch zwei senkrechte Striche dargestellt, die genauso eingegeben werden müssen, wie sie in der Syntax-Definition erscheinen.

Beispiel:

```
WRITE [USING] { FORM } operand1 [operand2 ... ]
```

- WRITE, USING, MAP und FORM sind Natural-Schlüsselwörter, die Sie genauso eingeben müssen wie angegeben.
- operand1 und operand2 sind Variablen, an deren Stelle Sie die Namen der betreffenden Objekte eingeben.
- Die geschweiften Klammern bedeuten, dass Sie entweder FORM oder MAP angeben können, aber eins von beiden angeben müssen.
- Die eckigen Klammern bedeuten, dass USING und operand2 optionale Elemente sind, die Sie angeben können, aber nicht müssen.
- Die Auslassungspunkte bedeuten, dass Sie operand2 mehrmals angeben können.

Operandentabelle

Enthält die Syntax eines Natural-Statements einen oder mehrere Operanden, so finden Sie bestimmte Informationen zu diesen Operanden jeweils in der folgenden Tabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A G N/M E	A U N P I F B D T L C G O	ja/nein	ja/nein

Die Tabelle enthält folgende Informationen zu jedem Operanden:

Mögliche Struktur

Gibt an, welche Struktur der Operand haben darf:

C	Konstante.	
S	Einzelne Ausprägung (Skalar, ein Feld bzw. eine Variable mit nur einer Ausprägung; d.h. weder ein Array noch eine Gruppe).	
A	Array.	
G	Gruppe.	
N/M	Natural-Systemvariable:	
	N	Es dürfen alle Systemvariablen verwendet werden.
	M	Nur modifizierbare Systemvariablen dürfen verwendet werden. Ob eine Systemvariable <i>modifizierbar</i> ist, steht in der <i>Systemvariablen</i> -Dokumentation.
E	Arithmetische Ausdrücke.	

Mögliche Formate

Gibt an, welche Formate der Operand haben darf:

A	Alphanumerisch (ASCII-Codepage)
U	Alphanumerisch (Unicode)
N	Numerisch ungepackt
P	Gepackt numerisch
I	Integer (= ganzzahlig)
F	Floating point (= Gleitkomma)
B	Binär

D	Datum
T	Time (= Zeit)
L	Logisch
C	Attributkontrolle
G	GUI-Handle (HANDLE OF GUI)
O	Objekt-Handle (HANDLE OF OBJECT)

Referenzierung erlaubt

Gibt an, ob der Operand über ein Statement-Label bzw. die Sourcecode-Zeilenummer referenziert werden darf.

Dynam. Definition

Gibt an, ob der Operand dynamisch im Programm definiert werden darf. Dies ist nur im *Reporting Mode* möglich.

3 Statements nach Funktionen

- Datenbankzugriffe und Datenbankänderungen 10
- Arithmetische Funktionen und Datenzuweisungen 12
- Schleifenverarbeitung 12
- Erstellen von Ausgabe-Reports 13
- Bildschirmgenerierung für interaktive Verarbeitung 13
- Verarbeitung logischer Bedingungen 14
- Aufrufen von Programmen und Unterprogrammen 14
- Beenden von Programmen und Sessions 15
- Verarbeitung von Arbeitsdateien 15
- Komponentenbasierte Programmierung 15
- Ereignisgesteuerte Programmierung 16
- Speicherverwaltung für dynamische Variablen/X-Arrays 16
- Natural Remote Procedure Call 16
- Internet und XML 17
- Sonstige Statements 17
- Reporting Mode-Statements 18

Dieses Kapitel liefert eine Übersicht über die nach Funktionen eingeteilten Statements:



Anmerkungen:

1. Manche Statements können sowohl im Structured Mode als auch im Reporting Mode verwendet werden, während andere nur im Reporting Mode verwendet werden können. Siehe auch *Natural-Programmiermodi* im *Leitfaden zur Programmierung*
2. Die Statements `DLOGOFF`, `DLOGON`, `SHOW`, `IMPORT` und `EXPORT` sind nur verfügbar, wenn Entire DB installiert ist. Eine Beschreibung dieser Statements finden Sie in der *Entire DB*-Dokumentation.

Datenbankzugriffe und Datenbankänderungen

Natural-DML-Statements

Die folgenden Natural Data Manipulation Language-Statements (DML) dienen zum Zugriff auf und zum Ändern von in einer Datenbank gespeicherten Daten:

<code>READ</code>	Lesen einer Datei in physischer oder logischer Reihenfolge der Datensätze.
<code>FIND</code>	Auswählen von Datensätzen aufgrund bestimmter Kriterien.
<code>HISTOGRAM</code>	Lesen von Werten eines Datenbankfeldes.
<code>GET</code>	Lesen eines Datensatzes mit einer bestimmten ISN (Internal Sequence Number) bzw. SNR (Record Number).
<code>GET SAME</code>	Erneutes Lesen des gerade verarbeiteten Datensatzes.
<code>ACCEPT/REJECT</code>	Annehmen/Ablehnen von Datensätzen aufgrund bestimmter Kriterien.
<code>PASSW</code>	Angabe eines Passworts zur Zugriffsberechtigung auf eine passwortgeschützte Datei.
<code>LIMIT</code>	Begrenzen der Anzahl der Ausführungen einer <code>READ</code> -, <code>FIND</code> - oder <code>HISTOGRAM</code> -Schleife.
<code>STORE</code>	Anlegen eines neuen Datensatzes in der Datenbank.
<code>UPDATE</code>	Ändern eines Datensatzes in der Datenbank.
<code>DELETE</code>	Löschen eines Datensatzes von der Datenbank.
<code>END TRANSACTION</code>	Festlegen des Endes einer logischen Transaktion.
<code>BACKOUT TRANSACTION</code>	Abbrechen einer nicht vollständig abgeschlossenen logischen Transaktion.
<code>GET TRANSACTION DATA</code>	Lesen von Transaktionsdaten, die mit einem vorhergegangenen <code>END TRANSACTION</code> -Statement gespeichert wurden.
<code>RETRY</code>	Erneuter Versuch, einen Datensatz zu lesen, der vorher von einem anderen Benutzer benutzt wurde.
<code>AT START OF DATA</code>	Ausführen von Statements, wenn in einer Schleife der erste Datensatz verarbeitet wird.

AT END OF DATA	Ausführen von Statements, nachdem in einer Schleife der letzte Datensatz verarbeitet wurde.
AT BREAK	Ausführen von Statements bei einem Wertwechsel in einem bestimmten Feld (Gruppenwechsel).
BEFORE BREAK PROCESSING	Ausführen von Statements vor einer Gruppenwechsel-Verarbeitung.
PERFORM BREAK PROCESSING	Sofortiges Ausführen einer Gruppenwechsel-Verarbeitung.

Natural-SQL-Statements

Zusätzlich zu den Natural-DML-Statements bietet Natural auch SQL-Statements zur Benutzung in Natural-Programmen, so dass SQL unmittelbar verwendet werden kann.

Folgende SQL-Statements sind verfügbar:

CALLDBPROC	Dient dazu, eine „Stored Procedure“ des SQL-Datenbanksystems, mit dem Natural verbunden ist, aufzurufen.
COMMIT	Entspricht dem END TRANSACTION-Statement. Es markiert das Ende einer logischen Transaktion und bewirkt, dass alle während der Transaktion gesperrten Daten freigegeben werden. Alle Datenänderungen werden gespeichert und sind damit definitiv.
DELETE	Löscht Reihen aus einer Tabelle, ohne einen Cursor zu verwenden („ searched “ DELETE), oder löscht Reihen aus einer Tabelle, auf die der Cursor zeigt („ positioned “ DELETE).
INSERT	Fügt einer Tabelle eine oder mehrere neue Reihen hinzu.
PROCESS SQL	Dient dazu, mit SQL-Statements auf eine Datenbank zuzugreifen.
READ RESULT SET	Liest einen Result Set, der von einer mit einem vorhergehenden CALLDBPROC-Statement aufgerufenen „Stored Procedure“ erzeugt wurde.
ROLLBACK	Entspricht dem Statement BACKOUT TRANSACTION. Es macht alle seit dem Beginn der letzten „Recovery Unit“ ausgeführten Datenbankänderungen rückgängig.
SELECT	Gemäß der Standard-SQL-Funktionalität unterstützt Natural sowohl das cursor-orientierte SELECT, mit dem eine beliebige Anzahl von Reihen gelesen werden kann, als auch das nicht cursor-orientierte „ singleton “ SELECT, das maximal eine Reihe liest.
UPDATE	Führt UPDATE-Operationen auf Reihen in einer Tabelle aus, ohne einen Cursor zu verwenden („ searched “ UPDATE), oder auf Spalten in einer Reihe, in der ein Cursor positioniert ist („ positioned “ UPDATE).

Arithmetische Funktionen und Datenzuweisungen

Die folgenden Statements werden verwendet, um arithmetische Operationen sowie Datenzuweisungen durchzuführen:

COMPUTE	Rechenoperationen ausführen oder Feldern Werte zuweisen.
ADD	Addieren von Operanden.
SUBTRACT	Subtrahieren von Operanden.
MULTIPLY	Multiplizieren von Operanden.
DIVIDE	Dividieren eines Operanden durch einen anderen.
EXAMINE TRANSLATE	Konvertiert die in einem Feld enthaltenen Zeichen in Groß- oder Kleinbuchstaben oder in andere Zeichen.
MOVE	Übertragen eines Operandenwertes in ein Feld oder mehrere Felder.
MOVE ALL	Übertragen sämtlicher Werte einer bestimmten Größe in ein anderes Feld.
COMPRESS	Aneinanderreihen mehrerer Feldwerte in einem Feld.
SEPARATE	Aufteilen eines Feldwertes in zwei oder mehr Felder.
EXAMINE	Absuchen eines Feldes nach einem bestimmten Wert und anschließend Ersetzen des Wertes und/oder Zählen, wie oft der Wert vorkommt.
RESET	Zurücksetzen eines Feldwertes auf Null (numerisches Feld) bzw. Leerwert (alphanumerisches Feld) oder auf einen Ausgangswert.

Schleifenverarbeitung

Die folgenden Statements werden in Verbindung mit der Ausführung von Verarbeitungsschleifen verwendet:

ESCAPE	Ausführung einer Verarbeitungsschleife abbrechen.
FOR	Initiieren einer Verarbeitungsschleife und Steuerung der Anzahl der Schleifendurchläufe.
REPEAT	Initiieren einer Verarbeitungsschleife (und Beenden in Abhängigkeit von einer bestimmten Bedingung).
SORT	Sortieren von Datensätzen.

Erstellen von Ausgabe-Reports

Die folgenden Statements werden bei der Erzeugung von Ausgabe-Reports verwendet:

FORMAT	Spezifizieren von Ausgabe-Parametern.
DISPLAY	Ausgabe von Feldwerten in Spalten untereinander.
WRITE / PRINT	Ausgabe von Feldwerten ohne Spalteneinteilung.
WRITE TITLE	Überschreiben einer Standard-Seitenüberschrift mit einer eigenen Seitenüberschrift.
WRITE TRAILER	Ausgabe eines Fußzeilentextes, der auf jeder Ausgabeseite erscheinen soll.
AT TOP OF PAGE	Spezifizieren der Verarbeitung, die beim Beginn einer neuen Ausgabeseite ausgeführt werden soll.
AT END OF PAGE	Spezifizieren der Verarbeitung, die beim Erreichen des Endes einer Ausgabeseite ausgeführt werden soll.
SKIP	Generieren von Leerzeilen in der Ausgabe.
EJECT	Seitenvorschub ohne Titel und Überschriften.
NEWPAGE	Seitenvorschub mit Titel und Überschriften.
SUSPEND IDENTICAL SUPPRESS	Aussetzen der „Identical Suppress“-Bedingung für einen einzelnen Datensatz.
DEFINE PRINTER	Bestimmen des Druckers oder logischen Zielorts, an dem ein Report ausgegeben werden soll.
CLOSE PRINTER	Schließen eines Druckers.

Bildschirmgenerierung für interaktive Verarbeitung

Die folgenden Statements werden in Verbindung mit der Verwendung von Bildschirmmasken (Maps) bei interaktiver Datenverarbeitung benutzt:

INPUT	Erstellen einer formatierten Map zur Datenausgabe/-eingabe.
REINPUT	Erneutes Ausführen eines INPUT-Statements (falls die auf das INPUT-Statement erfolgte Dateneingabe fehlerhaft war).
DEFINE WINDOW	Größe, Position und Attribute eines Windows festlegen.
SET WINDOW	Aktivieren und Deaktivieren eines Windows.
PROCESS PAGE	Erstellen eines Daten-Mappings auf einen Web Rich GUI-Schirm.
PROCESS PAGE USING	Ausführen einer GUI I/O-Verarbeitung unter Verwendung eines aus einem Seiten-Layout erzeugten Objekts vom Typ Adapter.

PROCESS PAGE UPDATE	Erneutes Ausführen eines PROCESS PAGE-Statements.
PROCESS PAGE MODAL	Initiieren eines Verarbeitungsblocks und Kontrolle des Lebenszyklus eines Rich GUI Window.

Verarbeitung logischer Bedingungen

Mit den folgenden Statements wird die Ausführung von Statements in Abhängigkeit von Bedingungen gesteuert, die während der Ausführung eines Natural-Programms auftreten:

IF	Ausführen von Statements aufgrund einer logischen Bedingung.
IF SELECTION	Prüfen, ob in einer Reihe von alphanumerischen Feldern genau ein Wert enthält.
DECIDE FOR	Ausführen von Statements aufgrund von logischen Bedingungen.
DECIDE ON	Ausführen von Statements aufgrund des Inhaltes einer Variablen.

Aufrufen von Programmen und Unterprogrammen

Die folgenden Statements werden zum Aufrufen von Programmen und Unterprogrammen verwendet:

CALL	Aufrufen eines Nicht-Natural-Programms von einem Natural-Programm aus.
CALLNAT	Aufrufen eines Natural-Subprogramms.
CALL FILE	Aufrufen eines Nicht-Natural-Programms, um einen Datensatz von einer Nicht-Adabas-Datei zu lesen.
CALL LOOP	Generieren einer Verarbeitungsschleife, die den Aufruf eines Nicht-Natural-Programms beinhaltet.
DEFINE SUBROUTINE	Definieren einer Natural-Subroutine.
ESCAPE	Abbrechen eines Unterprogramms.
FETCH	Aufrufen eines Natural-Programms.
PERFORM	Aufrufen einer Natural-Subroutine.
PROCESS COMMAND	Aufrufen eines Kommando-Prozessors.
RUN	Kompilieren und Ausführen eines Source-Programms.
Function Call	Aufrufen eines Natural-Objekts vom Typ Function.

Beenden von Programmen und Sessions

Die folgenden Natural-Statements dienen zum Beenden der Ausführung einer Anwendung oder der Natural-Session.

STOP	Beendet die Ausführung einer Anwendung.
TERMINATE	Beendet die Natural-Session.

Verarbeitung von Arbeitsdateien

Die folgenden Natural-Statements werden verwendet, um Daten auf eine physisch-sequentielle (nicht-Adabas) Arbeitsdatei zu schreiben bzw. von dieser zu lesen:

WRITE WORK FILE	Schreibt Daten auf eine Arbeitsdatei.
READ WORK FILE	Liest Daten von einer Arbeitsdatei.
CLOSE WORK FILE	Schließt eine Arbeitsdatei.
DEFINE WORK FILE	Weist einer Arbeitsdatei einen Dateinamen zu.

Komponentenbasierte Programmierung

Folgende Natural-Statements werden zur komponentenbasierten Programmierung verwendet:

DEFINE CLASS	Gibt innerhalb eines Natural-Klassenmoduls eine Klasse an.
CREATE OBJECT	Erstellt ein Objekt (auch bekannt als "Instanz") einer gegebenen Klasse.
SEND METHOD	Ruft eine Methode eines Objekts auf.
INTERFACE	Definiert eine Schnittstelle (eine Sammlung von Methoden und Eigenschaften) für eine bestimmte Funktion einer Klasse.
METHOD	Weist außerhalb einer Schnittstellendefinition ein Unterprogramm als Implementierung einer Methode zu.
PROPERTY	Weist außerhalb einer Schnittstellendefinition eine Objektdaten-Variable als Implementierung einer Eigenschaft zu.

Ereignisgesteuerte Programmierung

Die Natural-Statements werden zur ereignisgesteuerten Programmierung verwendet:

OPEN DIALOG	Öffnen eines Dialogs.
CLOSE DIALOG	Schließen eines Dialogs.
SEND EVENT	Auslösen eines benutzerdefinierten Ereignisses.
PROCESS GUI	Ausführen einer Standardprozedur in einer ereignisgesteuerten Anwendung.

Speicherverwaltung für dynamische Variablen/X-Arrays

EXPAND	Erweitert den zugewiesenen Speicher für dynamische Variablen auf eine gegebene Größe bzw. erweitert die Anzahl der Ausprägungen eines X-arrays.
REDUCE	Reduziert die Größe einer dynamischen Variablen bzw. die Anzahl der Ausprägungen eines X-arrays.
RESIZE	Passt die Größe einer dynamischen Variablen bzw. die Anzahl der Ausprägungen eines X-arrays an.

Natural Remote Procedure Call

OPEN CONVERSATION	Ermöglicht dem Client, eine Konversation zu öffnen und die Remote-Subprogramme anzugeben, die an der Konversation beteiligt sein sollen.
CLOSE CONVERSATION	Ermöglicht dem Client, eine Konversation zu schließen. Sie können die aktuelle Konversation, eine andere offene Konversation oder alle offenen Konversationen schließen.
DEFINE DATA CONTEXT	Dient zur Definition von als Kontext-Variablen bekannte Variablen, die für mehrere Subprogramme auf einem externen (Remote-)Rechner innerhalb einer Konversation zur Verfügung stehen sollen, ohne dass Sie die Variablen explizit als Parameter mit den entsprechenden CALLNAT-Statements übergeben müssen.

Internet und XML

PARSE	Gestattet es, XML-Dokumente von einem Natural-Programm aus zu analysieren und die darin enthaltenen Informationen der weiteren Verarbeitung zur Verfügung zu stellen.
REQUEST DOCUMENT	Ermöglicht den Zugang zu einem externen System.

Sonstige Statements

DEFINE DATA	Definiert die Datenelemente, die in einem Natural-Programm oder -Unterprogramm verwendet werden sollen.
END	Zeigt das Ende des Sourcecodes eines Natural-Programms bzw. -Unterprogramms an.
INCLUDE	Einfügen von Natural-Copycode während der Kompilierung.
ON ERROR	Abfangen von Laufzeitfehlern, die normalerweise eine Fehlermeldung und den Abbruch des ausgeführten Programms bewirken würden.
PROCESS REPORTER	Kommunikation mit dem Natural Reporter aus einem Programm heraus, um diesen anzuweisen, eine bestimmte Aktion auszuführen.
RELEASE	Löschen aller im Natural-Stack gehaltenen Daten; Freigabe aller Daten, die über eine RETAIN-Klausel in einem FIND-Statement gehalten wurden; Zurücksetzen von globalen Variablen auf die ursprünglichen Werte.
SET CONTROL	Ausführen eines Natural-Terminalkommandos aus einem Natural-Programm heraus.
SET KEY	Zuweisen von Funktionen zu Funktionstasten.
SET TIME	Setzen eines zeitlichen Bezugspunkts für eine *TIMD-Systemvariable.
STACK	Zwischenlagern von Daten bzw. Kommandos im Natural-Stack.

Reporting Mode-Statements

Die folgenden Statements gelten nur für den Reporting Mode:

CLOSE LOOP	Schließt eine Verarbeitungsschleife.
DO/DOEND	Spezifikation einer Gruppe von Statements, die auf der Grundlage einer logischen Bedingung ausgeführt werden sollen.
OBTAIN	Bewirkt, dass ein Feld oder mehrere Felder von einer Datei gelesen werden.
REDEFINE	Redefiniert ein Feld.

Die folgenden Statements können sowohl im Structured Mode als auch im Reporting Mode benutzt werden, allerdings ist die Statement-Struktur und bei einigen Statements auch die Funktionalität anders:

AT START OF DATA	Gibt Statements an, die ausgeführt werden sollen, wenn der erste in einer Reihe von Datensätzen in einer Verarbeitungsschleife verarbeitet wird.
AT END OF DATA	Gibt Statements an, die ausgeführt werden sollen, nachdem der letzte einer Reihe von Datensätzen in einer Verarbeitungsschleife verarbeitet wurde.
AT BREAK	Gibt Statements an, die ausgeführt werden sollen, wenn sich der Wert eines Kontrollfeldes ändert (Gruppenwechselverarbeitung).
AT TOP OF PAGE	Gibt eine Verarbeitung an, die ausgeführt werden soll, wenn eine neue Ausgabeseite gestartet wird.
AT END OF PAGE	Gibt eine Verarbeitung an, die ausgeführt werden soll, wenn das Ende einer Ausgabeseite erreicht ist.
BEFORE BREAK PROCESSING	Gibt Statements an, die vor Durchführung des Gruppenwechsels ausgeführt werden sollen.
CALL LOOP	Erzeugt eine Verarbeitungsschleife, die einen Aufruf an ein Nicht-Natural-Programm enthält.
CALL FILE	Ruft ein Nicht-Natural-Programm auf, um einen Datensatz aus einer Nicht-Adabas-Datei zu lesen.
COMPUTE	Führt arithmetische Operationen durch oder weist Feldern Werte zu.
DEFINE SUBROUTINE	Definiert eine Natural-Subroutine.
ESCAPE	Hält die Ausführung einer Verarbeitungsschleife an.
FIND	Wählt nach Benutzerkriterien Datensätze aus einer Datenbank- Datei aus.
GET SAME	Liest den gerade verarbeiteten Datensatz wieder ein.
HISTOGRAM	Liest die Werte eines Datenbankfeldes.
IF	Führt Statements in Abhängigkeit von einer logischen Bedingung aus.
IF SELECTION	Prüft, ob in einer Reihe von alphanumerischen Feldern wirklich nur eines einen Wert enthält.

ON ERROR	Fängt Laufzeitfehler ab, die ansonsten zu einer Natural-Fehlermeldung führen würden und beendet dann das Natural-Programm.
READ	Liest eine Datenbank-Datei in physischer oder logischer Reihenfolge der Datensätze.
READ WORK FILE	Liest Daten aus einer Arbeitsdatei.
REPEAT	Initiiert eine Verarbeitungsschleife (und beendet sie abhängig von einer angegebenen Bedingung).
SORT	Sortiert Datensätze.
STORE	Fügt der Datenbank einen neuen Datensatz hinzu.
UPDATE	Aktualisiert einen Datensatz in der Datenbank.

4 ACCEPT/REJECT

▪ Funktion	22
▪ Syntax-Beschreibung	23
▪ Verarbeitung mehrerer ACCEPT/REJECT-Statements	23
▪ Limit-Notation	24
▪ Beispiele	24

$\left\{ \begin{array}{l} \text{ACCEPT} \\ \text{REJECT} \end{array} \right\} [\text{IF}] \textit{logical-condition}$

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | HISTOGRAM | GET | GET SAME | GET TRANSACTION DATA | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Mit den Statements ACCEPT und REJECT können Sie eine logische Bedingung (*logical-condition*) angeben, aufgrund welcher ein gelesener Datensatz akzeptiert (ACCEPT) oder zurückgewiesen (REJECT) werden soll.

Beide Statements können in Verbindung mit Statements eingesetzt werden, die Datensätze in einer Verarbeitungsschleife lesen (FIND, READ, HISTOGRAM, CALL FILE, SORT oder READ WORK FILE). Die logische Bedingung wird erst ausgewertet, nachdem ein Datensatz ausgewählt/gelesen worden ist.

Wenn ein ACCEPT- bzw. REJECT-Statement ausgeführt wird, bezieht es sich auf die innerste gerade aktive Verarbeitungsschleife, die mit einem der oben genannten Statements initiiert wurde.

Befindet sich ein ACCEPT- bzw. REJECT-Statement in einer Subroutine und wird aufgrund der logischen Bedingung ein Datensatz zurückgewiesen, so wird die Subroutine automatisch beendet und die Verarbeitung mit dem nächsten Datensatz der innersten gerade aktiven Verarbeitungsschleife fortgesetzt.

Syntax-Beschreibung

IF	Eine IF-Klausel kann mit einem ACCEPT- oder REJECT-Statement verwendet werden, um logische Bedingungen über die Bedingungen hinaus anzugeben, die spezifiziert wurden, als der Satz mit einem FIND-, READ- oder HISTOGRAM-Statement ausgewählt/gelesen wurde. Die logischen Bedingungen werden ausgewertet, nachdem der Satz gelesen und seine Verarbeitung gestartet wurde.
<i>logical-condition</i>	<p>Das Basis-Kriterium ist ein relationaler Ausdruck. Mehrere relationale Ausdrücke können mit logischen Operatoren zu komplexen Kriterien kombiniert werden (AND, OR).</p> <p>Arithmetische Ausdrücke können auch zur Bildung eines relationalen Ausdrucks benutzt werden.</p> <p>Felder können Datenbankfelder oder Benutzervariablen sein. Weitere Informationen zu logischen Bedingungen, siehe <i>Logische Bedingungen</i> im Leitfaden zur Programmierung.</p> <p>Wenn ACCEPT/REJECT mit einem HISTOGRAM-Statement benutzt wird, kann nur das im HISTOGRAM-Statement angegebene Datenbankfeld als logische Bedingung verwendet werden.</p>

Verarbeitung mehrerer ACCEPT/REJECT-Statements

Pro Verarbeitungsschleife genügt in der Regel ein ACCEPT- bzw. REJECT-Statement. Wollen Sie in einer Verarbeitungsschleife mehrere ACCEPT/REJECT-Statements unmittelbar hintereinander verwenden, so beachten Sie bitte folgende Regeln:

- Befinden sich innerhalb einer Verarbeitungsschleife mehrere ACCEPT/REJECT-Statements direkt hintereinander, so werden sie in der angegebenen Reihenfolge verarbeitet.
- Wird aufgrund einer erfüllten ACCEPT-Bedingung ein Datensatz akzeptiert, so werden die unmittelbar nachfolgenden ACCEPT/REJECT-Statements ignoriert.
- Wird aufgrund einer erfüllten REJECT-Bedingung ein Datensatz zurückgewiesen, so werden die unmittelbar nachfolgenden ACCEPT/REJECT-Statements ignoriert.
- Geht die Verarbeitung bis zum letzten ACCEPT/REJECT-Statement, so entscheidet dieses letzte Statement, ob der betreffende Datensatz akzeptiert wird oder nicht.

Befindet sich zwischen zwei ACCEPT/REJECT-Statements ein anderes Statement, so werden beide ACCEPT/REJECT-Statements unabhängig voneinander verarbeitet.

Limit-Notation

Ist die Anzahl der Durchläufe einer Verarbeitungsschleife durch ein `LIMIT`-Statement oder eine andere Einschränkung begrenzt, so gilt diese für die Anzahl der gelesenen Datensätze, und zwar unabhängig davon, wieviele der gelesenen Datensätze aufgrund eines `ACCEPT`- oder `REJECT`-Statements akzeptiert oder zurückgewiesen werden.

Beispiele

- [Beispiel 1 — ACCEPT](#)
- [Beispiel 2 — ACCEPT / REJECT](#)

Beispiel 1 — ACCEPT

```
** Example 'ACREX1': ACCEPT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SEX
  2 MAR-STAT
END-DEFINE
*
LIMIT 50
READ EMPLOY-VIEW
  ACCEPT IF SEX='M' AND MAR-STAT = 'S'
  WRITE NOTITLE '=' NAME '=' SEX 5X '=' MAR-STAT
END-READ
END
```

Ausgabe des Programms ACREX1:

```
NAME: MORENO           S E X: M           MARITAL STATUS: S
NAME: VAUZELLE        S E X: M           MARITAL STATUS: S
NAME: BAILLET         S E X: M           MARITAL STATUS: S
NAME: HEURTEBISE     S E X: M           MARITAL STATUS: S
NAME: LION            S E X: M           MARITAL STATUS: S
NAME: DEZELUS        S E X: M           MARITAL STATUS: S
NAME: BOYER          S E X: M           MARITAL STATUS: S
NAME: BROUSSE        S E X: M           MARITAL STATUS: S
NAME: DROMARD        S E X: M           MARITAL STATUS: S
NAME: DUC            S E X: M           MARITAL STATUS: S
NAME: BEGUERIE       S E X: M           MARITAL STATUS: S
```

```

NAME: FOREST           S E X: M     MARITAL STATUS: S
NAME: GEORGES          S E X: M     MARITAL STATUS: S

```

Beispiel 2 — ACCEPT / REJECT

```

** Example 'ACREX2': ACCEPT/REJECT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 SALARY      (1)
*
1 #PROC-COUNT (N8) INIT <0>
END-DEFINE
*
EMP. FIND EMPLOY-VIEW WITH NAME = 'JACKSON'
  WRITE NOTITLE *COUNTER NAME FIRST-NAME 'SALARY:' SALARY(1)
  /*
  ACCEPT IF SALARY (1) LT 50000
  WRITE *COUNTER 'ACCEPTED FOR FURTHER PROCESSING'
  /*
  REJECT IF SALARY (1) GT 30000
  WRITE *COUNTER 'NOT REJECTED'
  /*
  ADD 1 TO #PROC-COUNT
END-FIND
*
SKIP 2
WRITE NOTITLE 'TOTAL PERSONS FOUND ' *NUMBER (EMP.) /
              'TOTAL PERSONS SELECTED' #PROC-COUNT
END

```

Ausgabe des Programms ACREX2:

```

      1 JACKSON           CLAUDE           SALARY:      33000
      1 ACCEPTED FOR FURTHER PROCESSING
      2 JACKSON           FORTUNA           SALARY:      36000
      2 ACCEPTED FOR FURTHER PROCESSING
      3 JACKSON           CHARLIE           SALARY:      23000
      3 ACCEPTED FOR FURTHER PROCESSING
      3 NOT REJECTED

TOTAL PERSONS FOUND           3
TOTAL PERSONS SELECTED       1

```


5 ADD

▪ Funktion	28
▪ Syntax-Beschreibung	28
▪ Beispiel	30

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das ADD-Statement wird benutzt, um zwei oder mehr Operanden zu addieren.



Anmerkungen:

1. Zu dem Zeitpunkt, zu dem das ADD-Statement ausgeführt wird, muss jeder bei der arithmetischen Operation benutzte Operand einen gültigen Wert enthalten.
2. Zu Additionen mit Arrays, siehe auch den Abschnitt *Arithmetische Operationen mit Arrays* im *Leitfaden zur Programmierung*.
3. Zum Format der Operanden, siehe auch den Abschnitt *Formatwahl im Hinblick auf die Verarbeitungszeit* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung

Mit dem ADD-Statement können Sie zwei oder mehrere Operanden addieren.

- [Syntax 1](#)
- [Syntax 2](#)

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Syntax 1

```
ADD [ROUNDED] operand1... TO operand2
```


Operanden-Definitionstabelle (Syntax 1):

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A N	N P I F D T	ja	nein
<i>operand2</i>	S A M	N P I F D T	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<i>operand1</i> ist der Addend (zweiter Summand).
ROUNDED	Wünschen Sie das Ergebnis gerundet, geben Sie das Schlüsselwort ROUNDED an. Die für das Runden gültigen Regeln finden Sie unter <i>Regeln für arithmetische Operationen im Leitfaden zur Programmierung</i> .
TO <i>operand2</i>	<i>operand2</i> wird in die Addition einbezogen und enthält anschließend das Ergebnis der Operation.

Beispiel:

Das Statement

ADD #A(*) TO #B(*)	ist gleichbedeutend mit	COMPUTE #B(*) := #A(*) + #B(*)
ADD #S TO #R	ist gleichbedeutend mit	COMPUTE #R := #S + #R
ADD #S #T TO #R	ist gleichbedeutend mit	COMPUTE #R := #S + #T + #R
ADD #A(*) TO #R	ist gleichbedeutend mit	COMPUTE #R := #A(*) + #R

Syntax 2

ADD [ROUNDED] *operand1*... GIVING *operand2*

Operanden-Definitionstabelle (Syntax 2):

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A N	N P I F D T	ja	nein
<i>operand2</i>	S A M	A U N P I F B* D T	ja	ja

* Format B von *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

<i>operand1</i>	<i>operand1</i> ist der Addend (zweiter Summand).
ROUNDED	Wünschen Sie das Ergebnis gerundet, geben Sie das Schlüsselwort ROUNDED an. Die für das Runden gültigen Regeln finden Sie unter <i>Regeln für arithmetische Operationen</i> im <i>Leitfaden zur Programmierung</i> .
GIVING <i>operand2</i>	<i>operand2</i> erhält nur das Ergebnis der Operation und wird nicht in die Addition einbezogen. Wird <i>operand2</i> mit alphanumerischem Format definiert, dann wird das Ergebnis in alphanumerisches Format umgewandelt.



Anmerkung: Bei Verwendung von Syntax 2 gilt Folgendes: Das Feld bzw. die Felder (*operand1*) links vom Schlüsselwort **GIVING** sind die Terme der Addition, das Feld rechts von **GIVING** (*operand2*) wird nur zum Empfang des Ergebnisses genutzt. Wird nur ein einzelnes Feld (*operand1*) geliefert, dann wird aus der **ADD**-Operation in eine Zuweisung.

Beispiel:

Das Statement

```

ADD #S      GIVING #R  ist gleichbedeutend mit  COMPUTE #R := #S
ADD #S #T   GIVING #R  ist gleichbedeutend mit  COMPUTE #R := #S + #T
ADD #A(*) 0  GIVING #R  ist gleichbedeutend mit  COMPUTE #R := #A(*) + 0
  Dies ist eine zulässige Operation aufgrund der Regeln im Abschnitt Arithmetische Operationen mit Arrays
ADD #A(*)   GIVING #R  ist gleichbedeutend mit  COMPUTE #R := #A(*)
  Dies ist eine unzulässige Operation aufgrund der Regeln im Abschnitt Zuweisungen bei Arrays
    
```

Beispiel

```

** Example 'ADDEX1': ADD
*****
DEFINE DATA LOCAL
1 #A      (P2)
1 #B      (P1.1)
1 #C      (P1)
1 #DATE   (D)
1 #ARRAY1 (P5/1:4,1:4) INIT (2,*) <5>
1 #ARRAY2 (P5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
*
ADD +5 -2 -1 GIVING #A
    
```

```

WRITE NOTITLE 'ADD +5 -2 -1 GIVING #A' 15X '=' #A
*
ADD .231 3.6 GIVING #B
WRITE          / 'ADD .231 3.6 GIVING #B' 15X '=' #B
*
ADD ROUNDED 2.9 3.8 GIVING #C
WRITE          / 'ADD ROUNDED 2.9 3.8 GIVING #C' 8X '=' #C
*
MOVE *DATX TO #DATE
ADD 7 TO #DATE
WRITE          / 'CURRENT DATE:'          *DATX (DF=L) 13X
              / 'CURRENT DATE + 7:' #DATE (DF=L)
*
WRITE          / '#ARRAY1 AND #ARRAY2 BEFORE ADDITION'
              / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
WRITE          / '#ARRAY1 AND #ARRAY2 AFTER ADDITION'
              / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
*
END

```

Ausgabe des Programms ADDEX1:

```

ADD +5 -2 -1 GIVING #A           #A:   2
ADD .231 3.6 GIVING #B         #B:   3.8
ADD ROUNDED 2.9 3.8 GIVING #C   #C:   7
CURRENT DATE: 2005-01-10        CURRENT DATE + 7: 2005-01-17

#ARRAY1 AND #ARRAY2 BEFORE ADDITION
#ARRAY1:    5    5    5    5 #ARRAY2:    10    10    10    10

#ARRAY1 AND #ARRAY2 AFTER ADDITION
#ARRAY1:    5    5    5    5 #ARRAY2:    15    15    15    15

```


6 ASSIGN

Siehe Statement [COMPUTE](#).

7 AT BREAK

▪ Funktion	36
▪ Syntax-Beschreibung	37
▪ Gruppenwechsel auf mehreren Ebenen	39
▪ Beispiele	40

Structured Mode-Syntax

```
[AT] BREAK [(r)] [OF] operand1 [/n/]  
    statement ...  
END-BREAK
```

Reporting Mode-Syntax

```
[AT] BREAK [(r)] [OF] operand1 [/n/]  
    { statement  
      DO statement... DOEND }  
}
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [ACCEPT/REJECT](#) | [AT START OF DATA](#) | [AT END OF DATA](#) | [BACKOUT TRANSACTION](#) | [BEFORE BREAK PROCESSING](#) | [DELETE](#) | [END TRANSACTION](#) | [FIND](#) | [GET](#) | [GET SAME](#) | [GET TRANSACTION DATA](#) | [HISTOGRAM](#) | [LIMIT](#) | [PASSW](#) | [PERFORM BREAK PROCESSING](#) | [READ](#) | [RETRY](#) | [STORE](#) | [UPDATE](#)

Gehört zur Funktionsgruppe: [Datenbankzugriffe und Datenbankänderungen](#)

Funktion

Das Statement `AT BREAK` dient dazu, in einer mit `FIND`, `READ`, `HISTOGRAM`, `SORT` oder `READ WORK FILE` initiierten Verarbeitungsschleife eine an einen automatischen Gruppenwechsel geknüpfte Verarbeitung anzugeben. Mit dem `AT BREAK`-Statement können Sie ein oder mehrere andere Statements angeben, die jedesmal ausgeführt werden sollen, wenn der Wert eines bestimmten Feldes (**Kontrollfeld**) sich ändert.

Die automatische Gruppenwechsel-Verarbeitung funktioniert folgendermaßen: Unmittelbar nachdem ein Datensatz in der Verarbeitungsschleife gelesen worden ist, wird das Kontrollfeld geprüft. Wenn im Vergleich zum vorangegangenen Datensatz eine Wertänderung festgestellt wird, dann werden die im `AT BREAK`-Statement-Block enthaltenen Statements ausgeführt. Dies gilt nicht für den ersten Datensatz in der Verarbeitungsschleife. Zusätzlich wird am Ende der Verarbeitungsschleife (weil alle Datensätze gelesen sind oder wegen eines `ESCAPE BOTTOM`-Statements) eine letzte Ausführung der `AT BREAK`-Statement-Block enthaltenen Statements veranlasst.

Weitere Informationen siehe *Automatische Gruppenwechsel-Verarbeitung* im Leitfaden zur Programmierung.

Ein `AT BREAK`-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die Gruppenwechsel-Bedingung auftritt, aktiv ist.

Es ist auch möglich, mit einer `AT BREAK`-Verarbeitung eine weitere Verarbeitungsschleife zu initiieren. Die Schleife muss allerdings innerhalb der `AT BREAK`-Verarbeitung wieder beendet werden.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Natural-Systemfunktionen können in Verbindung mit einem `AT BREAK`-Statement benutzt werden, siehe *Natural-Systemfunktionen für Verarbeitungsschleifen* in der *Systemfunktionen-Dokumentation* und *Beispiel für Systemfunktionen mit AT BREAK-Statement* im *Leitfaden zur Programmierung*.

Weitere Informationen siehe auch den Abschnitt *AT BREAK-Statement* im *Leitfaden zur Programmierung*. Darin werden Themen behandelt wie zum Beispiel:

- Gruppenwechsel basierend auf einem Datenbankfeld
- Gruppenwechsel basierend auf einer Benutzervariablen

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S	A U N P I F B D T L	ja	nein

Syntax-Element-Beschreibung:

<i>(r)</i>	<p>Referenzierungsnotation: Ein <code>AT BREAK</code>-Statement wird zum letztenmal ausgeführt, wenn eine mit <code>FIND</code>, <code>READ</code>, <code>READ WORK FILE</code>, <code>HISTOGRAM</code> oder <code>SORT</code> initiierte Verarbeitungsschleife beendet wird. Normalerweise bezieht sich das <code>AT BREAK</code>-Statement hierbei auf die äußerste aktive Schleife.</p> <p>Wollen Sie, dass sich die abschließende <code>AT BREAK</code>-Verarbeitung auf eine andere offene Schleife bezieht (die Schleife, in der das <code>AT BREAK</code>-Statement steht, oder eine äußere Schleife), so verwenden Sie hierzu die Notation (r), wobei r das Statement-Label bzw. die Sourcecode-Zeilenummer des betreffenden schleifeninitiiierenden Statements ist.</p> <p>Beispiel:</p>
------------	---

	<pre> 0110 ... 0120 READ ... 0130 FIND ... 0140 FIND ... 0150 AT BREAK ... 0160 FIND ... 0170 END-FIND 0180 END-BREAK 0190 END-FIND 0200 END-FIND 0210 END-READ 0220 ... </pre>
	<p>In diesem Beispiel bezieht sich die abschließende AT BREAK-Bedingung auf die in Zeile 0120 initiierte READ-Schleife. Es wäre auch möglich, sie an eine der in Zeile 0130 bzw. 0140 initiierten FIND-Schleifen zu knüpfen, nicht jedoch an die in Zeile 0160 initiierte.</p> <p>Soll eine ganze Hierarchie von AT BREAK-Statements sich auf eine andere als die aktive Schleife beziehen, so müssen Sie die Notation (<i>r</i>) bei dem ersten AT BREAK-Statement angeben; sie bezieht sich dann auch auf alle innerhalb der Hierarchie folgenden AT BREAK-Statements.</p>
<i>operand1</i>	<p>Kontrollfeld: In der Regel wird als Kontrollfeld ein Datenbankfeld verwendet. Sie können aber auch eine Benutzervariable nehmen, müssen diese allerdings vor der Gruppenwechsel-Verarbeitung definiert haben (siehe BEFORE BREAK PROCESSING-Statement). Sie können auch eine bestimmte Ausprägung eines Arrays als Kontrollfeld verwenden.</p>
<i>/n/</i>	<p>Sie haben auch die Möglichkeit, einen Teil eines Feldes zum Kontrollfeld zu machen:</p> <p>Mit der Notation <i>/n/</i> geben Sie an, dass nur die ersten <i>n</i> Stellen (von links nach rechts) des Feldes als Kontrollfeld dienen sollen, d.h. das AT BREAK-Statement wird nur ausgeführt, wenn der Wert der ersten <i>n</i> Stellen sich ändert. Diese Möglichkeit besteht allerdings nur bei Feldern, die das Format A, B, N oder P haben.</p> <p>Ein AT BREAK-Statement wird immer dann ausgeführt, wenn ein Gruppenwechsel stattfindet, das heißt, wenn der Wert des Kontrollfeldes sich ändert. Es wird außerdem ausgeführt, nachdem alle Datensätze in der Schleife, auf die sich das AT BREAK-Statement bezieht, verarbeitet worden sind.</p>
END-BREAK	<p>Das reservierte Natural-Wort END-BREAK muss zum Beenden des AT BREAK-Statements benutzt werden.</p>

Gruppenwechsel auf mehreren Ebenen

Innerhalb einer Verarbeitungsschleife in demselben Programm-Modul können Sie mehrere AT BREAK-Statements verwenden. Damit schaffen Sie eine Hierarchie von AT BREAK-Statements, und zwar unabhängig davon, ob die AT BREAK-Statements unmittelbar aufeinander folgen oder zwischen ihnen noch andere Statements stehen. Das erste AT BREAK-Statement befindet sich auf der untersten Ebene der Hierarchie, jedes weitere auf einer nächsthöheren.

Für jede Schleife können Sie in einer Schleife eine eigene AT BREAK-Hierarchie aufbauen.

Beispiel:

Structured Mode:	Reporting Mode:
<pre> FIND ... AT BREAK ... END-BREAK AT BREAK ... END-BREAK AT BREAK ... END-BREAK END-FIND ... </pre>	<pre> FIND ... AT BREAK DO ... DOEND AT BREAK DO ... DOEND ... </pre>

Bei einem Gruppenwechsel auf einer bestimmten Ebene werden auch alle AT BREAK-Statements auf jeweils untergeordneten Ebenen der Hierarchie ausgeführt, unabhängig davon, ob im Kontrollfeld einer unteren Ebene ebenfalls ein Gruppenwechsel stattgefunden hat.

Der Übersichtlichkeit halber empfiehlt es sich, die einzelnen AT BREAK-Statements einer Hierarchie unmittelbar aufeinanderfolgen zu lassen.

Siehe auch [Beispiel 3](#) und den Abschnitt *Gruppenwechsel auf mehreren Ebenen* im *Leitfaden zur Programmierung*.

Beispiele

- Beispiel 1 — AT BREAK
- Beispiel 2 — AT BREAK mit der Notation /n/
- Beispiel 3 — AT BREAK mit Gruppenwechselln auf mehreren Ebenen

Weitere Beispiele für AT BREAK siehe *Systemfunktionen in Verarbeitungsschleifen*, Beispiele ATBEX3 und ATBEX4.

Beispiel 1 — AT BREAK

```

** Example 'ATBEX1S': AT BREAK (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 NAME
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY CITY
AT BREAK OF CITY
  SKIP 1
  END-BREAK
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
END-READ
*
END
    
```

Ausgabe des Programms ATBEX1S:

CITY	COUNTRY	NAME
AIKEN	USA	SENKO
AIX EN OTHE	F	GODEFROY
AJACCIO		CANALE
ALBERTSLUND	DK	PLOUG
ALBUQUERQUE	USA	HAMMOND ROLLING FREEMAN

```

                                LINCOLN
ALFRETON                UK      GOLDBERG
ALICANTE                E       GOMEZ

```

Äquivalentes Reporting-Mode-Beispiel: [ATBEX1R](#).

Beispiel 2 — AT BREAK mit der Notation /n/

```

** Example 'ATBEX2': AT BREAK (with /n/ notation)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 DEPT
  2 NAME
END-DEFINE
*
LIMIT 10
READ EMPLOY-VIEW BY DEPT STARTING FROM 'A'
AT BREAK OF DEPT /4/
  SKIP 1
  END-BREAK
  DISPLAY NOTITLE DEPT NAME
END-READ
*
END

```

Ausgabe des Programms ATBEX2:

```

DEPARTMENT          NAME
  CODE
-----
ADMA01      JENSEN
ADMA01      PETERSEN
ADMA01      MORTENSEN
ADMA01      MADSEN
ADMA01      BUHL
ADMA02      HERMANSEN
ADMA02      PLOUG
ADMA02      HANSEN

COMP01      HEURTEBISE
COMP01      TANCHOU

```

Beispiel 3 — AT BREAK mit Gruppenwechseln auf mehreren Ebenen

```

** Example 'ATBEX5S': AT BREAK (multiple break levels) (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 DEPT
  2 NAME
  2 LEAVE-DUE
1 #LEAVE-DUE-L (N4)
END-DEFINE
*
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'PHILADELPHIA' OR = 'PITTSBURGH'
      SORTED BY CITY DEPT
      MOVE LEAVE-DUE TO #LEAVE-DUE-L
      DISPLAY CITY (IS=ON) DEPT (IS=ON) NAME #LEAVE-DUE-L
/*
AT BREAK OF DEPT
  WRITE NOTITLE /
      T*DEPT OLD(DEPT) T*#LEAVE-DUE-L SUM(#LEAVE-DUE-L) /
END-BREAK
AT BREAK OF CITY
  WRITE NOTITLE
      T*CITY OLD(CITY) T*#LEAVE-DUE-L SUM(#LEAVE-DUE-L) //
END-BREAK
END-FIND
*
END
    
```

Ausgabe des Programms ATBEX5:

CITY	DEPARTMENT CODE	NAME	#LEAVE-DUE-L
PHILADELPHIA	MGMT30	WOLF-TERROINE	11
		MACKARNESS	27
	MGMT30		38
	TECH10	BUSH	39
		NETTLEFOLDS	24
	TECH10		63
PHILADELPHIA			101

PITTSBURGH	MGMT10	FLETCHER	34
	MGMT10		34
PITTSBURGH			34

Äquivalentes Reporting-Mode-Beispiel: [ATBEX5R](#).

8 AT END OF DATA

▪ Funktion	46
▪ Einschränkungen	47
▪ Syntax-Beschreibung	47
▪ Beispiel	48

Structured Mode-Syntax

```
[AT] END [OF] DATA [(r)]
  statement ...
END-ENDDATA
```

Reporting Mode-Syntax

```
[AT] END [OF] DATA [(r)]
{
  statement
  DO statement ... DOEND
}
```

Dieser Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Mit dem Statement AT END OF DATA können Sie eine Verarbeitung angeben, die ausgeführt werden soll, nachdem in einer Verarbeitungsschleife alle Datensätze verarbeitet worden sind.

Dieses Abschnitt behandelt folgende Themen:

- Verarbeitung
- Feldwerte der Datenbankfelder
- Positionierung
- Systemfunktionen

Siehe auch AT START/END OF DATA-Statement im Leitfaden zur Programmierung.

Verarbeitung

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Feldwerte der Datenbankfelder

Zu dem Zeitpunkt, zu dem das AT END OF DATA-Statement ausgeführt wird, enthalten alle Datenbankfelder die Werte des zuletzt verarbeiteten Datensatzes.

Positionierung

Das AT END OF DATA-Statement muss im selben Objektmodul stehen wie das Statement, mit dem die Schleife initiiert wurde.

Systemfunktionen

Natural-Systemfunktionen können in Verbindung mit einem AT END OF DATA-Statement verwendet werden, wie im Abschnitt *Systemfunktionen für Verarbeitungsschleifen benutzen* in der *Systemfunktionen*-Dokumentation beschrieben.

Einschränkungen

- Das Statement kann nur bei einer Verarbeitungsschleife eingesetzt werden, die mit einem der folgenden Statements initiiert wurde: `FIND`, `READ`, `READ WORK FILE`, `HISTOGRAM` oder `SORT`.
- Pro Schleife darf höchstens ein AT END OF DATA-Statement verwendet werden.
- Das AT END OF DATA-Statement wird nur dann ausgeführt, wenn die betreffende Schleife tatsächlich durchlaufen wird.

Syntax-Beschreibung

(r)	<p>Referenzieren einer bestimmten Verarbeitungsschleife: Normalerweise bezieht sich das Statement AT END OF DATA auf die jeweils äußerste aktive Verarbeitungsschleife. Wollen Sie, dass es sich auf eine andere aktive Schleife bezieht, so verwenden Sie hierzu die Notation (r), wobei r das Statement-Label oder die Sourcecode-Zeilenummer des Statements ist, welches die gewünschte Schleife initiiert.</p>
END-ENDDATA	Das reservierte Natural-Wort END-ENDDATA muss zum Beenden des AT END OF DATA-Statements benutzt werden.

Beispiel

```
** Example 'AEDEX1S': AT END OF DATA
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
    SALARY (1) CURR-CODE (1)
/*
  AT END OF DATA
  IF *COUNTER (EMP.) = 0
    WRITE 'NO RECORDS FOUND'
    ESCAPE BOTTOM
  END-IF
  WRITE NOTITLE / 'SALARY STATISTICS:'
    / 7X 'MAXIMUM:' MAX(SALARY(1)) CURR-CODE (1)
    / 7X 'MINIMUM:' MIN(SALARY(1)) CURR-CODE (1)
    / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDDATA
/*
END-FIND
*
END
```

Siehe auch *Natural-Systemfunktionen für Verarbeitungsschleifen*.

Ausgabe des Programms AEDEX1S:

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
11100328	BERGHAUS	ROSE	70800	DM
11100329	BARTHEL	PETER	42000	DM
11300313	AECKERLE	SUSANNE	55200	DM
11300316	KANTE	GABRIELE	61200	DM
11500304	KLUGE	ELKE	49200	DM
SALARY STATISTICS:				
	MAXIMUM:	70800	DM	
	MINIMUM:	42000	DM	
	AVERAGE:	55680	DM	

Äquivalentes Reporting-Mode-Beispiel: [AEDEX1R](#).

9 AT END OF PAGE

▪ Funktion	52
▪ Syntax-Beschreibung	54
▪ Beispiele	54

Structured Mode-Syntax

```
[AT] END [OF] PAGE [(rep)]
  statement ...
END-ENDPAGE
```

Reporting Mode-Syntax

```
[AT] END [OF] PAGE [(rep)]
{
  statement
  DO statement ... DOEND
}
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [AT TOP OF PAGE](#) | [CLOSE PRINTER](#) | [DEFINE PRINTER](#) | [DISPLAY](#) | [EJECT](#) | [FORMAT](#) | [NEWPAGE](#) | [PRINT](#) | [SKIP](#) | [SUSPEND IDENTICAL SUPPRESS](#) | [WRITE](#) | [WRITE TITLE](#) | [WRITE TRAILER](#)

Gehört zur Funktionsgruppe: [Erstellen von Ausgabe-Reports](#)

Funktion

Mit dem `AT END OF PAGE`-Statement können Sie eine Verarbeitung angeben, die ausgeführt werden soll, wenn das Ende einer logischen Seite erreicht ist (End-of-Page-Bedingung; siehe Session-Parameter `PS` in der *Parameter-Referenz*). Eine End-of-Page-Bedingung kann auch aufgrund eines `SKIP`- oder `NEWPAGE`-Statements auftreten, nicht aber aufgrund eines `EJECT`- oder `INPUT`-Statements.

Siehe auch die folgenden Abschnitte im *Leitfaden zur Programmierung*::

- *Steuerung der Ausgabe von Daten*
- *Report-Spezifikation – (rep)-Notation*
- *Layout einer Ausgabeseite*
- *AT END OF PAGE-Statement*

Verarbeitung

Ein `AT END OF PAGE`-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die End-of-Page-Bedingung auftritt, aktiv ist.

Ein `AT END OF PAGE`-Statement darf nicht in einer internen Subroutine stehen.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Länge der logischen Seite

Da erst überprüft wird, ob eine End-of-Page-Bedingung besteht, nachdem ein `DISPLAY`- oder `WRITE`-Statement vollständig ausgeführt ist, kann es vorkommen, dass die von einem `DISPLAY`- oder `WRITE`-Statement erzeugte Ausgabe das Ende einer physischen Seite bereits überschritten hat, bevor eine End-of-Page-Bedingung entdeckt wird.

Um dies zu vermeiden und um sicherzustellen, dass über ein `AT END OF PAGE`-Statement ausgegebene Informationen wirklich am Ende einer physischen Ausgabeseite erscheint, muss die logische Seitenlänge (Session-Parameter `PS`) entsprechend kleiner als die Länge einer physischen Ausgabeseite gesetzt werden.

Letzte Seite

In einem Hauptprogramm ist eine End-of-Page-Bedingung auch dann gegeben, wenn die Ausführung des Programms durch ein `ESCAPE`-, `STOP`- oder `END`-Statement beendet wird.

In einer Subroutine gilt dies nicht; das heißt, `ESCAPE-ROUTINE`, `RETURN` oder `END-SUBROUTINE` lösen in einer Subroutine keine End-of-Page-Bedingung aus.

Systemfunktionen

Natural-Systemfunktionen können in Verbindung mit einem `AT END OF DATA`-Statement verwendet werden, wie im Abschnitt *Systemfunktionen für Verarbeitungsschleifen* benutzen in der *Systemfunktionen*-Dokumentation beschrieben.

Wenn eine Systemfunktion in einem `AT END OF PAGE`-Statement-Block verwendet wird, muss das betreffende `DISPLAY`-Statement eine `GIVE SYSTEM FUNCTIONS`-Klausel enthalten.

INPUT-Statement im AT END OF PAGE

Wenn Sie im AT END OF PAGE-Block ein INPUT-Statement verwenden, wird keine Seitenvorschub-Operation ausgeführt. Sie müssen in diesem Fall den Wert des Session-Parameters PS soweit reduzieren, dass die vom INPUT-Statement erzeugten Zeilen noch auf derselben physischen Seite Platz haben.

Siehe auch:

- *Geteilter Schirm (Split Screen)* beim INPUT-Statement
- *Beispiel 2 – AT END OF PAGE mit INPUT-Statement*

Syntax-Beschreibung

<i>(rep)</i>	<p>Report-Spezifikation: Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls (<i>rep</i>) nicht angegeben wird, bezieht sich das AT END OF PAGE-Statement auf den ersten Report (Report 0).</p> <p>Informationen zum Steuern des Formats eines mit Natural erzeugten Ausgabe-Reports siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
END-ENDPAGE	Das reservierte Natural-Wort END-ENDPAGE muss zum Beenden des AT END OF PAGE-Statements benutzt werden.

Beispiele

- *Beispiel 1 – AT END OF PAGE*
- *Beispiel 2 – AT END OF PAGE mit INPUT-Statement*

Beispiel 1 – AT END OF PAGE

```

** Example 'AEPEX1S': AT END OF PAGE (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY      (1)
    
```

```

2 CURR-CODE (1)
END-DEFINE
*
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM Funktions
    NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*

AT END OF PAGE
  WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
END-ENDPAGE

END-READ
*
END

```

Siehe auch *Systemfunktionen für Verarbeitungsschleifen*.

Ausgabe des Programms AEPEX1S:

NAME	CURRENT POSITION	SALARY	CURRENCY CODE
CREMER	ANALYST	34000	USD
MARKUSH	TRAINEE	22000	USD
GEE	MANAGER	39500	USD
KUNEY	DBA	40200	USD
NEEDHAM	PROGRAMMER	32500	USD
JACKSON	PROGRAMMER	33000	USD
AVERAGE SALARY: ...		33533	USD

Äquivalentes Reporting-Mode-Beispiel: [AEPEX1R](#).

Beispiel 2 — AT END OF PAGE mit INPUT-Statement

```

** Example 'AEPEX2': AT END OF PAGE (with INPUT)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 POST-CODE
  2 CITY
*

```

AT END OF PAGE

```
1 #START-NAME (A20)
END-DEFINE
*
FORMAT PS=21
*
REPEAT
  READ (15) EMPLOY-VIEW BY NAME = #START-NAME
  DISPLAY NOTITLE NAME FIRST-NAME POST-CODE CITY
END-READ
NEWPAGE
/*
AT END OF PAGE
MOVE NAME TO #START-NAME
INPUT / '-' (79)
  / 10T 'Reposition to name ==>'
  #START-NAME (AD=MI) '('.'.' to exit)'
IF #START-NAME = '.'
  STOP
END-IF
END-ENDPAGE
/*
END-REPEAT
END
```

Ausgabe des Programms AEPEX2S:

NAME	FIRST-NAME	POSTAL ADDRESS	CITY
ABELLAN	KEPA	28014	MADRID
ACHIESON	ROBERT	DE3 4TR	DERBY
ADAM	SIMONE	89300	JOIGNY
ADKINSON	JEFF	11201	BROOKLYN
ADKINSON	PHYLLIS	90211	BEVERLEY HILLS
ADKINSON	HAZEL	20760	GAITHERSBURG
ADKINSON	DAVID	27514	CHAPEL HILL
ADKINSON	CHARLIE	21730	LEXINGTON
ADKINSON	MARTHA	17010	FRAMINGHAM
ADKINSON	TIMMIE	17300	BEDFORD
ADKINSON	BOB	66044	LAWRENCE
AECKERLE	SUSANNE	7000	STUTTGART
AFANASSIEV	PHILIP	39401	HATTIESBURG
AFANASSIEV	ROSE	60201	EVANSTON
AHL	FLEMMING	2300	SUNDBY

Reposition to name ==> AHL (.'.' to exit)

10 AT START OF DATA

▪ Funktion	58
▪ Syntax-Beschreibung	59
▪ Beispiel	59

Structured Mode-Syntax

```
[AT] START [OF] DATA [(r)]
  statement ...
END-START
```

Reporting Mode-Syntax

```
[AT] START [OF] DATA [(r)]
{
  statement
  DO statement... DOEND
}
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `ACCEPT/REJECT` | `AT BREAK` | `AT END OF DATA` | `BACKOUT TRANSACTION` | `BEFORE BREAK PROCESSING` | `DELETE` | `END TRANSACTION` | `FIND` | `GET` | `GET SAME` | `GET TRANSACTION DATA` | `HISTOGRAM` | `LIMIT` | `PASSW` | `PERFORM BREAK PROCESSING` | `READ` | `RETRY` | `STORE` | `UPDATE`

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Mit dem Statement `AT START OF DATA` können Sie eine Verarbeitung angeben, die ausgeführt werden soll, unmittelbar nachdem der erste Datensatz einer mit einem Statement `READ`, `FIND`, `HISTOGRAM`, `SORT` oder `READ WORK FILE` initiierten Verarbeitungsschleife gelesen worden ist.

Siehe auch *AT START/END OF DATA-Statement* im *Leitfaden zur Programmierung*.

Verarbeitung

Falls das schleifeninitiiierende Statement eine `WHERE`-Klausel enthält, wird die `AT START OF DATA`-Verarbeitung erst dann ausgeführt, wenn der erste Datensatz gelesen wird, der sowohl das primäre Suchkriterium als auch die `WHERE`-Bedingung erfüllt.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Feldwerte der Datenbankfelder

Zu dem Zeitpunkt, zu dem das AT START OF DATA-Statement ausgeführt wird, enthalten alle Datenbankfelder die Werte des zuerst verarbeiteten Datensatzes (d.h. des ersten Datensatzes, der die AT START OF DATA-Bedingung erfüllt).

Positionierung

Das AT START OF DATA-Statement muss *innerhalb* der betreffenden Verarbeitungsschleife stehen. Pro Verarbeitungsschleife darf höchstens ein AT START OF DATA-Statement verwendet werden.

Syntax-Beschreibung

<i>(r)</i>	Referenzieren einer bestimmten Verarbeitungsschleife: Normalerweise bezieht sich das Statement AT START OF DATA auf die jeweils äußerste aktive Verarbeitungsschleife. Wollen Sie, dass es sich auf eine andere aktive Schleife bezieht, so verwenden Sie hierzu die Notation (<i>r</i>), wobei <i>r</i> das Statement-Label oder die Sourcecode-Zeilenummer des Statements ist, welches die gewünschte Schleife initiiert.
END-START	Das reservierte Natural-Wort END-START muss zum Beenden des AT START OF DATA-Statements benutzt werden.

Beispiel

```

** Example 'ASDEX1S': AT START OF DATA (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #CNTL (A1) INIT <' '>
1 #CITY (A20) INIT <' '>
END-DEFINE
*
REPEAT
  INPUT 'ENTER VALUE FOR CITY' #CITY
  IF #CITY = ' ' OR = 'END'
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH CITY = #CITY
  IF NO RECORDS FOUND

```

AT START OF DATA

```
WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
ESCAPE BOTTOM
END-NOREC
/*
AT START OF DATA
INPUT (AD=0) 'RECORDS FOUND' *NUMBER //
          'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=A)
IF #CNTL NE 'D'
  ESCAPE BOTTOM
END-IF
END-START
/*
DISPLAY NAME FIRST-NAME
END-FIND
END-REPEAT
END
```

Ausgabe des Programms ASDEX1S:

```
ENTER VALUE FOR CITY PARIS
```

Nach Eingabe und Bestätigung des Namens der Stadt:

```
RECORDS FOUND          26
ENTER 'D' TO DISPLAY RECORDS D
```

Angezeigte Datensätze:

NAME	FIRST-NAME
MAIZIERE	ELISABETH
MARX	JEAN-MARIE
REIGNARD	JACQUELINE
RENAUD	MICHEL
REMOUE	GERMAINE
LAVENDA	SALOMON
BROUSSE	GUY
GIORDA	LOUIS
SIECA	FRANCOIS
CENSIER	BERNARD
DUC	JEAN-PAUL
CAHN	RAYMOND
MAZUY	ROBERT
FAURIE	HENRI
VALLY	ALAIN
BRETON	JEAN-MARIE

GIGLEUX	JACQUES
KORAB-BRZOZOWSKI	BOGDAN
XOLIN	CHRISTIAN
LEGRIS	ROGER
VVVV	

Äquivalentes Reporting-Mode-Beispiel: [ASDEX1R](#).

11 AT TOP OF PAGE

▪ Funktion	64
▪ Einschränkung	65
▪ Syntax-Beschreibung	65
▪ Beispiel	66

Structured Mode-Syntax

```
[AT] TOP [OF] PAGE [(rep)]  
    statement ...  
END-TOPPAGE
```

Reporting Mode-Syntax

```
[AT] TOP [OF] PAGE [(rep)]  
    { statement          }  
    { DO statement ... DOEND }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [AT END OF PAGE](#) | [CLOSE PRINTER](#) | [DEFINE PRINTER](#) | [DISPLAY](#) | [EJECT](#) | [FORMAT](#) | [NEWPAGE](#) | [PRINT](#) | [SKIP](#) | [SUSPEND IDENTICAL SUPPRESS](#) | [WRITE](#) | [WRITE TITLE](#) | [WRITE TRAILER](#)

Gehört zur Funktionsgruppe: [Erstellen von Ausgabe-Reports](#)

Funktion

Mit dem `AT TOP OF PAGE`-Statement können Sie eine Verarbeitung angeben, die ausgeführt werden soll, wenn eine neue Seite beginnt.

Siehe auch folgende Abschnitte im *Leitfaden zur Programmierung*:

- *Steuern der Ausgabe von Daten*
- *Report-Spezifikation – (rep) Notation*
- *Layout einer Ausgabeseite*
- *AT TOP OF PAGE-Statement*

Verarbeitung

Eine neue Seite beginnt, wenn entweder die ausgegebene Zeilenzahl die mit dem Session-Parameter PS gesetzte Seitenlänge überschreitet oder ein **NEWPAGE**-Statement ausgeführt wird. Ein **EJECT**-Statement führt ebenfalls zu einem Seitenvorschub, löst aber keine AT TOP OF PAGE-Verarbeitung aus.

Ein AT TOP OF PAGE-Statement-Block wird nur ausgeführt, wenn das Objekt, das den Statement-Block enthält, zu dem Zeitpunkt, zu dem die Top-of-Page-Bedingung auftritt, aktiv ist.

Erzeugt ein AT TOP OF PAGE-Statement eine Ausgabe, so wird diese unter der Seitentitelzeile ausgegeben, wobei zwischen beiden automatisch eine Leerzeile ausgegeben wird.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkung

Ein AT TOP OF PAGE-Statement darf nicht in einer internen Subroutine stehen.

Syntax-Beschreibung

<i>(rep)</i>	<p>Report-Spezifikation: Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das AT TOP OF PAGE-Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls (<i>rep</i>) nicht angegeben wird, bezieht sich das AT TOP OF PAGE-Statement auf den ersten Report (Report 0).</p> <p>Weitere Informationen zur Steuerung des Formats eines mit Natural erzeugten Ausgabe-Reports siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
END-TOPPAGE	Das reservierte Natural-Wort END-TOPPAGE muss zum Beenden des AT TOP OF PAGE-Statements benutzt werden.

Beispiel

```

** Example 'ATPEX1S': AT TOP OF PAGE (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 DEPT
END-DEFINE
*
FORMAT PS=15
LIMIT 15
READ EMPLOY-VIEW BY NAME STARTING FROM 'L'
  DISPLAY 2X NAME 4X FIRST-NAME CITY DEPT
  WRITE TITLE UNDERLINED 'EMPLOYEE REPORT'
  WRITE TRAILER '-' (78)
/*
  AT TOP OF PAGE
    WRITE 'BEGINNING NAME:' NAME
  END-TOPPAGE
/*
  AT END OF PAGE
    SKIP 1
    WRITE 'ENDING NAME:  ' NAME
  END-ENDPAGE
END-READ
END

```

Ausgabe des Programms ATPEX1S:

EMPLOYEE REPORT			

BEGINNING NAME: LAFON			
NAME	FIRST-NAME	CITY	DEPARTMENT CODE
-----	-----	-----	-----
LAFON	CHRISTIANE	PARIS	VENT18
LANDMANN	HARRY	ESCHBORN	MARK29
LANE	JACQUELINE	DERBY	MGMT02
LANKATILLEKE	LALITH	FRANKFURT	PROD22
LANNON	BOB	LINCOLN	SALE20
LANNON	LESLIE	SEATTLE	SALE30
LARSEN	CARL	FARUM	SYSA01
LARSEN	MOGENS	VERMELEV	SYSA02

ENDING NAME: LARSEN

Äquivalentes Reporting-Mode-Beispiel: [ATPEX1R](#).

12 BACKOUT TRANSACTION

▪ Funktion	70
▪ Einschränkung	71
▪ Datenbank-spezifische Anmerkungen	71
▪ Beispiel	71

BACKOUT [TRANSACTION]

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das BACKOUT TRANSACTION-Statement bewirkt, dass alle Datenbankänderungen, die während der laufenden, noch nicht abgeschlossenen logischen Transaktion ausgeführt wurden, rückgängig gemacht werden; außerdem bewirkt es, dass alle während der Transaktion gehaltenen Datensätze wieder freigegeben werden.

Das BACKOUT TRANSACTION-Statement wird nur ausgeführt, wenn eine Datenbanktransaktion unter Kontrolle von Natural stattgefunden hat. Für welche Datenbanken das Statement ausgeführt wird, hängt davon ab, wie der Profilparameter ET (Ausführung von END/BACKOUT TRANSACTION-Statements) gesetzt ist:

- Ist ET=OFF gesetzt, wird das Statement nur für die von der Transaktion betroffene Datenbank ausgeführt.
- Ist ET=ON gesetzt, wird das Statement für alle Datenbanken ausgeführt, die seit der letzten Ausführung eines BACKOUT TRANSACTION- oder END TRANSACTION-Statements referenziert wurden.

BACKOUT TRANSACTION über Abbruchkommando

Unterbricht der Benutzer mit einem Natural-Terminalkommando (Kommando %% oder CLEAR-Taste) eine gerade aktive Natural-Operation, dann führt Natural ein BACKOUT TRANSACTION-Statement aus.

Weitere Informationen siehe Terminalkommando %% in der *Terminalkommando*-Dokumentation.

Weitere Informationen

Weitere Informationen zur Natural-Transaktionslogik und zum Beenden/Abbrechen einer logischen Transaktion finden Sie im Kapitel *Datenbankzugriffe* im *Leitfaden zur Programmierung*.

Einschränkung

Mit Entire System Server kann dieses Statement nicht verwendet werden.

Datenbank-spezifische Anmerkungen

SQL-Datenbanken	Da die meisten SQL-Datenbanken bei Beendigung einer logischen Arbeitseinheit alle Cursor schließen, darf ein BACKOUT TRANSACTION-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muss nach einer solchen platziert werden.
XML-Datenbanken	Ein BACKOUT TRANSACTION-Statement darf nicht in einer datenbankverändernde Programmschleife, sondern muss nach einer solchen platziert werden.

Beispiel

```

** Example 'BOTEX1': BACKOUT TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 DEPT
  2 LEAVE-DUE
  2 LEAVE-TAKEN
*
1 #DEPT (A6)
1 #RESP (A3)
END-DEFINE
*
LIMIT 3
INPUT 'DEPARTMENT TO BE UPDATED:' #DEPT
IF #DEPT = ' '
  STOP
END-IF

```

BACKOUT TRANSACTION

```
*
FIND EMPLOY-VIEW WITH DEPT = #DEPT
  IF NO RECORDS FOUND
    REINPUT 'NO RECORDS FOUND'
  END-NOREC
  INPUT 'NAME:          ' NAME (AD=0) /
        'LEAVE DUE:    ' LEAVE-DUE (AD=M) /
        'LEAVE TAKEN:' LEAVE-TAKEN (AD=M)
  UPDATE
END-FIND
*
INPUT 'UPDATE TO BE PERFORMED? YES/NO:' #RESP
DECIDE ON FIRST #RESP
  VALUE 'YES'
  END TRANSACTION
  VALUE 'NO'
  BACKOUT TRANSACTION
  NONE
  REINPUT 'PLEASE ENTER YES OR NO'
END-DECIDE
*
END
```

Ausgabe des Programms BOTEX1:

```
DEPARTMENT TO BE UPDATED: MGMT30
```

Ergebnis für die Abteilung MGMT30:

```
NAME:          POREE
LEAVE DUE:     45
LEAVE TAKEN:   31
```

Aufforderung zur Bestätigung:

```
UPDATE TO BE PERFORMED YES/NO: NO
```

13 BEFORE BREAK PROCESSING

▪ Funktion	74
▪ Einschränkungen	75
▪ Syntax-Beschreibung	75
▪ Beispiel	76

Structured Mode-Syntax

```
BEFORE [BREAK] [PROCESSING]
    statement ...
END-BEFORE
```

Reporting Mode-Syntax

```
[BEFORE [BREAK] [PROCESSING]
{
    statement
    DO statement ... DOEND
}
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement BEFORE BREAK PROCESSING wird im Zusammenhang mit einem automatischen Gruppenwechsel verwendet, und zwar um Verarbeitungen anzugeben, die ausgeführt werden sollen:

- bevor der Wert des Gruppenwechsel-Kontrollfeldes geprüft wird;
- bevor ein AT BREAK-Statement-Block ausgeführt wird;
- bevor Natural-Systemfunktionen ausgewertet werden.

Meistens wird BEFORE BREAK PROCESSING eingesetzt, um Benutzervariablen zu initialisieren oder zu berechnen, die bei einer anschließenden Gruppenwechsel-Verarbeitung (siehe AT BREAK-Statement) benutzt werden sollen.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Siehe auch die folgenden Abschnitte im *Leitfaden zur Programmierung*:

- *Gruppenwechsel*

- *BEFORE BREAK PROCESSING-Statement*
- *Beispiel für BEFORE BREAK PROCESSING-Statement*

Einschränkungen

- Das Statement `BEFORE BREAK PROCESSING` kann nur in Verbindung mit einer Verarbeitungsschleife verwendet werden, die mit den folgenden Statements initiiert wird:
 - `FIND`
 - `READ`
 - `HISTOGRAM`
 - `SORT`
 - `READ WORK FILE`

In einer Verarbeitungsschleife darf höchstens ein `BEFORE BREAK PROCESSING`-Statement stehen. Das Statement darf an beliebiger Stelle innerhalb einer Schleife stehen und bezieht sich immer auf die Schleife, in der es steht.

- Ein `BEFORE BREAK PROCESSING`-Statement darf nicht in Verbindung mit einem `PERFORM BREAK PROCESSING`-Statement verwendet werden.

Syntax-Beschreibung

<i>statement...</i>	Siehe Beispiel weiter unten. Wird keine an einen Gruppenwechsel geknüpfte Verarbeitung ausgeführt (d.h. wenn die betreffende Verarbeitungsschleife kein <code>AT BREAK</code> -Statement enthält), so wird der <code>BEFORE BREAK PROCESSING</code> -Statement-Block <i>nicht</i> ausgeführt.
END-BEFORE	Im Structured Mode: Das reservierte Natural-Wort <code>END-BEFORE</code> muss zum Beenden des <code>BEFORE BREAK PROCESSING</code> -Statements benutzt werden.

Beispiel

```

** Example 'BBPEX1': BEFORE BREAK PROCESSING
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY (1)
  2 BONUS (1,1)
*
1 #INCOME (P11)
END-DEFINE
*
LIMIT 7
READ EMPLOY-VIEW BY CITY = 'L'
/*
  BEFORE BREAK PROCESSING
    COMPUTE #INCOME = SALARY (1) + BONUS (1,1)
  END-BEFORE
/*
  AT BREAK OF CITY
    WRITE NOTITLE 'AVERAGE INCOME FOR' OLD (CITY) 20X AVER(#INCOME) /
  END-BREAK
/*
  DISPLAY CITY 'NAME' NAME 'SALARY' SALARY (1) 'BONUS' BONUS (1,1)
END-READ
END

```

Ausgabe des Programms BBPEX1:

CITY	NAME	SALARY	BONUS	
LA BASSEE	HULOT	165000	70000	
AVERAGE INCOME FOR LA BASSEE				235000
LA CHAPELLE ST LUC	GUILLARD	124100	23000	
LA CHAPELLE ST LUC	BERGE	198500	50000	
LA CHAPELLE ST LUC	POLETTE	124090	23000	
LA CHAPELLE ST LUC	DELAUNEY	115000	23000	
LA CHAPELLE ST LUC	SCHECK	125600	23000	
LA CHAPELLE ST LUC	KREEBS	184550	50000	
AVERAGE INCOME FOR LA CHAPELLE ST LUC				177306

14 CALL

▪ Funktion	78
▪ Syntax-Beschreibung	78
▪ Return Code	79
▪ User Exits	80
▪ INTERFACE4	80

CALL [INTERFACE4] *operand1* [USING] [*operand2*] ... 128

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: CALL FILE | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | ESCAPE | FETCH | PERFORM

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Mit dem CALL-Statement können Sie von einem Natural-Programm aus ein anderes, in einer anderen Standard-Programmiersprache geschriebenes Programm oder eine Funktion aufrufen, wobei im Anschluss daran die Verarbeitung des Natural-Programms mit dem nächsten Statement nach dem CALL-Statement fortgesetzt wird.

Das aufgerufene Program oder die Function kann in einer beliebigen anderen Programmiersprache, die eine Standard-CALL-Schnittstelle unterstützt, geschrieben sein. Mehrere CALL-Statements können verwendet werden, um ein Programm oder eine Function mehrmals oder mehrere Programme oder Functions aufzurufen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate										Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	C	S			A												ja	nein
<i>operand2</i>	C	S	A	G	A	U	N	P	I	F	B	D	T	L	C	G	ja	ja

Syntax-Element-Beschreibung:

INTERFACE4	Das optionale Schlüsselwort INTERFACE4 gibt den Typ der Schnittstelle an, die für den Aufruf des externen Programms verwendet wird. Siehe den Abschnitt INTERFACE4 weiter unten.
<i>operand1</i>	<p>Name des aufgerufenen Programms bzw. der aufgerufenen Function:</p> <p>Der Name der aufgerufenen Function (<i>operand1</i>) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Programme oder Functions aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname oder Function-Name muss linksbündig in der Variablen stehen.</p>
[USING] <i>operand2</i>	<p>Parameter:</p> <p>Das CALL-Statement kann bis zu 128 Parameter (<i>operand2</i>) enthalten. Für jedes angegebene Parameterfeld wird in der Parameterliste eine Adresse übergeben.</p> <p>Wird ein Gruppenname verwendet, so wird die Gruppe in einzelne Felder umgesetzt, d.h. der Benutzer muss das erste Feld der Gruppe angeben, falls er die Anfangsadresse einer Gruppe spezifizieren will.</p> <p>Anmerkung: Wenn eine anwendungsunabhängige Variable (AIV) als Parameter an einen User Exit übergeben wird, gilt die folgende Einschränkung: Falls der User Exit ein Natural-Subprogramm aufruft, das eine neue AIV oder Kontextvariable anlegt, dann ist der Parameter nach der Rückkehr vom Subprogramm ungültig. Dies gilt unabhängig davon, ob die neue AIV bzw. Kontextvariable durch das Subprogramm selbst oder durch ein anderes, direkt oder indirekt von dem Subprogramm aufgerufenes Objekt angelegt wird.</p>

Return Code

Um den Condition Code eines aufgerufenen Programms oder einer Function zu erhalten, können Sie die Natural-Systemfunktion **RET** verwenden.

Beispiel:

```

...
RESET #RETURN(B4)
CALL 'PROG1'
IF RET ('PROG1') > #RETURN
  WRITE 'ERROR OCCURRED IN PROGRAM1'
END-IF
...

```

User Exits

User Exits werden benötigt, um auf externe Funktionen zugreifen zu können, die mit einem CALL-Statement aufgerufen werden. Die User Exits müssen in eine DLL (Dynamic Link Library) gestellt werden. Weitere Informationen zu User Exits finden Sie in der folgenden Datei:

```
%NATDIR%\%NATVERS%\natural\samples\sysexuex\readme.txt
```

Voraussetzung: Um Zugang zur die Datei *readme.txt* zu bekommen, müssen Sie bei der Installation von Natural das Merkmal "Samples" ausgewählt haben.

INTERFACE4

Das Schlüsselwort `INTERFACE4` gibt den Typ der Schnittstelle an, die zum Aufruf des externen Programms verwendet wird. Dieses Schlüsselwort ist optional. Wenn dieses Schlüsselwort angegeben wird, wird die als `INTERFACE4` definierte Schnittstelle zum Aufruf des externen Programms verwendet.

Folgende Themen werden behandelt:

- Unterschiede zwischen CALL-Statement mit und ohne `INTERFACE4`
- `INTERFACE4` — Externe 3GL-Programmierschnittstelle
- Operanden-Struktur für `INTERFACE4`
- `INTERFACE4` — Parameter-Zugriff
- Exportierte Funktionen

Unterschiede zwischen CALL-Statement mit und ohne `INTERFACE4`

Die folgende Tabelle zeigt die Unterschiede zwischen dem mit `INTERFACE4` benutzten CALL-Statement und dem ohne `INTERFACE4` benutzten.

	CALL-Statement ohne Schlüsselwort <code>INTERFACE4</code>	CALL-Statement mit Schlüsselwort <code>INTERFACE4</code>
Anzahl der möglichen Parameter	128	32767
Maximale Länge eines Parameters	65535	1 GB
Array-Informationen einlesen	nein	ja
Unterstützung großer und dynamischer Operanden	nein	ja
Parameter-Zugriff über API	nein	ja

INTERFACE4 — Externe 3GL-Programmierschnittstelle

Die Schnittstelle des externen 3GL-Programms wird wie folgt definiert, wenn INTERFACE4 im Natural-CALL-Statement angegeben wird:

```
NATFCT funktionname (numparm, parmhandle, traditional)
```

USR_WORD	numparm;	16 Bit umfassender Kurzwert ohne Vorzeichen, der die Gesamtzahl der übertragenen Operanden (<i>operand2</i>) enthält
void	*parmhandle;	Adresse der Parameterübergabe-Struktur.
void	*traditional;	Schnittstellen-Typ prüfen (wenn es keine NULL-Adresse ist, handelt es sich um die herkömmliche CALL-Schnittstelle).

Operanden-Struktur für INTERFACE4

Die Operanden-Struktur von INTERFACE4 wird als `parameter_description` bezeichnet und ist wie folgt definiert. Die Struktur wird mit der Header-Datei `natuser.h` ausgeliefert.

struct parameter_description		
void *	address	Adresse der Parameterdaten, nicht ausgerichtet, <code>realloc()</code> und <code>free()</code> sind nicht zulässig.
int	format	Felddatentyp: <code>NCXR_TYPE_ALPHA</code> , usw. (<i>natuser.h</i>).
int	length	Länge vor Dezimalpunkt (wenn zutreffend).
int	precision	Länge hinter Dezimalpunkt (wenn zutreffend).
int	byte_length	Länge des Feldes in Bytes; int Dimensionszahl (0 bis <code>IF4_MAX_DIM</code>)
int	dimensions	Anzahl Dimensionen (0 bis <code>IF4_MAX_DIM</code>).
int	length_all	Gesamtlänge des Arrays in Bytes.
int	flags	Mehrere Flag-Bits, durch OR bitweise miteinander kombiniert, Bedeutung:
	<code>IF4_FLG_PROTECTED</code>	Parameter ist schreibgeschützt.
	<code>IF4_FLG_DYNAMIC</code>	Parameter ist dynamische Variable.
	<code>IF4_FLG_NOT_CONTIGUOUS</code>	Array-Elemente berühren sich nicht (es steht ein Leerzeichen zwischen ihnen).
	<code>IF4_FLG_AIV</code>	Der Parameter ist eine anwendungsunabhängige Variable.
	<code>IF4_FLG_DYNVAR</code>	Parameter ist dynamische Variable.
	<code>IF4_FLG_XARRAY</code>	Parameter ist ein X-Array.
	<code>IF4_FLG_LBVAR_0</code>	Untere Grenze der Dimension 0 ist variabel.

		IF4_FLG_UBVAR_0	Obere Grenze der Dimension 0 ist variabel.
		IF4_FLG_LBVAR_1	Untere Grenze der Dimension 1 ist variabel.
		IF4_FLG_UBVAR_1	Obere Grenze der Dimension 1 ist variabel.
		IF4_FLG_LBVAR_2	Untere Grenze der Dimension 2 ist variabel.
		IF4_FLG_UBVAR_2	Obere Grenze der Dimension 2 ist variabel.
int	occurrences[IF4_MAX_DIM]	Array-Ausprägungen in jeder Dimension.	
int	indexfactors[IF4_MAX_DIM]	Array-Index-Faktoren für jede Dimension.	
void *	dynp	Reserviert für interne Zwecke.	
void *	pops	Reserviert für interne Zwecke.	

Das Adress-Element ist Null für Arrays dynamischer Variablen und für X-Arrays. In diesen Fällen kann auf die Array-Daten nicht als Ganzes zugegriffen werden, sondern es muss über die unten beschriebenen Parameterzugriffsfunktionen auf sie zugegriffen werden.

Für Arrays mit festen Grenzen von Variablen fester Länge kann auf den Array-Inhalt direkt über das Adress-Element zugegriffen werden. In diesen Fällen errechnet sich die Adresse eines Array-Elements (i, j, k) wie folgt (besonders, wenn die Array-Elemente sich nicht berühren):

```
elementaddress = address + i * indexfactors[0] + j * indexfactors[1] + k *
indexfactors[2]
```

Wenn das Array weniger als 3 Dimensionen hat, entfallen die letzten Ausdrücke.

INTERFACE4 — Parameter-Zugriff

Eine Reihe von Funktionen steht für den Zugriff auf die Parameter zur Verfügung. Der Ablauf der Verarbeitung ist wie folgt.

- Das 3GL-Programm wird über das CALL-Statement mit der Option INTERFACE4 aufgerufen, und die Parameter werden an das 3GL-Programm wie oben beschrieben übergeben.
- Das 3GL-Programm kann jetzt die exportierten Funktionen von Natural verwenden, um entweder die Parameterdaten selbst oder Informationen über die Parameter, wie Format, Länge, Array-Informationen usw. einzulesen.
- Die **exportierten Funktionen** dienen auch dazu, Parameterdaten zurückzugeben.

Es gibt außerdem Funktionen zum Erstellen und Initialisieren eines neuen Parameter-Sets, um arbiträre Subprogramme von einem 3GL-Programm aus aufzurufen. Mit dieser Technik ist der Zugriff auf Parameter gewährleistet, um zu verhindern, dass das 3GL-Programm den Speicher

überschreibt. Natural-Daten sind sicher – ein Überschreiben des Speichers im Bereich der Daten des 3GL-Programms ist noch möglich.

Exportierte Funktionen

Folgende Themen werden behandelt:

- Parameter-Informationen holen
- Parameterdaten holen
- Operanden-Daten zurückschreiben
- Parameter-Set erstellen, initialisieren und löschen
- Parameter-Set erstellen
- Parameter-Set löschen
- Skalar eines statischen Datentyps initialisieren
- Array eines statischen Datentyps initialisieren
- Skalar eines dynamischen Datentyps initialisieren
- Array eines dynamischen Datentyps initialisieren
- Größe eines X-Array-Parameters anpassen

Parameter-Informationen holen

Diese Funktion wird vom 3GL-Programm verwendet, um alle erforderlichen Informationen zu Parametern zu erhalten. Diese Informationen werden in einer als `struct parameter_description` bezeichneten, strukturierten Parameterbeschreibung zurückgegeben, siehe [oben](#).

Prototyp:

```
int ncxr_get_parm_info ( int parmnum, void *parmhandle, struct parameter_description *descr );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur	
descr	Adresse einer <code>struct parameter_description</code>	
return	Rückgabewert:	Informationen:
	0	OK
	-1	Fehlerhafte Parameter-Nummer
	-2	Interner Fehler
	-7	Schnittstellen-Versionskonflikt

Parameterdaten holen

Diese Funktion wird vom 3GL-Programm verwendet, um die Daten von beliebigen Parametern zu holen.

Natural identifiziert den Parameter über die vorgegebene Parameter-Nummer und schreibt die Parameterdaten unter der gegebenen Pufferadresse in der gegebenen Pufferlänge.

Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, schneidet Natural die Daten bis auf die gegebene Länge ab. Das externe 3GL-Programm kann die Funktion `ncxr_get_parm_info` nutzen, um die Länge der Parameterdaten abzufragen.

Es gibt zwei Funktionen zum Holen von Parameterdaten: `ncxr_get_parm` holt den gesamten Parameter (auch wenn der Parameter ein Array ist), während `ncxr_get_parm_array` das angegebene Array-Element holt.

Wenn vom 3GL-Programm für `buffer` kein Speicher der angegebenen Größe (dynamisch oder statisch) zugewiesen wird, sind die Ergebnisse der Operation nicht vorhersehbar. Natural überprüft dann nur die Pointer auf Gleichheit mit Null.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse vom Maschinentyp (Little Endian = höherwertiges Byte vorne /Big Endian = höherwertiges Byte hinten) abhängig. In einigen Anwendungen muss der User Exit programmiert werden, um keine statischen Daten zu verwenden, so dass eine Rekursion möglich wird.

Prototypen:

```
int ncxr_get_parm( int parmnum, void *parmhandle, int buffer_length, void *buffer )  
  
int ncxr_get_parm_array( int parmnum, void *parmhandle, int buffer_length, void  
*buffer, int *indexes )
```

Diese Funktion ist identisch mit `ncxr_get_parm`, außer dass die Indizes für jede Dimension angegeben werden können. Die Indizes für unbenutzte Dimensionen sollten als Null (0) angegeben werden.

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
buffer_length	Länge des Puffers, wohin die angeforderten Daten geschrieben werden müssen.	
buffer	Adresse des Puffers, wohin die angeforderten Daten geschrieben werden müssen. Dieser Puffer sollte ausgerichtet werden, um einen leichten Zugriff auf I2/I4/F4/F8-Variablen zu ermöglichen.	
indexes	Array mit Index-Informationen.	
return	Rückgabewert:	Informationen:
	< 0	Fehler beim Einlesen der Informationen.
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-3	Daten wurden abgeschnitten.
	-4	Daten sind kein Array.
	-7	Schnittstellen-Versionskonflikt.
	-100	Index für Dimension 0 liegt außerhalb des zulässigen Bereichs.
	-101	Index für Dimension 1 liegt außerhalb des zulässigen Bereichs.
	-102	Index für Dimension 2 liegt außerhalb des zulässigen Bereichs.
0	Erfolgreiche Ausführung.	
> 0	Erfolgreiche Ausführung, allerdings sind die Daten nur genau diese Anzahl Bytes lang (Puffer war länger als die Daten).	

Operanden-Daten zurückschreiben

Diese Funktionen werden vom 3GL-Programm verwendet, um die Daten auf beliebige Parameter zurückzuschreiben. Natural identifiziert den Parameter über die gegebene Parameter-Nummer und schreibt die Parameterdaten von der gegebenen Pufferadresse in der gegebenen Pufferlänge auf die Parameterdaten.

Wenn die Parameterdaten kürzer als die gegebene Pufferlänge sind, werden die Daten bis auf die Länge der Parameterdaten abgeschnitten, d.h. der Rest des Puffers wird ignoriert. Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, werden die Daten nur in der angegebenen Pufferlänge kopiert, die verbleibenden Parameter bleiben davon unberührt. Dies gilt gleichermaßen für Arrays. Bei dynamischen Variablen als Parameter wird der Parameter auf die angegebene Pufferlänge geändert.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse abhängig vom Maschinentyp (Little Endian = höherwertiges Byte vorne, Big Endian = höherwertiges Byte hinten). In einigen Anwendungen muss der User Exit programmiert werden, um keine statischen Daten zu verwenden, so dass eine Rekursion möglich wird.

Prototypen:

```
int ncxr_put_parm      ( int parmnum, void *parmhandle,
                       int buffer_length, void *buffer );
int ncxr_put_parm_array ( int parmnum, void *parmhandle,
                          int buffer_length, void *buffer,
                          int *indexes );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
buffer_length	Länge der Daten, die in die Adresse des Puffers zurück zu kopieren sind, von wo die Daten herkommen.	
indexes	Index information	
return	Rückgabewert:	Informationen:
	< 0	Fehler beim Zurückkopieren der Informationen.
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-3	Zu viele Daten angegeben. Zurückkopieren erfolgte über die Parameterlänge.
	-4	Parameter ist kein Array.
	-5	Parameter ist geschützt (konstant oder AD=0).
	-6	Die Länge der dynamischen Variable konnte aufgrund einer Out of Memory-Bedingung (kein Speicher verfügbar) nicht geändert werden.
	-7	Schnittstellen-Versionskonflikt.
	-13	Der vorhandene Puffer enthält ein unvollständiges Unicode-Zeichen.
	-100	Index für Dimension 0 liegt außerhalb des zulässigen Bereichs.
	-101	Index für Dimension 1 liegt außerhalb des zulässigen Bereichs.
-102	Index für Dimension 2 liegt außerhalb des zulässigen Bereichs.	

	0	Erfolgreiche Ausführung.
	> 0	Erfolgreiche Ausführung, allerdings sind die Parameter genau diese Anzahl Bytes lang (Länge des Parameters > gegebene Länge).

Parameter-Set erstellen, initialisieren und löschen

Wenn ein 3GL-Programm ein Natural-Subprogramm aufrufen möchte, muss es einen Parameter-Set erstellen, die den Parametern entspricht, welche das Subprogramm erwartet. Die Funktion `ncxr_create_parm` wird benutzt, um einen Parameter-Set zu erstellen, die mit einem Aufruf an `ncxr_if_callnat` übergeben werden sollen. Der erstellte Parameter-Set wird durch eine transparente Parameter-Struktur dargestellt, wie der Parameter-Set, der an das 3GL-Programm mit dem Statement `CALL INTERFACE4` übergeben wird. Somit kann der neu erstellte Parameter-Set mit den Funktionen `ncxr_put_parm*` und `ncxr_get_parm*` wie weiter oben beschrieben bearbeitet werden.

Der neu erstellte Parameter-Set wird noch nicht initialisiert, nachdem die Funktion `ncxr_create_parm` aufgerufen worden ist. Ein einzelner Parameter wird durch einen Set von unten beschriebenen `ncxr_parm_init*`-Funktionen für einen spezifischen Datentyp initialisiert. Die Funktionen `ncxr_put_parm*` und `ncxr_get_parm*` werden dann benutzt, um auf den Inhalt jedes einzelnen Parameters zuzugreifen. Nachdem der Aufrufende den Parameter-Set abgearbeitet hat, müssen sie die Parameter-Struktur löschen. So sieht dann eine typische Reihenfolge bei der Erstellung und Benutzung eines Sets von Parametern für ein Subprogramm aus, das über `ncxr_if4_callnat` aufgerufen werden soll:

```
ncxr_create_parm
ncxr_init_parm*
ncxr_init_parm*
...
ncxr_put_parm*
ncxr_put_parm*
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_if4_callnat
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_get_parm*
ncxr_get_parm*
...
ncxr_delete_parm
```

Parameter-Set erstellen

Die Funktion `ncxr_create_parm` wird benutzt, um einen Parameter-Set zu erstellen, die mit einem Aufruf an `ncxr_if_callnat` übergeben werden soll.

Prototyp:

```
int ncxr_create_parm( int parmnum, void** pparmhandle )
```

Parameter-Beschreibung:

parmnum	Anzahl der zu erstellenden Parameter.	
pparmhandle	Zeiger zur erstellten Parameter-Struktur.	
return	Rückgabewert:	Information:
	< 0	Fehler
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung
0	Erfolgreiche Ausführung.	

Parameter-Set löschen

Die Funktion `ncxr_delete_parm` wird benutzt, um einen Parameter-Set zu löschen, der mit `ncxr_create_parm` erstellt wurde.

Prototyp:

```
int ncxr_delete_parm( void* parmhandle )
```

Parameter-Beschreibung:

parmhandle	Zeiger zu der zu löschenden Parameter-Struktur.	
return	Rückgabewert:	Information:
	< 0	Fehler.
	-2	Interner Fehler.
	0	Erfolgreiche Ausführung.

Skalar eines statischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_s( int parmnum, void *parmhandle,
    char format, int length, int precision, int flags );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
format	Format des Parameters.	
length	Länge des Parameters.	
precision	Präzision des Parameters.	
flags	Eine Kombination der Flags IF4_FLG_PROTECTED	
return	Rückgabewert:	Information:
	< 0	Fehler.
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung.
	-8	Ungültiges Format.
	-9	Ungültige Länge oder Präzision.
	0	Erfolgreiche Ausführung.

Array eines statischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_sa( int parmnum, void *parmhandle,
    char format, int length, int precision,
    int dim, int *occ, int flags );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
format	Format des Parameters.	
length	Länge des Parameters.	
precision	Präzision des Parameters.	
dim	Dimension des Arrays.	
occ	Anzahl der Ausprägungen pro Dimension.	
flags	Eine Kombination der Flags IF4_FLG_PROTECTED IF4_FLG_LBVAR_0 IF4_FLG_UBVAR_0 IF4_FLG_LBVAR_1 IF4_FLG_UBVAR_1 IF4_FLG_LBVAR_2 IF4_FLG_UBVAR_2	
return	Rückgabewert:	Information:
	< 0	Fehler.
	-1	Ungültige Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung.
	-8	Ungültiges Format.
	-9	Ungültige Länge oder Präzision.
	-10	Ungültige Anzahl Dimensionen.
	-11	Ungültige Kombination variabler Grenzen.
0	Erfolgreiche Ausführung.	

Skalar eines dynamischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_d( int parmnum, void *parmhandle,
char format, int flags );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
format	Format des Parameters.	
flags	Eine Kombination der Flags IF4_FLG_PROTECTED	
return	Rückgabewert:	Information:
	< 0	Fehler:
	-1	Ungültige Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung.
	-8	Ungültiges Format.
0	Erfolgreiche Ausführung.	

Array eines dynamischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_da( int parmnum, void *parmhandle,
    char format, int dim, int *occ, int flags );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
format	Format des Parameters.	
dim	Dimension des Arrays.	
occ	Anzahl der Ausprägungen pro Dimension.	
flags	Eine Kombination der Flags IF4_FLG_PROTECTED IF4_FLG_LBVAR_0 IF4_FLG_UBVAR_0 IF4_FLG_LBVAR_1 IF4_FLG_UBVAR_1 IF4_FLG_LBVAR_2 IF4_FLG_UBVAR_2	
return	Rückgabewert:	Information:
	< 0	Fehler.

	-1	Ungültige Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung.
	-8	Ungültiges Format.
	-10	Ungültige Anzahl Dimensionen.
	-11	Ungültige Kombination variabler Grenzen.
	0	Erfolgreiche Ausführung.

Größe eines X-Array-Parameters anpassen

Prototype:

```
int ncxr_resize_parm_array( int parmnum, void *parmhandle, int *occ );
```

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
occ	Neue Anzahl der Ausprägungen pro Dimension	
return	Rückgabewert:	Information:
	< 0	Fehler.
	-1	Ungültige Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung.
	-12	Operand kann größenmäßig nicht angepasst werden (in einer der angegebenen Dimensionen).
	0	Erfolgreiche Ausführung.

Alle Funktionsprototypen sind in der Datei *natuser.h* deklariert.

15 CALL FILE

▪ Funktion	94
▪ Einschränkung	94
▪ Syntax-Beschreibung	95
▪ Beispiel	95

Structured Mode-Syntax

```
CALL FILE 'program-name' operand1 operand2  
    statement ...  
END-FILE
```

Reporting Mode-Syntax

```
CALL FILE 'program-name' operand1 operand2  
    statement ...  
[LOOP]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [CALL](#) | [CALL LOOP](#) | [CALLNAT](#) | [DEFINE SUBROUTINE](#) | [ESCAPE](#) | [FETCH](#) | [PERFORM](#)

Gehört zur Funktionsgruppe: [Aufrufen von Programmen und Unterprogrammen](#)

Funktion

Das Statement `CALL FILE` dient dazu, ein nicht in Natural geschriebenes Programm aufzurufen, das einen Datensatz von einer Nicht-Adabas-Datei liest und diesen Datensatz an das aufrufende Natural-Programm zur Verarbeitung übergibt.

Einschränkung

Innerhalb einer `CALL FILE`-Schleife dürfen die Statements `AT BREAK`, `AT START OF DATA` und `AT END OF DATA` nicht verwendet werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A	A U N P I F B D T L C	ja	ja
<i>operand2</i>	S A G	A U N P I F B D T L C	ja	ja

Syntax-Element-Beschreibung:

<i>'program-name'</i>	Der Name des aufzurufenden Nicht-Natural-Programmes.
<i>operand1</i>	Kontrollfeld: <i>operand1</i> dient dazu, Kontrollinformationen zu liefern.
<i>operand2</i>	<i>operand2</i> definiert den Datensatz-Bereich. Das Format des zu lesenden Datensatzes kann mit Felddefinitionseinträgen (oder FILLER <i>nX</i>), die hinter dem ersten Feld des Datensatzes stehen, beschrieben werden. Die Felder, die dazu dienen, das Format des Datensatzes zu definieren, brauchen im Natural-Programm nicht vorher definiert werden. Dadurch ist gewährleistet, dass Natural die Felder benachbarten Speicherplätzen zuordnet.
<i>statement ...</i>	Das Statement CALL FILE initiiert eine Verarbeitungsschleife, die mit einem ESCAPE- oder STOP-Statement beendet werden muss. Um die Schleife in Abhängigkeit von verschiedenen Bedingungen zu beenden, können Sie mehrere ESCAPE-Statements verwenden.
END-FILE	Ein END-FILE-Statement muss benutzt werden, um die Verarbeitungsschleife zu schließen.

Beispiel

Aufrufendes Programm:

```
** Example 'CFIEX1': CALL FILE
*****
DEFINE DATA LOCAL
1 #CONTROL (A3)
1 #RECORD
  2 #A      (A10)
  2 #B      (N3.2)
  2 #FILL1  (A3)
```

```

2 #C      (P3.1)
END-DEFINE
*
CALL FILE 'USER1' #CONTROL #RECORD
  IF #CONTROL = 'END'
    ESCAPE BOTTOM
  END-IF
END-FILE
/*****
/* ... PROCESS RECORD ...
/*****
END

```

Die Byte-Belegung des vom aufgerufenen Programm an das Natural-Programm übergebenen Datensatzes sieht folgendermaßen aus:

```

CONTROL #A      #B  FILLER #C
(A3)   (A10)   (N3.2) 3X  (P3.1)

xxx xxxxxxxxxxx xxxxx   xxx   xxx

```

Aufgerufenes COBOL-Programm:

```

ID DIVISION.
PROGRAM-ID. USER1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT USRFILE ASSIGN UT-S-FILEUSR.
DATA DIVISION.
FILE SECTION.
FD  USRFILE RECORDING F LABEL RECORD OMITTED
    DATA RECORD DATA-IN.
01  DATA-IN          PIC X(80).
LINKAGE SECTION.
01  CONTROL-FIELD    PIC XXX.
01  RECORD-IN        PIC X(21).
PROCEDURE DIVISION USING CONTROL-FIELD RECORD-IN.
BEGIN.
    GO TO FILE-OPEN.
FILE-OPEN.
    OPEN INPUT USRFILE
    MOVE SPACES TO CONTROL-FIELD.
    ALTER BEGIN TO PROCEED TO FILE-READ.
FILE-READ.
    READ USRFILE INTO RECORD-IN
    AT END
    MOVE 'END' TO CONTROL-FIELD
    CLOSE USRFILE

```

ALTER BEGIN TO PROCEED TO FILE-OPEN.
GOBACK.

16 CALL LOOP

▪ Funktion	100
▪ Einschränkung	100
▪ Syntax-Beschreibung	101
▪ Beispiel	101

Structured Mode-Syntax

```
CALL LOOP operand1 [operand2] ...40
    statement ...
END-LOOP
```

Reporting Mode-Syntax

```
CALL LOOP operand1 [operand2] ...40
    statement ...
[LOOP]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [CALL](#) | [CALL FILE](#) | [CALLNAT](#) | [DEFINE SUBROUTINE](#) | [ESCAPE](#) | [FETCH](#) | [PERFORM](#)

Gehört zur Funktionsgruppe: [Aufrufen von Programmen und Unterprogrammen](#)

Funktion

Das Statement `CALL LOOP` dient dazu, eine Verarbeitungsschleife zu generieren, die den Aufruf eines Nicht-Natural-Programms beinhaltet.

Im Gegensatz zum `CALL`-Statement erzeugt das `CALL LOOP`-Statement eine Verarbeitungsschleife, die dazu dient, das Nicht-Natural-Programm wiederholt aufzurufen. Zu der `CALL`-Verarbeitung siehe `CALL`-Statement.

Einschränkung

Innerhalb einer `CALL LOOP`-Verarbeitungsschleife dürfen die Statements `AT BREAK`, `AT START OF DATA` und `AT END OF DATA` nicht verwendet werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	ja	nein
<i>operand2</i>	C S A G	A U N P I F B D T L C	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	Der Name des aufgerufenen Nicht-Natural-Programms (<i>operand1</i>) kann entweder als Konstante angegeben werden oder – falls je nach Programmlogik verschiedene Programme aufgerufen werden sollen – als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname muss linksbündig in der Variablen stehen.
<i>operand2</i>	Mit dem CALL LOOP-Statement können Sie bis zu 40 Parameter angeben. Der Aufbau der Parameterliste entspricht der für das CALL-Statement. In der Parameterliste verwendete Felder können schon vorher definiert werden oder erst im CALL LOOP-Statement selbst.
<i>statement ...</i>	Die mit CALL LOOP initiierte Verarbeitungsschleife muss mit einem ESCAPE-Statement beendet werden.
END-LOOP	Ein END-LOOP-Statement muss benutzt werden, um die Verarbeitungsschleife zu schließen.

Beispiel

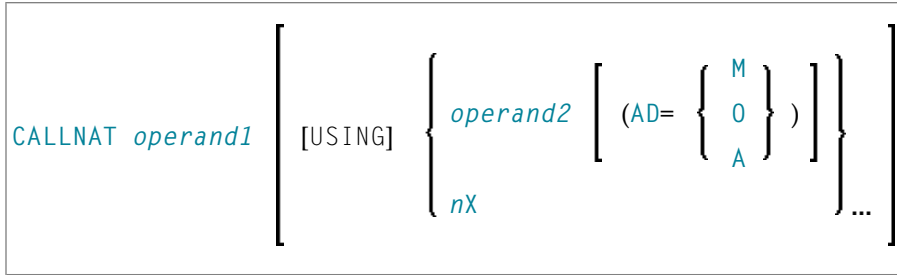
```

DEFINE DATA LOCAL
1 PARAMETER1 (A10)
END-DEFINE
CALL LOOP 'ABC' PARAMETER1
  IF PARAMETER1 = 'END'
    ESCAPE BOTTOM
  END-IF
END-LOOP
END

```


17 CALLNAT

▪ Funktion	104
▪ Syntax-Beschreibung	105
▪ Übertragung von Parametern mit dynamischen Variablen	107
▪ Beispiele	108



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `CALL` | `CALL FILE` | `CALL LOOP` | `DEFINE SUBROUTINE` | `ESCAPE` | `FETCH` | `PERFORM`

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Das Statement `CALLNAT` dient dazu, ein Natural-Subprogramm zur Ausführung aufzurufen. Ein Natural-Subprogramm kann nur über ein `CALLNAT`-Statement aufgerufen werden; es kann nicht selbständig ausgeführt werden.

Wenn das `CALLNAT`-Statement ausgeführt wird, wird die Ausführung des aufrufenden Objekts (d.h. des Objekts, das das `CALLNAT`-Statement enthält) unterbrochen und das aufgerufene Subprogramm ausgeführt. Die Ausführung des Subprogramms dauert an, bis entweder sein `END`-Statement erreicht ist oder die Verarbeitung des Subprogramms durch die Ausführung eines `ESCAPE ROUTINE`-Statements gestoppt wird. In beiden Fällen wird dann die Verarbeitung des aufrufenden Objekts mit dem nächsten Statement nach dem `CALLNAT`-Statement fortgesetzt.



Anmerkungen:

1. Ein Subprogramm kann wiederum andere Subprogramme aufrufen.
2. Ein Subprogramm hat keinen Zugriff auf die von dem aufrufenden Objekt benutzte Global Data Area. Wenn ein Subprogramm wiederum eine Subroutine oder Helproutine aufruft, kann es seine eigene Global Data Area erstellen, und diese zusammen mit der Subroutine/Helproutine gemeinsam benutzen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S			A												ja	nein	
<i>operand2</i>	C	S	A	G	A	U	N	P	I	F	B	D	T	L	C	G	O	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Subprogramm-Name:</p> <p>Als <i>operand1</i> geben Sie den Namen des Subprogramms an, das aufgerufen werden soll. Dieser kann entweder als 1 bis 8 Zeichen lange Konstante angegeben werden oder – falls je nach Programmlogik unterschiedliche Subprogramme aufgerufen werden sollen – als alphanumerische Variable mit Länge 1 bis 8.</p> <p>Der Name des Subprogramms darf ein Und-Zeichen (&) enthalten; zur Ausführungszeit wird dieses Zeichen durch den aus einem Zeichen bestehenden Code ersetzt, der dem aktuellen Wert der Systemvariablen *LANGUAGE entspricht. Dadurch ist es beispielsweise möglich, je nachdem in welcher Sprache eine Eingabe gemacht wird, zur Verarbeitung der Eingabe unterschiedliche Subprogramme aufzurufen.</p>
<i>operand2</i>	<p>Parameter:</p> <p>Werden Parameter an das Subprogramm übergeben, muss die Struktur der Parameterliste in einem <code>DEFINE DATA PARAMETER</code>-Statement definiert werden. Die mit dem CALLNAT-Statement angegebenen Parameter sind die einzigen Daten, die dem Subprogramm vom aufrufenden Objekt zur Verfügung stehen.</p> <p>Standardmäßig erfolgt die Übergabe der Parameter durch Referenzierung („By Reference“), d.h. die Daten werden über Adress-Parameter übergeben, die Parameterwerte selbst werden nicht übertragen. Es besteht aber auch die Möglichkeit, die Parameterwerte selbst zu übergeben. Hierzu definieren Sie die betreffenden Felder im <code>DEFINE DATA PARAMETER</code>-Statement des Subprogramms mit der Option <code>BY VALUE</code> bzw. <code>BY VALUE RESULT</code> (siehe parameter-data-definition in der Beschreibung des <code>DEFINE DATA</code>-Statements).</p> <ul style="list-style-type: none"> ■ Für die Parameterübergabe durch Referenzierung („By Reference“) gilt: Reihenfolge, Format und Länge der Parameter im aufrufenden Objekt müssen genau den Angaben im <code>DEFINE DATA PARAMETER</code>-Statement des Subprogramms entsprechen. Die Namen der Variablen im aufrufenden Objekt und im aufgerufenen Subprogramm können unterschiedlich sein. ■ Für die Übergabe der Parameterwerte selbst („By Value“) gilt: die Reihenfolge der Parameter im aufrufenden Objekt muss der Reihenfolge im <code>DEFINE DATA PARAMETER</code>-Statement des Subprogramms entsprechen. Formate und Längen der Variablen im aufrufenden Objekt und im Subprogramm können unterschiedlich sein, müssen aber datenübertragungskompatibel

sein (vgl. entsprechende Tabelle im Abschnitt *Regeln für arithmetische Operationen, Datenübertragung im Leitfaden zur Programmierung*). Die Namen der Variablen im aufrufenden Objekt und im aufgerufenen Subprogramm können unterschiedlich sein. Um Parameterwerte, die im Subprogramm verändert wurden, an das aufrufende Objekt zurückgeben zu können, müssen Sie die betreffenden Felder mit **BY VALUE RESULT** definieren.

Mit **BY VALUE** (ohne **RESULT**) ist es nicht möglich, veränderte Parameterwerte an das aufrufende Objekt zurückzugeben (unabhängig von der **AD**-Parameter-Angabe; vgl. unten).

Anmerkung: Intern wird bei **BY VALUE** eine Kopie der Parameterwerte erzeugt. Das Subprogramm greift auf diese Kopie zu und kann sie modifizieren, was aber keinen Einfluss auf die Originalparameterwerte im aufrufenden Objekt hat. Bei **BY VALUE RESULT** wird ebenfalls eine Kopie erzeugt, aber nach Beendigung des Subprogramms überschreiben die (modifizierten) Werte der Kopie die Originalparameterwerte.

Für beide Arten der Parameterübergabe sind folgende Punkte zu beachten:

- Wenn als *operand2* eine Gruppe angegeben wird, werden die einzelnen in der Gruppe enthaltenen Felder an das Subprogramm übergeben; d.h. für jedes dieser Felder muss in der Parameter Data Area des Subprogramms ein entsprechendes Feld definiert werden.
- Eine Gruppe darf in der Parameter Data Area eines Subprogramms nur innerhalb eines **REDEFINE**-Blocks redefiniert werden.
- Bei der Übergabe eines Arrays muss die Anzahl seiner Dimensionen und Ausprägungen in der Parameter Data Area des Subprogramms denen in der **CALLNAT**-Parameterliste entsprechen.

Anmerkung: Wenn mehrere Ausprägungen eines Arrays, das als Teil einer indizierten Gruppe definiert ist, mit dem **CALLNAT**-Statement übergeben werden, dürfen die entsprechenden Felder in der Parameter Data Area des Subprogramms nicht redefiniert werden, da sonst die falschen Adressen übergeben werden.

Wenn die Option **PCHECK** des Systemkommandos **COMPOPT** auf **ON** gesetzt ist, überprüft der Compiler Anzahl, Format, Länge und Array-Indexgrenzen der Parameter, die in einem **CALLNAT**-Statement angegeben sind. Die Funktion **OPTIONAL** des **DEFINE DATA PARAMETER**-Statements wird bei der Parameter-Prüfung mit berücksichtigt.

AD=	Attribut-Definition:	
	Wenn <i>operand2</i> eine Variable ist, können Sie sie wie folgt kennzeichnen:	
	AD=O	Nicht modifizierbar, siehe Session-Parameter AD=O . Anmerkung: Intern wird AD=O genauso verarbeitet wie BY VALUE (siehe parameter-data-definition in der Beschreibung des DEFINE DATA -Statements).
AD=M	Modifizierbar, siehe Session-Parameter AD=M . Dies ist die Standardeinstellung.	

	AD=A	Nur für Eingabe, siehe Session-Parameter AD=A.
	Wenn <i>operand2</i> eine Konstante ist, kann der Session-Parameter AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=0.	
nX	<p>Mit der Notation <i>nX</i> können Sie angeben, dass die nächsten <i>n</i> Parameter übersprungen werden sollen (z.B. 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); dies bedeutet, dass für die nächsten <i>n</i> Parameter keine Werte an das Subprogramm übergeben werden. Der mögliche Wertebereich für <i>n</i> ist 1 – 4096.</p> <p>Ein zu überspringender Parameter muss mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Subprogramms definiert werden. OPTIONAL bedeutet, dass ein Wert vom aufrufenden Objekt an solch einen Parameter übergeben werden kann, aber nicht unbedingt übergeben werden muss.</p>	

Übertragung von Parametern mit dynamischen Variablen

Dynamische Variablen können als Parameter an ein aufgerufenes Programmobjekt (CALLNAT, PERFORM) übergeben werden.

Eine Übergabe durch Referenzierung („Call By Reference“) ist möglich, weil dynamische Variablen einen zusammenhängenden Wertebereich darstellen.

Bei einer Übergabe der Parameterwerte selbst („Call By Value“) wird die Variablen-Definition des Aufrufenden als Ausgangsoperand und die Parameter-Definition als Zieloperand zugewiesen. Ein „Call By Value Result“ bewirkt des weiteren eine Umkehrung der Zuordnung. Bei „Call By Reference“ müssen beide Definitionen **DYNAMIC** sein. Wenn nur eine von ihnen **DYNAMIC** ist, wird ein Laufzeitfehler hervorgerufen. Bei „Call By Value (Result)“ sind alle Kombinationen möglich.

Die folgende Tabelle veranschaulicht die gültigen Kombinationen statisch und dynamisch definierter Variablen des Aufrufenden und statisch und dynamisch definierter Parameter hinsichtlich der Übertragung von Parametern.

Call By Reference

<i>operand2</i> vom Aufrufenden	Parameter-Definition	
	Statisch	Dynamisch
Statisch	ja	nein
Dynamisch	nein	ja

Die Formate der dynamischen Variablen A oder B müssen miteinander übereinstimmen.

Call by Value (Result)

<i>operand2</i> vom Aufrufenden	Parameter-Definition	
	Statisch	Dynamisch
Statisch	ja	ja
Dynamisch	ja	ja



Anmerkung: Bei statischen/dynamischen oder dynamischen/statischen Definitionen kann es vorkommen, dass ein Wert nach den Datenübertragungsregeln der entsprechenden Zuweisungen abgeschnitten wird.

Beispiele

- [Beispiel 1](#)
- [Beispiel 2](#)

Beispiel 1

Aufrufendes Programm:

```

** Example 'CNTEX1': CALLNAT
*****
DEFINE DATA LOCAL
1 #FIELD1 (N6)
1 #FIELD2 (A20)
1 #FIELD3 (A10)
END-DEFINE
*
CALLNAT 'CNTEX1N' #FIELD1 (AD=M) #FIELD2 (AD=0) #FIELD3 'P4 TEXT'
*
WRITE '=' #FIELD1 '=' #FIELD2 '=' #FIELD3
*
END

```


Aufgerufenes Subprogramm CNTEX1N:

```

** Example 'CNTEX1N': CALLNAT (called by CNTEX1)
*****
DEFINE DATA PARAMETER
1 #FIELDA (N6)
1 #FIELDB (A20)
1 #FIELD C (A10)
1 #FIELD D (A7)
END-DEFINE
*
*
#FIELD A := 4711
*
#FIELD B := 'HALLO'
*
#FIELD C := 'ABC'
*
WRITE '=' #FIELD A '=' #FIELD B '=' #FIELD C '=' #FIELD D
*
END

```

Beispiel 2**Aufrufendes Programm:**

```

** Example 'CNTEX2': CALLNAT
*****
DEFINE DATA LOCAL
1 #ARRAY1 (N4/1:10,1:10)
1 #NUM (N2)
END-DEFINE
*
*
CALLNAT 'CNTEX2N' #ARRAY1 (2:5,*)
*
FOR #NUM 1 TO 10
WRITE #NUM #ARRAY1(#NUM,1:10)
END-FOR
*
END

```

Aufgerufenes Subprogramm CNTEX2N:

```
** Example 'CNTEX2N': CALLNAT (called by CNTEX2)
*****
DEFINE DATA
PARAMETER
1 #ARRAY (N4/1:4,1:10)
LOCAL
1 I      (I2)
END-DEFINE
*
*
FOR I 1 10
  #ARRAY(1,I) := I
  #ARRAY(2,I) := 100 + I
  #ARRAY(3,I) := 200 + I
  #ARRAY(4,I) := 300 + I
END-FOR
*
END
```

18 CLOSE CONVERSATION

- Funktion 112
- Syntax-Beschreibung 112
- Weitere Informationen und Beispiele 113



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `DEFINE DATA CONTEXT` | `OPEN CONVERSATION`

Gehört zur Funktionsgruppe: *Natural Remote Procedure Call*

Funktion

Das Statement `CLOSE CONVERSATION` wird im Zusammenhang mit Natural Remote Procedure Call (RPC) verwendet. Es ermöglicht dem Client, eine Konversation zu schließen. Sie können die aktuelle Konversation, eine andere offene Konversation oder alle offenen Konversationen schließen.



Anmerkung: Ein Logon in eine andere Library bewirkt kein automatisches Schließen von Konversationen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A	I	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	Zu schließende Konversation: Um eine bestimmte offene Konversation zu schließen, geben Sie Ihre ID als <i>operand1</i> an. <i>operand1</i> muss eine Variable mit Format/Länge I4 sein.
*CONVID	Um die aktuelle Konversation zu schließen, geben Sie *CONVID an. Die ID der aktuellen Konversation wird bestimmt durch den Wert der Systemvariablen *CONVID.
ALL	Um alle offenen Konversationen zu schließen, geben Sie ALL an.

Weitere Informationen und Beispiele

Siehe folgende Abschnitte in der *Natural Remote Procedure Call (RPC)*-Dokumentation:

- *Natural RPC Operation in Conversational Mode*
- *Using a Conversational RPC*

19

CLOSE DIALOG

▪ Funktion	116
▪ Syntax-Beschreibung	116
▪ Weitere Informationen und Beispiele	117

```
CLOSE DIALOG [USING] [DIALOG-ID] { operand1
                                *DIALOG-ID }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [OPEN DIALOG](#) | [PROCESS GUI](#) | [SEND EVENT](#)

Gehört zur Funktionsgruppe: *Ereignisgesteuerte Programmierung*

Funktion

Das Statement `CLOSE DIALOG` dient dazu, einen Dialog dynamisch zu schließen.



Anmerkung: Wenn ein modaler Dialog ein Child in einer Hierarchie von Dialogen ist, sollte der modale Dialog seine(n) Parent-Dialog(e) nicht schließen, weil dies zu einem Stillstand im Funktionsablauf führen würde.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S	I	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	Identifizier: <i>operand1</i> ist der Identifizier des zu schließenden Dialogs.
*DIALOG-ID	Um den aktuellen Dialog zu schließen, geben Sie die Systemvariable <i>*DIALOG-ID</i> an, die die ID der aktuellen Instanz eines Dialogs enthält.

Weitere Informationen und Beispiele

Siehe Abschnitt *Event-Driven Programming Techniques* im *Leitfaden zur Programmierung*.

20 CLOSE PRINTER

▪ Funktion	120
▪ Syntax-Beschreibung	120
▪ Beispiel	121

<code>CLOSE PRINTER</code> { <i>(logical-printer-name)</i> } { <i>(printer-number)</i> }

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `AT END OF PAGE` | `AT TOP OF PAGE` | `DEFINE PRINTER` | `DISPLAY` | `EJECT` | `FORMAT` | `NEWPAGE` | `PRINT` | `SKIP` | `SUSPEND IDENTICAL SUPPRESS` | `WRITE` | `WRITE TITLE` | `WRITE TRAILER`

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `CLOSE PRINTER` dient dazu, einen bestimmten Drucker zu schließen. Mit dem `CLOSE PRINTER`-Statement können Sie explizit in einem Programm angeben, dass ein Drucker geschlossen werden soll.

In folgenden Situationen wird ein Drucker automatisch geschlossen:

- wenn ein `DEFINE PRINTER`-Statement, in dem derselbe Drucker wieder definiert ist, ausgeführt wird;
- bei Erreichen des Kommando-Modus.

Syntax-Beschreibung

<i>logical-printer-name</i>	Logischer Drucker-Name: Mit <i>logical-printer-name</i> geben Sie den Drucker an, der geschlossen werden soll. Der Name entspricht den Angaben in dem <code>DEFINE PRINTER</code> -Statement, in dem Sie den betreffenden Drucker definiert haben. Für den <i>logical-printer-name</i> gelten die gleichen Namenskonventionen wie für Benutzervariablen; siehe <i>Namenskonventionen für Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i> .
<i>printer-number</i>	Drucker-Nummer: Außer dem <i>logical-printer-name</i> können Sie auch die <i>printer-number</i> nehmen, um anzugeben, welcher Drucker geschlossen werden soll.

Die *printer-number* kann eine Zahl von 0 bis 31 sein. Diese Zahl kann auch in einem **DISPLAY-**, **WRITE-** oder **DEFINE PRINTER**-Statement benutzt werden.

Die *printer-number* 0 gibt den Hardcopy-Drucker an.

Beispiel

```

** Example 'CLPEX1': CLOSE PRINTER
*****
DEFINE DATA LOCAL
1 EMP-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #I-NAME (A20)
END-DEFINE
*
DEFINE PRINTER (PRT01=1)
*
REPEAT
  INPUT 'SELECT PERSON' #I-NAME
  IF #I-NAME = ' '
    STOP
  END-IF
  FIND EMP-VIEW WITH NAME = #I-NAME
  WRITE (PRT01) 'NAME           :' NAME ',' FIRST-NAME
              /      'PERSONNEL-ID :' PERSONNEL-ID
              /      'BIRTH           :' BIRTH (EM=YYYY-MM-DD)
  END-FIND
/*
  CLOSE PRINTER (PRT01)
/*
END-REPEAT
END

```


21

CLOSE WORK FILE

▪ Funktion	124
▪ Syntax-Beschreibung	124
▪ Beispiel	124

CLOSE WORK [FILE] <i>work-file-number</i>

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `DEFINE WORK FILE` | `READ WORK FILE` | `WRITE WORK FILE`

Gehört zur Funktionsgruppe: *Kontrolle von Arbeitsdateien / PC-Dateien*

Funktion

Das Statement `CLOSE WORK FILE` dient dazu, eine bestimmte Arbeitsdatei zu schließen. Es erlaubt Ihnen, in einem Programm explizit anzugeben, dass eine Arbeitsdatei geschlossen werden soll.

Eine Arbeitsdatei schließt sich auch automatisch,

- wenn der Kommando-Modus erreicht ist;
- wenn eine Dateiende-Bedingung bei Ausführung eines `READ WORK FILE`-Statements auftritt;
- bevor ein `DEFINE WORK FILE`-Statement ausgeführt wird, das der betreffenden Arbeitsdateinummer eine andere Datei zuweist.

Syntax-Beschreibung

<i>work-file-number</i>	Als <i>work-file-number</i> geben Sie die Nummer der Arbeitsdatei an (mit der sie für Natural definiert ist), die geschlossen werden soll.
-------------------------	--

Beispiel

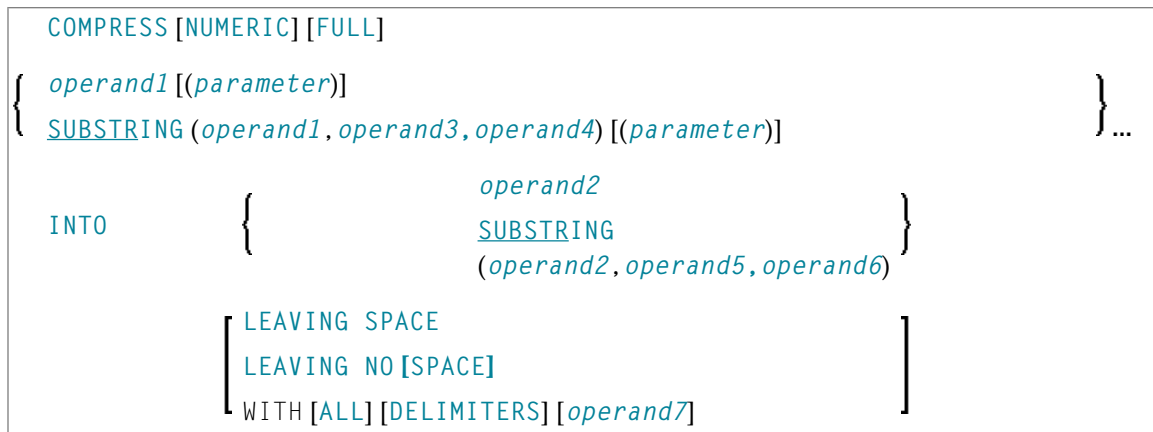
```
** Example 'CWFEX1': CLOSE WORK FILE
*****
DEFINE DATA LOCAL
1 W-DAT   (A20)
1 REC-NUM (N3)
1 I      (P3)
END-DEFINE
*
REPEAT
```



```
READ WORK FILE 1 ONCE W-DAT /* READ MASTER RECORD
/*
AT END OF FILE
  ESCAPE BOTTOM
END-ENDFILE
INPUT 'PROCESSING FILE' W-DAT (AD=0)
  / 'ENTER RECORDNUMBER TO DISPLAY' REC-NUM
IF REC-NUM = 0
  STOP
END-IF
  FOR I = 1 TO REC-NUM
  /*
  READ WORK FILE 1 ONCE W-DAT
  /*
  AT END OF FILE
    WRITE 'RECORD-NUMBER TOO HIGH, LAST RECORD IS'
    ESCAPE BOTTOM
  END-ENDFILE
END-FOR
I := I - 1
WRITE 'RECORD' I ':' W-DAT
/*
CLOSE WORK FILE 1
/*
END-REPEAT
END
```

22 COMPRESS

▪ Funktion	128
▪ Syntax-Beschreibung	128
▪ Verarbeitung	132
▪ Beispiele	132



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: *ASSIGN* | *COMPUTE* | *EXAMINE* | *MOVE* | *MOVE ALL* | *SEPARATE*

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das Statement *COMPRESS* dient dazu, den Inhalt eines oder mehrerer Operanden in ein einziges alphanumerisches Feld zu übertragen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A G N	A U N P I F B D T G O	ja	nein
<i>operand2</i>	S	A U B	ja	ja
<i>operand3</i>	C S	N P I B*	ja	nein
<i>operand4</i>	C S	N P I B*	ja	nein
<i>operand5</i>	C S	N P I B*	ja	nein
<i>operand6</i>	C S	N P I B*	ja	nein
<i>operand7</i>	C S	A U B	ja	nein

* Format B von *operand3*, *operand4*, *operand5* und *operand6* kann nur mit einer Länge von kleiner oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

NUMERIC	<p>Diese Option bestimmt, wie Vorzeichen und Dezimalzeichen behandelt werden:</p> <p>Ohne NUMERIC werden Dezimalkommas und Vorzeichen bei numerischen Ausgangswerten unterdrückt, bevor die Werte in das Zielfeld übertragen werden. Zum Beispiel:</p> <pre>COMPRESS -123 1.23 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: 123*123</pre> <p>Mit NUMERIC werden Dezimalkommas und Vorzeichen aus numerischen Ausgangswerten ebenfalls mit in das Zielfeld übertragen.</p> <p>Für Gleitkomma-Ausgangswerte werden Dezimalzeichen und Vorzeichen übertragen, ungeachtet der Tatsache, ob NUMERIC angegeben wurde oder nicht.</p> <p>Beispiel 1:</p> <pre>COMPRESS NUMERIC -123 1.23 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: -123*1.23</pre> <p>Beispiel 2:</p> <pre>COMPRESS NUMERIC 'ABC' -0056.00 -0056.10 -0056.01 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: ABC*-56*-56.1*-56.01</pre> <p>Beispiel 3:</p> <pre>COMPRESS NUMERIC FULL 'ABC' -0056.00 -0056.10 -0056.01 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: ABC*-0056.00*-0056.10*-0056.01</pre>
FULL	<p>Ohne FULL werden folgende Zeichen aus den Ausgangsfeldern entfernt, bevor die Werte übertragen werden:</p> <ul style="list-style-type: none"> ■ vorangestellte Nullen vor dem Komma oder Dezimalpunkt für Felder vom Format N, P oder I, ■ nachfolgende Nullen nach dem Komma oder Dezimalpunkt für Felder vom Format N, P, ■ nachfolgende Leerzeichen für Felder vom Format A ■ und führende binäre Nullen für Felder vom Format B

	<p>Enthält ein numerisches Ausgangsfeld lauter Nullen, wird eine Null (0) übertragen. Zum Beispiel:</p> <pre>COMPRESS 'ABC ' 001 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: ABC*1</pre> <p>Mit FULL werden die Werte der Ausgangsfelder in ihrer tatsächlichen Länge — d.h. inklusive vorangestellter Nullen und nachfolgender Leerzeichen — ins Zielfeld übertragen. Mit anderen Worten werden die folgenden Zeichen wie eingegeben angezeigt:</p> <ul style="list-style-type: none"> ■ vorangestellte Nullen vor dem Komma oder Dezimalpunkt für Felder vom Format N, P oder I, ■ nachfolgende Nullen nach dem Komma oder Dezimalpunkt für Felder vom Format N, P, ■ nachfolgende Leerzeichen für Felder vom Format A ■ und führende binäre Nullen für Felder vom Format B <p>Beispiel:</p> <pre>COMPRESS FULL 'ABC ' 001 INTO #TARGET WITH DELIMITER '*' Content of #TARGET is: ABC *001</pre>		
<i>operand1</i>	<p>Ausgangsfelder:</p> <p>Mit <i>operand1</i> geben Sie die Felder an, deren Inhalt übertragen werden soll.</p> <p>Anmerkung: Wenn <i>operand1</i> nicht Format A oder B hat, wird sein Inhalt in alphanumerische Darstellung konvertiert, bevor er übertragen wird. Wenn erforderlich, wird die alphanumerische Darstellung abgeschnitten.</p> <p>Wenn <i>operand1</i> eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übertragen, aber nicht die Datumskomponente.</p>		
<i>operand2</i>	<p>Zielfeld</p> <p>Mit <i>operand2</i> geben Sie das Feld an, das die Werte aus den Ausgangsfeldern aufnehmen soll.</p> <p>Wenn das Zielfeld vom Format U (Unicode) hat, und wenn es sich um ein Ausgangsfeld mit Format B handelt, muss die Länge des sendenden Binärfeldes gleich sein.</p>		
LEAVING SPACE	<p>Wenn Sie das COMPRESS-Statement ohne weitere Optionen verwenden oder LEAVING SPACE (gilt auch standardmäßig) angeben, so werden die Werte im Zielfeld jeweils durch ein Leerzeichen voneinander getrennt.</p>		
LEAVING NO SPACE	<p>Wenn Sie LEAVING NO SPACE angeben, werden die Werte im Zielfeld weder durch ein Leerzeichen noch durch ein anderes Zeichen voneinander getrennt.</p>		
<i>parameter</i>	<p>Als <i>parameter</i> können Sie die Option PM=I oder den Session-Parameter DF angeben:</p> <table border="1" data-bbox="289 1793 1385 1873"> <tr> <td data-bbox="289 1793 808 1873">PM=I</td> <td data-bbox="808 1793 1385 1873">Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft,</td> </tr> </table>	PM=I	Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft,
PM=I	Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft,		

		<p>können Sie die Option <code>PM=I</code> angeben, um den Wert von <i>operand1</i> invers (d.h. von rechts nach links) in <i>operand2</i> zu übertragen.</p> <p>Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld <code>#B</code> den Inhalt <code>ZYXABC</code>:</p> <pre>MOVE 'XYZ' TO #A COMPRESS #A (PM=I) 'ABC' INTO #B LEAVING NO SPACE</pre> <p>Nachfolgende Leerzeichen in <i>operand1</i> werden entfernt (außer wenn <code>FULL</code> angegeben ist), dann wird der Wert Zeichen für Zeichen umgedreht und anschließend in <i>operand2</i> übertragen.</p>
	DF	<p>Wenn <i>operand1</i> eine Datumsvariable ist, können Sie den Session-Parameter <code>DF</code> als <i>parameter</i> für diese Variable angeben.</p>
SUBSTRING <i>(operand1,</i> <i>operand3,</i> <i>operand4)</i>		<p>Wenn <i>operand1</i> alphanumerisches (A), Unicode (U) oder binäres Format (B) hat, können Sie die <code>SUBSTRING</code>-Option verwenden, um nur einen Teil des Ausgangsfeldes zu übertragen. Hinter dem Feldnamen (<i>operand1</i>) geben Sie zuerst die Startposition (<i>operand3</i>) und dann die Länge (<i>operand4</i>) des zu übertragenden Feldabschnitts ein.</p>
INTO SUBSTRING <i>(operand2,</i> <i>operand5,</i> <i>operand6)</i>		<p>Sie können die <code>SUBSTRING</code>-Option auch in der <code>INTO</code>-Klausel verwenden, um die Ausgangswerte in einen bestimmten Teil des Zielfeldes zu übertragen.</p> <p>Die Verwendung der <code>SUBSTRING</code>-Option in einem <code>COMPRESS</code>-Statement entspricht in beiden Fällen der in einem <code>MOVE</code>-Statement. Einzelheiten zur <code>SUBSTRING</code>-Option finden Sie beim MOVE-Statement.</p>
WITH DELIMITERS		<p>Möchten Sie, dass die Werte im Zielfeld jeweils durch ein bestimmtes Zeichen voneinander getrennt werden, dann verwenden Sie die <code>DELIMITERS</code>-Option.</p> <p>Wenn Sie <code>WITH DELIMITERS</code> ohne <i>operand7</i> angeben, werden die Werte durch das (mit der Session-Parameter <code>ID</code> definierte) Input-Delimiter-Zeichen voneinander getrennt.</p>
WITH DELIMITERS <i>operand7</i>		<p>Wenn Sie <code>WITH DELIMITERS</code> <i>operand7</i> angeben, werden die Werte durch das mit <i>operand7</i> angegebene Zeichen voneinander getrennt. <i>operand7</i> muss ein einzelnes Zeichen sein.</p> <p>Wenn <i>operand7</i> eine Variable ist, muss sie Format/Länge <code>A1</code> oder <code>B1</code> haben.</p> <p>Wenn das Zielfeld vom Format <code>A</code> oder <code>B</code> ist, muss das Format bzw. die Länge des Trennzeichens (<code>A1</code>), (<code>B1</code>) oder (<code>U1</code>) sein.</p> <p>Wenn das Zielfeld vom Format <code>U</code> (Unicode) ist, muss das Format bzw. die Länge des Trennzeichens (<code>A1</code>), (<code>B2</code>) oder (<code>U1</code>) sein.</p>

WITH ALL	<p>Ohne ALL werden im Zielfeld Delimiter-Zeichen nur zwischen tatsächlich übertragenen Werten gesetzt. Zum Beispiel:</p> <pre>COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH DELIMITERS '*' Content of #TARGET is: A*C</pre> <p>Mit ALL wird im Zielfeld auch für jeden (nicht übertragenen) Leerwert ein Delimiter-Zeichen gesetzt. Das heisst, die Anzahl der Delimiter-Zeichen im Zielfeld ist gleich der Anzahl der Ausgangsfelder minus 1. Dies kann z.B. sinnvoll sein, wenn der Inhalt des Zielfeldes mit einem SEPARATE-Statement anschließend wieder aufgeteilt werden soll. Zum Beispiel:</p> <pre>COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH ALL DELIMITERS '*' Content of #TARGET is: A**C*</pre>
-----------------	---

Verarbeitung

Ein Zielfeld vom Format B wird wie ein Zielfeld vom Format A behandelt.

Die COMPRESS-Operation wird beendet, sobald entweder alle Operanden übertragen sind oder das Zielfeld (*operand2*) voll ist.

Ist das Zielfeld länger als alle übertragenen Werte zusammen, so werden die verbleibenden Stellen von *operand2* mit Leerzeichen gefüllt. Ist das Zielfeld kürzer, wird der Wert abgeschnitten.

Falls *operand2* eine dynamische Variable ist, wird die COMPRESS-Operation beendet, wenn alle Ausgangs-Operanden verarbeitet worden sind. Es werden keine Zeichen abgeschnitten. Die Länge von *operand2* nach der COMPRESS-Operation entspricht dann der gemeinsamen Länge der Ausgangs-Operanden. Die aktuelle Länge einer dynamischen Variable kann durch die Systemvariable *LENGTH bestimmt werden.

Beispiele

- [Beispiel 1 — COMPRESS-Statement](#)
- [Beispiel 2 — COMPRESS-Statement mit LEAVING NO SPACE](#)

■ Beispiel 3 — COMPRESS-Statement mit WITH DELIMITER

Beispiel 1 — COMPRESS-Statement

```

** Example 'CMPEX1': COMPRESS
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
*
1 #COMPRESSED-NAME (A20)
END-DEFINE
*
LIMIT 4
READ EMPLOY-VIEW BY NAME
  COMPRESS FIRST-NAME MIDDLE-I NAME INTO #COMPRESSED-NAME
  DISPLAY NOTITLE
    FIRST-NAME MIDDLE-I NAME 5X #COMPRESSED-NAME
END-READ
*
END

```

Ausgabe des Programms CMPEX1:

FIRST-NAME	MIDDLE-I	NAME	#COMPRESSED-NAME
KEPA		ABELLAN	KEPA ABELLAN
ROBERT	W	ACHIESON	ROBERT W ACHIESON
SIMONE		ADAM	SIMONE ADAM
JEFF	H	ADKINSON	JEFF H ADKINSON

Beispiel 2 — COMPRESS-Statement mit LEAVING NO SPACE

```

** Example 'CMPEX2': COMPRESS (with LEAVING NO SPACE)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1)
  2 SALARY (1)
*
1 #CCSALARY (A20)
END-DEFINE
*

```

```

LIMIT 4
READ EMPL-VIEW BY NAME

  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY
    LEAVING NO SPACE
  DISPLAY NOTITLE
    NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
END-READ
*
END

```

Ausgabe des Programms CMPEX2:

NAME	CURRENCY CODE	ANNUAL SALARY	#CCSALARY
ABELLAN	PTA	1450000	PTA1450000
ACHIESON	UKL	11300	UKL11300
ADAM	FRA	159980	FRA159980
ADKINSON	USD	34500	USD34500

Beispiel 3 — COMPRESS-Statement mit WITH DELIMITER

```

** Example 'CMPEX3': COMPRESS (with delimiter)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1)
  2 SALARY (1)
*
1 #CCSALARY (A20)
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  COMPRESS CURR-CODE (1) SALARY (1) INTO #CCSALARY
    WITH DELIMITER '*'
  DISPLAY NOTITLE NAME CURR-CODE (1) SALARY (1) 5X #CCSALARY
END-READ
*
END

```

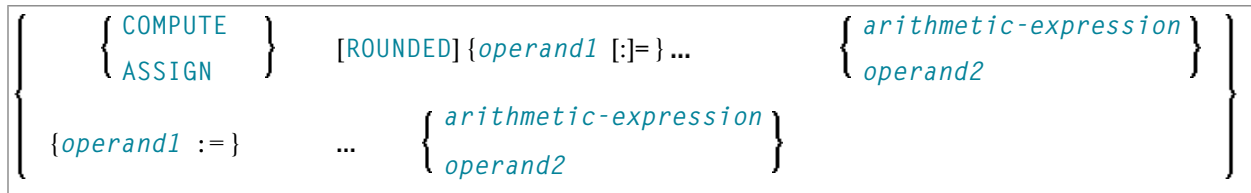
Ausgabe des Programms CMPEX3:

NAME	CURRENCY CODE	ANNUAL SALARY	#CCSALARY
ABELLAN	PTA	1450000	PTA*1450000
ACHIESON	UKL	11300	UKL*11300
ADAM	FRA	159980	FRA*159980
ADKINSON	USD	34500	USD*34500

23 COMPUTE

▪ Funktion	138
▪ Syntax-Beschreibung	140
▪ Ergebnisgenauigkeit einer Division	142
▪ SUBSTRING-Option	143
▪ Beispiele	143

Structured Mode-Syntax



Reporting Mode-Syntax



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das Statement COMPUTE dient zur Ausführung einer arithmetischen Operation sowie dazu, einem oder mehreren Feldern einen Wert zuzuweisen.

Ein COMPUTE-Statement mit mehreren Zieloperanden (*operand1*) ist identisch mit den entsprechenden einzelnen COMPUTE-Statements, wenn der Ausgangsoperand (*operand2*) kein arithmetischer Ausdruck ist.

```
#TARGET1 := #TARGET2 := #SOURCE
```

ist identisch mit

```
#TARGET1 := #SOURCE
#TARGET2 := #SOURCE
```

Beispiel:

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <3,0,9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
*
#INDEX :=      /* #INDEX is 3
#RESULT :=     /* #RESULT is 9
#ARRAY(#INDEX)
*
#INDEX := 2
*
#INDEX :=      /* #INDEX is 0
#ARRAY(3) :=   /* returns run time error NAT1316
#ARRAY(#INDEX)
END
```

Wenn der Ausgangsoperand ein arithmetischer Ausdruck ist, wird der Ausdruck ausgewertet und das Ergebnis in einer temporären Variablen abgelegt. Danach wird diese temporäre Variable den Zieloperanden zugeordnet.

```
#TARGET1 := #TARGET2 := #SOURCE1 + 1
is identical to
#TEMP := #SOURCE1 + 1
#TARGET1 := #TEMP
#TARGET2 := #TEMP
```

Beispiel:

```

DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <2, 0, 9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
*
#INDEX := /* #INDEX is 3
#RESULT := /* #RESULT is 3
#ARRAY(#INDEX) + 1
*
#INDEX := 2
*
#INDEX := /* #INDEX is 0
#ARRAY(3) := /* returns run time error NAT1316
#ARRAY(#INDEX)
END
    
```

Weitere Informationen siehe *Regeln für arithmetische Operationen* im Leitfaden zur Programmierung und dort insbesondere die folgenden Abschnitte:

- *Arithmetische Operationen mit Arrays*
- *Datenübertragung* (Informationen zur Kompatibilität der Datenübertragung und zu Regeln für die Datenübertragung)

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A M	A U N P I F B D T L C G O	ja	ja
<i>operand2</i>	C S A N E	A U N P I F B D T L C G O	ja	nein

Syntax-Element-Beschreibung:

COMPUTE ASSIGN [:]=	<p>Sie können das Statement in Kurzform angeben und den Statement-Namen COMPUTE (bzw. ASSIGN) weglassen.</p> <p>Wenn Sie im Structured Mode den Statement-Namen weglassen, müssen Sie vor das Gleichheitszeichen (=) einen Doppelpunkt (:) schreiben.</p> <p>Verwenden Sie die <code>ROUNDED</code>-Option, müssen Sie den Statement-Namen angeben.</p>
ROUNDED	<p>Wenn Sie das Schlüsselwort <code>ROUNDED</code> angeben, wird der Wert auf- bzw. abgerundet, bevor er <i>operand1</i> zugewiesen wird.</p> <p>Die für das Runden gültigen Regeln finden Sie im Abschnitt <i>Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten im Leitfaden zur Programmierung</i>.</p>
<i>operand1</i>	<p>Ergebnisfeld:</p> <p><i>operand1</i> nimmt das Ergebnis der arithmetischen Operation bzw. Zuweisung auf.</p> <p>Zur Genauigkeit des Ergebnisses siehe Abschnitt <i>Genauigkeit von Ergebnissen bei arithmetischen Operationen im Leitfaden zur Programmierung</i>.</p> <p>Wenn <i>operand1</i> ein Datenbankfeld ist, ändert sich der Wert des Feldes auf der Datenbank dadurch nicht.</p> <p>Falls <i>operand1</i> eine dynamische Variable ist, wird er bis zur Länge von <i>operand2</i> oder bis zur Länge des Ergebnisses der arithmetischen Operation einschließlich der nachfolgenden Leerzeichen aufgefüllt, und die Länge von <i>operand1</i> wird dann entsprechend angepasst.</p> <p>Die aktuelle Länge einer dynamischen Variable kann durch die Systemvariable <code>*LENGTH</code> bestimmt werden.</p> <p>Allgemeine Informationen zu dynamischen Variablen entnehmen Sie dem Abschnitt <i>Dynamische und große Variablen benutzen</i>.</p>
<i>arithmetic-expression</i>	<p>Ein arithmetischer Ausdruck (<i>arithmetic-expression</i>) besteht aus einer oder mehreren Konstanten, Datenbankfeldern bzw. Benutzervariablen.</p> <p>Mathematische Natural-Funktionen (siehe <i>Systemfunktionen-Dokumentation</i>) können ebenfalls als arithmetische Operanden verwendet werden.</p> <p>Die in einem arithmetischen Ausdruck verwendeten Operanden müssen eines der folgenden Formate haben: N, P, I, F, D oder T.</p> <p>Zum Format der Operanden siehe auch unter <i>Formatwahl im Hinblick auf die Verarbeitungszeit im Leitfaden zur Programmierung</i>.</p> <p>Die folgenden Operatoren können verwendet werden:</p>

	Operator	Symbol
	Klammern	()
	Potenzierung	**
	Multiplikation	*
	Division	/
	Addition	+
	Subtraktion	-
	<p>Jedem Operatorzeichen sollte jeweils mindestens ein Leerzeichen voran- und nachgestellt werden, damit es nicht zu Konflikten mit Variablenamen, die eines dieser Zeichen enthalten, kommen kann.</p> <p>Bei der Verarbeitung arithmetischer Operationen gilt folgende Reihenfolge:</p> <ol style="list-style-type: none"> 1. Klammerrechnung 2. Potenzrechnung 3. Multiplikation/Division (von links nach rechts) 4. Addition/Subtraktion (von links nach rechts) 	
<i>operand2</i>	<p>Ausgangsfeld:</p> <p><i>operand2</i> ist das Ausgangsfeld.</p> <p>Wenn <i>operand1</i> das Format C hat, kann <i>operand2</i> auch als eine Attribut-Konstante angegeben werden (siehe <i>Benutzerkonstanten im Leitfaden zur Programmierung</i>).</p>	

Ergebnisgenauigkeit einer Division

Die Genauigkeit (Anzahl der Dezimalstellen) des Ergebnisses einer Division in einem COMPUTE-Statement bestimmt sich entweder aus der Genauigkeit des ersten Operanden (Dividenden) oder der des ersten Ergebnisfeldes, je nachdem welche größer ist.

Bei einer Division von Ganzzahlen gilt dagegen folgendes: Die Ergebnisgenauigkeit einer Division von zwei Ganzzahl-Konstanten bestimmt sich aus der Genauigkeit des ersten Ergebnisfeldes; ist jedoch eine der beiden Ganzzahlen eine Variable, dann ist auch das Ergebnis eine Ganzzahl (d.h. ohne Dezimalstellen, ganz gleich welche Genauigkeit das Ergebnisfeld hat).

SUBSTRING-Option

Wenn die Operanden alphanumerisches, Unicode- oder binäres Format haben, können Sie die **SUBSTRING**-Option verwenden (in der gleichen Weise wie beim **MOVE**-Statement beschrieben), um *operand1* einen Teil von *operand2* zuzuweisen.

Beispiele

- Beispiel 1 — ASSIGN-Statement
- Beispiel 2 — COMPUTE-Statement

Beispiel 1 — ASSIGN-Statement

```

** Example 'ASGEX1S': ASSIGN (structured mode)
*****
DEFINE DATA LOCAL
1 #A (N3)
1 #B (A6)
1 #C (NO.3)
1 #D (NO.5)
1 #E (N1.3)
1 #F (N5)
1 #G (A25)
1 #H (A3/1:3)
END-DEFINE
*
ASSIGN #A = 5
ASSIGN #B = 'ABC'
ASSIGN #C = .45
ASSIGN #D = #E = -0.12345
ASSIGN ROUNDED #F = 199.999
#G := 'HELLO'
#H (1) := 'UVW'
#H (3) := 'XYZ'
*
WRITE NOTITLE '=' #A
WRITE '=' #B
WRITE '=' #C
WRITE '=' #D / '=' #E
WRITE '=' #F
WRITE '=' #G
WRITE '=' #H (1:3)
*
END

```

Ausgabe des Programms ASGEX1S:

```
#A: 5
#B: ABC
#C: .450
#D: -.12345
#E: -0.123
#F: 200
#G: HELLO
#H: UVW XYZ
```

Äquivalentes Reporting-Mode-Beispiel: [ASGEX1R](#).

Beispiel 2 — COMPUTE-Statement

```
** Example 'CPTX1': COMPUTE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
2 PERSONNEL-ID
2 SALARY (1:2)
*
1 #A (P4)
1 #B (N3.4)
1 #C (N3.4)
1 #CUM-SALARY (P10)
1 #I (P2)
END-DEFINE
*
COMPUTE #A = 3 * 2 + 4 / 2 - 1
WRITE NOTITLE 'COMPUTE #A = 3 * 2 + 4 / 2 - 1' 10X '=' #A
*
COMPUTE ROUNDED #B = 3 - 4 / 2 * .89
WRITE 'COMPUTE ROUNDED #B = 3 - 4 / 2 * .89' 5X '=' #B
*
COMPUTE #C = SQRT (#B)
WRITE 'COMPUTE #C = SQRT (#B)' 18X '=' #C
*
LIMIT 1
READ EMPLOY-VIEW BY PERSONNEL-ID STARTING FROM '20017000'
WRITE / 'CURRENT SALARY: ' 4X SALARY (1)
/ 'PREVIOUS SALARY:' 4X SALARY (2)
FOR #I = 1 TO 2
COMPUTE #CUM-SALARY = #CUM-SALARY + SALARY (#I)
END-FOR
WRITE 'CUMULATIVE SALARY:' #CUM-SALARY
END-READ
*
END
```

Ausgabe des Programms CPTX1:

```
COMPUTE #A = 3 * 2 + 4 / 2 - 1      #A:      7
COMPUTE ROUNDED #B = 3 -4 / 2 * .89  #B:     1.2200
COMPUTE #C = SQRT (#B)              #C:     1.1045

CURRENT SALARY:      34000
PREVIOUS SALARY:    32300
CUMULATIVE SALARY:  66300
```


24 CREATE OBJECT

▪ Funktion	148
▪ Syntax-Beschreibung	148

```
CREATE OBJECT operand1 OF [CLASS] operand2
  [ON [NODE] operand3]
  [GIVING operand4]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: DEFINE CLASS | INTERFACE | METHOD | PROPERTY | SEND METHOD

Gehört zur Funktionsgruppe: *Komponenten-basierte Anwendungen erstellen*

Funktion

Das Statement CREATE OBJECT dient zum Erstellen einer Instanz einer Klasse.

Wenn ein CREATE OBJECT-Statement auf Windows-Plattformen ausgeführt wird, prüft Natural, ob der Name der in dem Statement angegebenen Klasse als DCOM-Klasse registriert ist. Falls dies der Fall ist, wird das Objekt mittels DCOM erstellt. Andernfalls sucht Natural in der aktuellen Library oder in den Steplibs nach einer Klasse dieses Namens und erstellt das Objekt lokal.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate												Referenzierung erlaubt	Dynam. Definition				
<i>operand1</i>		S																O	nein	nein
<i>operand2</i>	C	S																	ja	nein
<i>operand3</i>	C	S																	ja	nein
<i>operand4</i>		S				N													ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Objekt-Handle:</p> <p><i>operand1</i> muss als Objekt-Handle (HANDLE OF OBJECT) definiert sein. Die Objekt-Handle wird gefüllt, wenn das Objekt erfolgreich erstellt wurde. Wenn <i>operand1</i> nicht erfolgreich zurückgegeben wird, enthält er den Wert NULL-HANDLE.</p>
<p>OF CLASS</p> <p><i>operand2</i></p>	<p>Klassen-Name:</p> <p><i>operand2</i> ist der Name der Klasse, für die das Objekt erstellt werden soll. Bei Klassen, die nicht als DCOM-Klassen registriert sind, muss er den im DEFINE CLASS-Statement definierten Klassen-Namen enthalten. Bei Klassen, die registriert sind, muss er entweder die ProgID der Klasse oder die Klasse GUID enthalten. Bei als DCOM registrierten Natural-Klassen entspricht die ProgID dem im DEFINE CLASS-Statement angegebenen Klassen-Namen.</p> <p>Weitere Informationen entnehmen Sie dem Unterabschnitt <i>Registration with Natural</i> im Abschnitt <i>Distributing Object-based Natural Applications</i> (siehe <i>NaturalX</i> im <i>Leitfaden zur Programmierung</i>).</p> <pre>CREATE OBJECT #01 OF CLASS "Employee" or CREATE OBJECT #01 OF CLASS "653BCFEO-84DA-11D0-BEB3-10005A66D231"</pre>
<p>ON NODE</p> <p><i>operand3</i></p>	<p>Node:</p> <p>Als <i>operand3</i> geben Sie den Node an, unter dem das Objekt erstellt wird. Dies ist nur dann möglich, wenn die Klasse eine registrierte DCOM-Klasse ist. Wenn Sie die Node-Klausel verwenden, versucht Natural, das Objekt unter diesem Node zu erstellen. Wenn Sie die Node-Klausel nicht verwenden oder wenn diese einen leeren Wert enthält, wird das Objekt unter dem Node erstellt, der für diese Klasse in der System Registry unter dem Schlüssel <code>RemoteServerName</code> erstellt. Wird der Registry-Schlüssel nicht angegeben, wird das Objekt in der lokalen Natural-Session erstellt. Zum Beispiel:</p> <pre>CREATE OBJECT #01 OF CLASS "Employee" ON NODE "volcano.iceland.com"</pre>
<p>GIVING</p> <p><i>operand4</i></p>	<p>GIVING-Klausel:</p> <p>Wenn die GIVING-Klausel angegeben wird, enthält <i>operand4</i> entweder die Natural-Meldungsnummer, falls ein Fehler auftritt, oder Null bei fehlerfreier Ausführung.</p> <p>Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung ausgelöst, falls ein Fehler auftritt.</p>

25

DECIDE FOR

▪ Funktion	152
▪ Syntax-Beschreibung	152
▪ Beispiele	153

```

DECIDE FOR      { FIRST }      CONDITION
                 { EVERY }
{WHEN logical-condition statement ...} ...
[WHEN ANY statement ...]
[WHEN ALL statement ...]
WHEN NONE statement ...
END-DECIDE
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `DECIDE ON` | `IF` | `IF SELECTION` | `ON ERROR`

Gehört zur Funktionsgruppe: *Logische Bedingungen*

Funktion

Das Statement `DECIDE FOR` dient dazu, in Abhängigkeit von mehreren Bedingungen eine oder mehrere Handlungen auszuführen.



Anmerkung: Falls unter einer bestimmten Bedingung *keine* Handlung ausgeführt werden soll, geben Sie das Statement `IGNORE` in der betreffenden Klausel des `DECIDE FOR`-Statements an.

Syntax-Beschreibung

FIRST CONDITION	Nur die erste wahre Bedingung soll verarbeitet werden. Siehe auch <i>Beispiel 1</i> .
EVERY CONDITION	Jede wahre Bedingung soll verarbeitet werden. Siehe auch <i>Beispiel 2</i> .
WHEN <i>logical-condition statement</i>	Mit dieser Klausel geben Sie eine oder mehrere logische Bedingungen (<i>logical-condition</i>) an, die verarbeitet werden sollen (siehe Abschnitt <i>Logische Bedingungen</i> im Leitfaden zur Programmierung).
WHEN ANY <i>statement</i>	Mit <code>WHEN ANY</code> können Sie das (die) Statement(s) angeben, die ausgeführt werden sollen, wenn irgendeine der angegebenen Bedingungen erfüllt ist.
WHEN ALL <i>statement</i>	Mit <code>WHEN ALL</code> können Sie das (die) Statement(s) angeben, die ausgeführt werden sollen, wenn alle angegebenen Bedingungen erfüllt sind. Diese Klausel kann nur in Verbindung mit dem Schlüsselwort <code>EVERY</code> eingesetzt werden.

WHEN NONE <i>statement</i>	Mit WHEN NONE können Sie das (die) Statement(s) angeben, die ausgeführt werden sollen, wenn keine der angegebenen Bedingungen erfüllt ist.
END-DECIDE	Das für Natural reservierte Wort END-DECIDE muss zum Beenden des DECIDE FOR-Statements benutzt werden.

Beispiele

- [Beispiel 1 — DECIDE FOR-Statement mit FIRST-Option](#)
- [Beispiel 2 - DECIDE FOR-Statement mit EVERY-Option](#)

Beispiel 1 — DECIDE FOR-Statement mit FIRST-Option

```

** Example 'DECEX1': DECIDE FOR (with FIRST option)
*****
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARAM    (A1)
END-DEFINE
*
INPUT #FUNCTION #PARAM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARAM = 'X'
    WRITE 'Funktion A with parameter X selected.'
  WHEN #FUNCTION = 'B' AND #PARAM = 'X'
    WRITE 'Funktion B with parameter X selected.'
  WHEN #FUNCTION = 'C' THRU 'D'
    WRITE 'Funktion C or D selected.'
  WHEN NONE
    REINPUT 'Please enter a valid function.'
    MARK *#FUNCTION
END-DECIDE
*
END

```

Ausgabe des Programms DECEX1:

```
#FUNCTION A #PARAM Y
```

Drücken Sie dann die EINGABE-Taste:

```
PLEASE ENTER A VALID FUNCTION
#FUNCTION A #PARM Y
```

Beispiel 2 - DECIDE FOR-Statement mit EVERY-Option

```
** Example 'DECEX2': DECIDE FOR (with EVERY option)
*****
DEFINE DATA LOCAL
1 #FIELD1 (N5.4)
END-DEFINE
*
INPUT #FIELD1
*
DECIDE FOR EVERY CONDITION
  WHEN #FIELD1 >= 0
    WRITE '#FIELD1 is positive or zero.'
  WHEN #FIELD1 <= 0
    WRITE '#FIELD1 is negative or zero.'
  WHEN FRAC(#FIELD1) = 0
    WRITE '#FIELD1 has nein decimal digits.'
  WHEN ANY
    WRITE 'Any of the above conditions is true.'
  WHEN ALL
    WRITE '#FIELD1 is zero.'
  WHEN NONE
    IGNORE
END-DECIDE
*
END
```

Ausgabe des Programms DECEX2:

```
#FIELD1 42
```

Drücken Sie dann die EINGABE-Taste:

```
Page      1                                05-01-11  14:56:26

#FIELD1 is positive or zero.
#FIELD1 has nein decimal digits.
Any of the above conditions is true.
```

26

DECIDE ON

▪ Funktion	156
▪ Syntax-Beschreibung	157
▪ Beispiele	158

```
DECIDE ON
{ FIRST
  EVERY } [VALUES] [OF]
      { operand1
        SUBSTRING (operand3,operand5,operand6) }
{ VALUES { operand2
            SUBSTRING (operand4,operand7,operand8) }
          [[{ operand2
              SUBSTRING (operand4,operand7,operand8) }]] ...
          [:{ operand2
              SUBSTRING (operand4,operand7,operand8) }]] statement ...}
[ANY [VALUES] statement ...]
[ALL [VALUES] statement ...]
NONE [VALUES] statement ...
END-DECIDE
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `DECIDE FOR` | `IF` | `IF SELECTION` | `ON ERROR`

Gehört zur Funktionsgruppe: *Logische Bedingungen*

Funktion

Das Statement `DECIDE ON` dient dazu, in Abhängigkeit vom Wert (bzw. von den Werten) einer Variablen eine oder mehrere Handlungen auszuführen.



Anmerkung: Falls unter einer bestimmten Bedingung *keine* Handlung ausgeführt werden soll, geben Sie das Statement `IGNORE` in der betreffenden Klausel des `DECIDE ON`-Statements an.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate										Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>		S	A		N	A	U	N	P	I	F	B	D	T	L	G	O	ja	nein
<i>operand2</i>	C	S	A			A	U	N	P	I	F	B	D	T	L	G	O	ja	nein
<i>operand3</i>		S	A			A	U					B						ja	nein
<i>operand4</i>	C	S	A			A	U					B						ja	nein
<i>operand5</i>	C	S							N	P	I	B*						ja	nein
<i>operand6</i>	C	S							N	P	I	B*						ja	nein
<i>operand7</i>	C	S							N	P	I	B*						ja	nein
<i>operand8</i>	C	S							N	P	I	B*						ja	nein

* Format B von *operand5*, *operand6*, *operand7* und *operand8* kann nur mit einer Länge von kleiner oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

FIRST/EVERY	Zu verarbeitender Wert: Mit einem dieser Schlüsselwörter geben Sie an, ob nur der erste gefundene Wert (FIRST) oder alle gefundenen Werte (EVERY) der Variablen verarbeitet werden sollen.
<i>operand1</i>	Kontrollfeld: Als <i>operand1</i> oder <i>operand2</i> geben Sie den Namen des Feldes an, dessen Werte geprüft werden sollen.
VALUES <i>operand2</i> [, <i>operand2</i>] ... [: <i>operand2</i>] <i>statement</i> ...	Wert des Kontrollfeldes: Mit dieser Klausel geben Sie den Wert (<i>operand2</i>) des Kontrollfeldes an, sowie die <i>statements</i> , die ausgeführt werden sollen, wenn das Kontrollfeld diesen Wert hat. Sie können einen Wert, mehrere Werte oder einen Bereich von Werten angeben, vor denen als Option einer oder mehrere Wert/e stehen können. Werden mehrere Werte angegeben, müssen diese entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander getrennt werden. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimal komma (mit dem Session-Parameter DC) definiert ist.

	Bei einem Bereich von Werten geben Sie, durch einen Doppelpunkt voneinander getrennt, den Anfangs- und den Endwert des Bereiches an.
SUBSTRING <i>(operand3,operand5,operand6)</i>	<p>SUBSTRING-Option:</p> <p>Wenn Sie die SUBSTRING-Option weglassen, wird der gesamte Inhalt eines Feldes geprüft. Mit der SUBSTRING-Option können Sie nur einen bestimmten Teil eines alphanumerischen, Unicode- oder binären Feldes prüfen.</p> <p>Nach dem Feldnamen (<i>operand3</i>) geben Sie zuerst die Startposition (<i>operand5</i>) und danach die Länge (<i>operand6</i>) des zu prüfenden Teils des Feldes an.</p>
SUBSTRING <i>(operand4,operand7,operand8)</i>	Nach dem Feldnamen (<i>operand4</i>) geben Sie zuerst die Startposition (<i>operand7</i>) und danach die Länge (<i>operand8</i>) des zu prüfenden Teils des Feldes an.
ANY statement	<p>ANY-Klausel:</p> <p>Mit ANY geben Sie das (die) Statement(s) an, die ausgeführt werden sollen, wenn irgendeiner der in der VALUES-Klausel angegebenen Werte gefunden wird. Diese Statements werden zusätzlich zu den in der VALUES-Klausel angegebenen Statements ausgeführt.</p>
ALL statement	<p>ALL-Klausel:</p> <p>Mit ALL geben Sie das (die) Statement(s) an, die ausgeführt werden sollen, wenn alle in der VALUES-Klausel angegebenen Werte gefunden werden. Diese Statements werden zusätzlich zu den in der VALUES-Klausel angegebenen Statements ausgeführt.</p> <p>Die ALL-Klausel kann nur in Verbindung mit dem Schlüsselwort EVERY eingesetzt werden</p>
NONE statement	<p>NONE-Klausel:</p> <p>Mit NONE geben Sie das (die) Statement(s) an, die ausgeführt werden sollen, wenn keiner der angegebenen Werte gefunden wurde.</p>
END-DECIDE	Das für Natural reservierte Wort END-DECIDE muss zum Beenden des DECIDE ON-Statements benutzt werden.

Beispiele

- Beispiel 1 — DECIDE ON-Statement mit FIRST-Option

- Beispiel 2 — DECIDE ON-Statement mit EVERY-Option

Beispiel 1 — DECIDE ON-Statement mit FIRST-Option

```

** Example 'DECEX3': DECIDE ON (with FIRST option)
*****
*
SET KEY ALL
INPUT 'Enter any PF key' /
      'and check result' /
*
DECIDE ON FIRST VALUE OF *PF-KEY
  VALUE 'PF1'
    WRITE 'PF1 key entered.'
  VALUE 'PF2'
    WRITE 'PF2 key entered.'
  ANY VALUE
    WRITE 'PF1 or PF2 key entered.'
  NONE VALUE
    WRITE 'Neither PF1 nor PF2 key entered.'
END-DECIDE
*
END

```

Ausgabe des Programms DECEX3:

```

Enter any PF key
and check result

```

Ausgabe nach Drücken von PF1:

```

Page      1                               05-01-11  15:08:50

PF1 key entered.
PF1 or PF2 key entered.

```

Beispiel 2 — DECIDE ON-Statement mit EVERY-Option

```
** Example 'DECEX4': DECIDE ON (with EVERY option)
*****
DEFINE DATA LOCAL
1 #FIELD (N1)
END-DEFINE
*
INPUT 'Enter any value between 1 and 9:' #FIELD (SG=OFF)
*
DECIDE ON EVERY VALUE OF #FIELD
  VALUE 1 : 4
    WRITE 'Content of #FIELD is 1-4'
  VALUE 2 : 5
    WRITE 'Content of #FIELD is 2-5'
  ANY VALUE
    WRITE 'Content of #FIELD is 1-5'
  ALL VALUE
    WRITE 'Content of #FIELD is 2-4'
  NONE VALUE
    WRITE 'Content of #FIELD is not 1-5'
  END-DECIDE
*
END
```

Ausgabe des Programms DECEX4:

```
ENTER ANY VALUE BETWEEN 1 AND 9: 4
```

Nach Eingabe und Bestätigung des Wertes 4:

```
Page      1                                     05-01-11  15:11:45

Content of #FIELD is 1-4
Content of #FIELD is 2-5
Content of #FIELD is 1-5
Content of #FIELD is 2-4
```

27 DEFINE CLASS

▪ Funktion	162
▪ Syntax-Beschreibung	163

```

DEFINE CLASS class-name
[ [WITH] ACTIVATION [POLICY] { EM } ]
[ ES ]
[ IM ]
[ OBJECT { USING { local-data-area } } ] ...
[ { parameter-data-area } ]
[ data-definition... ]
[ LOCAL { USING { local-data-area } } ] ...
[ { parameter-data-area } ]
[ data-definition ]
[ID class-GUID]
[ INTERFACE USING copycode ] ...
[ INTERFACE ] ...
[PROPERTY ] ...
[METHOD ] ...
END-CLASS

```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [CREATE OBJECT](#) | [INTERFACE](#) | [METHOD](#) | [PROPERTY](#) | [SEND METHOD](#)

Gehört zur Funktionsgruppe: [Komponenten-basierte Anwendungen erstellen](#)

Funktion

Das Statement `DEFINE CLASS` dient dazu, eine Klasse innerhalb eines Natural Class-Moduls anzugeben.

Ein Natural Class-Modul besteht aus einem `DEFINE CLASS`-Statement gefolgt von einem `END`-Statement.

Syntax-Beschreibung

<i>class-name</i>	<p>Klassen-Name:</p> <p>Dies ist der Name, der von Clients benutzt wird, um Objekte dieser Klasse zu erstellen. Er kann maximal bis zu 32 Zeichen lang sein und Punkte enthalten. Deshalb kann es Klassen-Namen geben wie:</p> <p><i>company-name. application-name. class-name</i></p> <p>Jeder Bestandteil zwischen den Punkten (...) muss den Natural-Namenskonventionen für Benutzervariablen entsprechen.</p> <p>Wenn die Klasse von in verschiedenen Programmiersprachen geschriebenen Clients verwendet werden soll, sollte der Klassen-Name so gewählt werden, dass er nicht gegen die in diesen Sprachen geltenden Namenskonventionen verstößt.</p>						
WITH ACTIVATION POLICY	<p>WITH ACTIVATION POLICY-Klausel:</p> <p>Diese Klausel dient dazu, die Activation Policy zu definieren, die für die aktuelle Klasse registriert ist.</p> <p>Sie können folgende Parameter angeben:</p> <table border="1" style="width: 100%;"> <tr> <td>EM</td> <td>Die Activation Policy ist ExternalMultiple.</td> </tr> <tr> <td>ES</td> <td>Die Activation Policy ist ExternalSingle.</td> </tr> <tr> <td>IM</td> <td>Die Activation Policy ist InternalMultiple.</td> </tr> </table> <p>Wenn die Klasse mit STOW gespeichert und registriert wird, überschreibt die Einstellung in der WITH ACTIVATION POLICY-Klausel die mit dem Profilparameter ACTPOLICY vorgenommene Einstellung, aber sie wird ihrerseits durch die manuelle Registrierung mittels REGISTER-Kommando und expliziter Activation-Policy-Definition überschrieben.</p> <p>Weitere Informationen siehe <i>Activation Policies</i> in der <i>Operations</i>-Dokumentation.</p>	EM	Die Activation Policy ist ExternalMultiple.	ES	Die Activation Policy ist ExternalSingle.	IM	Die Activation Policy ist InternalMultiple.
EM	Die Activation Policy ist ExternalMultiple.						
ES	Die Activation Policy ist ExternalSingle.						
IM	Die Activation Policy ist InternalMultiple.						
OBJECT	<p>OBJECT-Klausel:</p> <p>Die OBJECT-Klausel dient dazu, Objektdaten zu definieren. Die Syntax der OBJECT-Klausel entspricht der für die LOCAL-Klausel des DEFINE DATA-Statements. Weitere Informationen siehe Beschreibung der LOCAL-Klausel des DEFINE DATA-Statements.</p>						
LOCAL	<p>LOCAL-Klausel:</p> <p>Die LOCAL-Klausel dient dazu, global eindeutige IDs (GUID = Globally Unique ID) in die Klassen-Definition aufzunehmen. GUIDs müssen nur definiert werden, wenn eine Klasse für DCOM registriert werden soll. GUIDs werden meistens in einer Local Data Area (LDA) definiert. Weitere Informationen siehe <i>Globally Unique Identifiers (GUIDs)</i> im <i>Leitfaden zur Programmierung</i>.</p>						

	Die Syntax der LOCAL-Klausel entspricht der für die LOCAL-Klausel des DEFINE DATA-Statements. Weitere Informationen siehe Beschreibung der LOCAL-Klausel des DEFINE DATA-Statements.
ID	<p>ID-Klausel:</p> <p>Die ID-Klausel dient dazu, der Klasse eine GUID zuzuweisen. Die GUID der Klasse ist der Name einer in der Data Area definierten GUID, die mit der LOCAL-Klausel eingefügt wird. Die Klasse GUID ist eine (mit Namen versehene) alphanumerische Konstante. Einer Klasse muss eine GUID zugewiesen werden, wenn diese unter DCOM registriert werden soll.</p>
INTERFACE USING	<p>INTERFACE USING-Klausel:</p> <p>Die INTERFACE USING-Klausel wird verwendet, um einen Copycode aufzunehmen, der INTERFACE-Statements enthält.</p>
<i>copycode</i>	<p>Copycode:</p> <p>Der von der INTERFACE USING-Klausel verwendete Copycode kann eines oder mehrere INTERFACE-Statements enthalten.</p>
PROPERTY	<p>PROPERTY-Statement:</p> <p>Das PROPERTY-Statement wird benutzt, um einer Property einen Objektdaten-Operanden als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.</p>
METHOD	<p>METHOD-Statement:</p> <p>Das METHOD-Statement wird benutzt, um einer Methode ein Subprogramm als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.</p>
END-CLASS	Das für Natural reservierte Wort END-CLASS muss zum Beenden des DEFINE CLASS-Statements benutzt werden.

28 DEFINE DATA

Allgemeine Syntax

```
DEFINE DATA
  [GLOBAL USING global-data-area [WITH block[.block]...]]
  [
    PARAMETER { USING parameter-data-area } ] ...
                { parameter-data-definition... } ]
  [
    OBJECT { USING { local-data-area } } ] ...
              { parameter-data-area } } ]
              { data-definition... } ]
  [
    LOCAL { USING { local-data-area } } ] ...
              { parameter-data-area } } ]
              { direct-data-definition... } ]
  [INDEPENDENT AIV-data-definition ...]
  [
    CONTEXT { USING { local-data-area } } ]
                { parameter-data-area } } ]
                { context-data-definition ... } ]
END-DEFINE
```

Das DEFINE DATA-Statement bietet eine Reihe von Klauseln, um Datendefinitionen für ein Natural-Programm vorzunehmen, und zwar entweder durch Referenzieren vordefinierter Datendefinitionen, die in einer Local Data Area (LDA), Global Data Area (GDA) oder Parameter Data Area (PDA) enthalten sind, oder durch Schreiben von Inline-Definitionen.

Weitere Informationen zum Gebrauch des DEFINE DATA-Statements finden Sie im Abschnitt *Felder definieren* im Leitfaden zur Programmierung.

Die Dokumentation für das `DEFINE DATA`-Statement ist in die folgenden Abschnitte unterteilt:

- [Syntax-Übersicht](#)
- [DEFINE DATA - Allgemeines](#)

Spezifische Datendefinitionen:

- [Definition von Local Data](#)
- [Definition von Global Data](#)
- [Definition von Parameter Data](#)
- [Definition von anwendungsunabhängigen Variablen](#)
- [Definition von Kontext-Variablen für den Natural RPC](#)
- [Definition von NaturalX-Objekten](#)

Klauseln und Optionen:

- [Variablen-Definition](#)
- [View-Definition](#)
- [Redefinition](#)
- [Handle-Definition](#)
- [Definition der Array-Dimension](#)
- [Definition des Ausgangswertes](#)
- [Ausgangswerte und Konstanten-Werte für ein Array](#)
- [EM-, HD-, PD-Parameter für Feld/Variable](#)

Beispiele:

- [Beispiele für die Benutzung des DEFINE DATA-Statements](#)

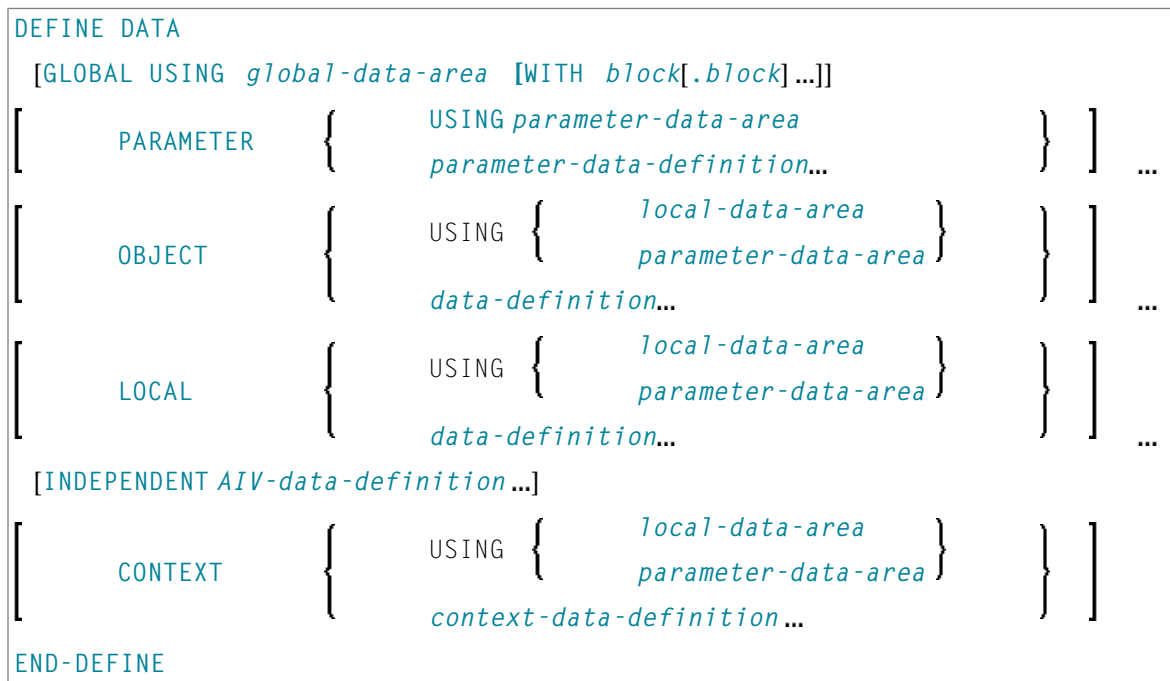
29 Syntax-Übersicht

▪ Allgemeine Syntax	168
▪ Basis-Syntaxelemente	168

Dieses Kapitel enthält eine Übersicht über alle in den Beschreibungen des `DEFINE DATA`-Statements benutzten Syntax-Diagrammen.

Es liefert Informationen zu der Art und Weise, wie die Schlüsselwörter, Klauseln, Parameter, Optionen und andere Syntax-Elemente in den Programm-Statementzeilen angeordnet und miteinander kombiniert werden sollen.

Allgemeine Syntax



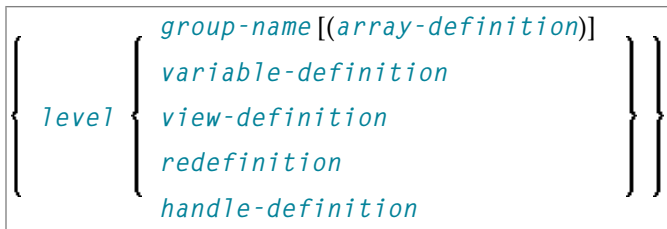
Basis-Syntaxelemente

Folgende Syntax-Elemente werden vorgestellt:

- `data-definition`
- `parameter-data-definition`
- `parameter-handle-definition`
- `variable-definition`
- `view-definition`
- `redefinition`
- `init-definition`
- `array-definition`
- `array-init-definition`

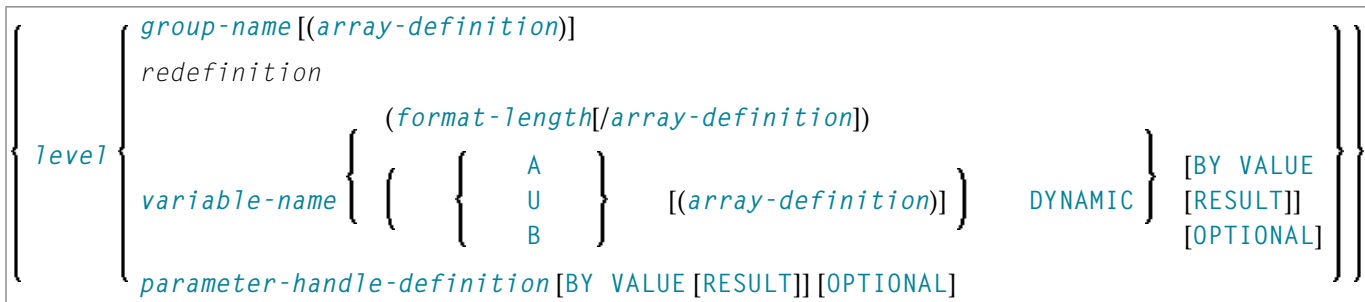
- emhdpn
- AIV-data-definition
- context-data-definition

data-definition



Weitere Informationen entnehmen Sie den Abschnitten *Definition von Local Data* oder *Definition von NaturalX-Objekten*.

parameter-data-definition



Weitere Informationen siehe *Definition von Parameter Data*.

parameter-handle-definition



Weitere Informationen siehe *Parameter-Handle-Data-Definition*.

variable-definition

```
{ <scalar-definition>
  <array-definition> }
```

<scalar-definition>

```
variable-name { (format-length)
                ( { { A
                   U
                   B } } DYNAMIC ) [ { { CONSTANT
                                       INIT } } init-definition ] [emhdpm]
```

<array-definition>

```
variable-name { (format-length/array-definition)
                ( { { A
                   U
                   B } /array-definition ) DYNAMIC } [ { { CONSTANT
                                                           INIT } } array-init-definition ] [emhdpm]
```

Weitere Informationen siehe *Definition von Variablen*.

view-definition

```
view-name
VIEW [OF]
ddm-name [ level { ddm-field [ [ ( ( [format-length]/[array-definition])
                                   ( { { A
                                       U
                                       B } } [/array-definition] )
                                   DYNAMIC ] [emhdpm] ] ] ] ...
         [ redefinition ] ] ] ]
```

Weitere Informationen siehe *View-Definition*.

redefinition

```
REDEFINE field-name { level { rgroup
                           rfield(format-length [/array-definition]) } }...
                           FILLER nX }
```

Weitere Informationen siehe [Redefinition](#).

init-definition

```
{ <constant>
  <system-variable>
  FULL LENGTH <character-s>
  LENGTH n <character-s> }
```

Weitere Informationen siehe [Ausgangswerte/Konstanten-Werte für ein Array](#).

array-definition

```
{bound[:bound]}...3
```

Weitere Informationen siehe [Definition der Array-Dimension](#).

array-init-definition

```
{ { { ALL
      ( { index[:index]
        { v },...3 ) } } } { FULL LENGTH
                              LENGTH n } <character-s,...>
  < { constant
      system-variable,... } > } ...
```

Weitere Informationen siehe [Ausgangswerte/Konstanten-Werte für ein Array](#).

emhdpm

`([EM=value] [HD= 'text '] [PM=value])`

Weitere Informationen siehe *EM-, HD-, PM-Parameter für Feld/Variable*.

AIV-data-definition

`level { variable-definition }
redefinition
handle-definition }`

Weitere Informationen siehe *Definition von anwendungsunabhängigen Variablen*.

context-data-definition

`level { variable-definition }
redefinition
handle-definition }`

Weitere Informationen siehe *Definition von Kontext-Variablen für den Natural RPC*

30

DEFINE DATA - Allgemeines

▪ Funktion	174
▪ Regeln	174
▪ Programmiermodi	174
▪ Weitere Informationen	175

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `DEFINE DATA` bietet eine Reihe von Klauseln, um Datendefinitionen in einem Natural-Programm vorzunehmen, und zwar entweder durch Referenzieren vordefinierter Datendefinitionen, die in einer Local Data Area (LDA), Global Data Area (GDA) oder Parameter Data Area (PDA) enthalten sind, oder durch Eingabe von Inline-Definitionen.

Regeln

- Wenn ein `DEFINE DATA`-Statement benutzt wird, muss es das erste Statement des Programms oder der Subroutine sein.
- Ein „leeres“ `DEFINE DATA`-Statement ist nicht zulässig; mit anderen Worten, es muss *mindestens eine Klausel* (`LOCAL`, `GLOBAL`, `PARAMETER`, `INDEPENDENT`, `CONTEXT` oder `OBJECT`) angegeben werden, und *mindestens ein Feld* muss definiert sein.
- Sie können mehr als eine Klausel angeben; in diesem Fall müssen die Klauseln in der in den Syntaxdiagrammen gezeigten Reihenfolge angegeben werden.
- Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden des `DEFINE DATA`-Statements benutzt werden.

Programmiermodi

Das `DEFINE DATA`-Statement steht im Structured Mode und im Reporting Mode zur Verfügung. Unterschiede sind in der `DEFINE DATA`-Statement-Beschreibung entsprechend markiert.

Im Allgemeinen gilt Folgendes:

- `Structured Mode`

- Reporting Mode

Structured Mode

Im Structured Mode müssen alle verwendeten Variablen (außer **anwendungsunabhängigen Variablen** = AIVs) im `DEFINE DATA`-Statement definiert werden; sie dürfen *innerhalb* eines Programms an keiner anderen Stelle definiert werden. (*Außerhalb* eines Programms können Variablen in Data Areas definiert werden; siehe *Datenbereiche (Data Areas)* im *Leitfaden zur Programmierung*.)

AIVs dürfen nicht an anderer Stelle im Programm definiert werden, wenn ein `DEFINE DATA INDEPENDENT`-Statement benutzt wird.

Reporting Mode

Im Reporting Mode ist das `DEFINE DATA`-Statement nicht zwingend erforderlich, da Variablen auch an anderer Stelle im Programm definiert werden können.

Wenn Sie jedoch im Reporting Mode ein `DEFINE DATA LOCAL`-Statement verwenden, dürfen Sie an anderer Stelle im Programm keine weiteren Variablen (außer **anwendungsunabhängigen Variablen** = AIVs) definieren.

Wenn Sie im Reporting Mode ein `DEFINE DATA INDEPENDENT`-Statement verwenden, dürfen Sie an anderer Stelle im Programm keine weiteren AIVs definieren.

Weitere Informationen

Weitere Informationen zum `DEFINE DATA`-Statement finden Sie in den folgenden Abschnitten im *Leitfaden zur Programmierung*:

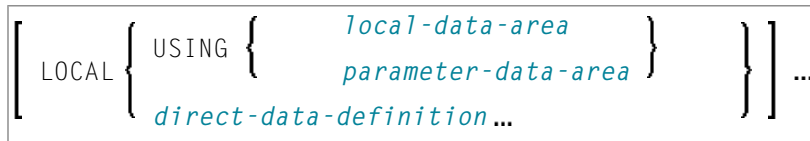
- *Felder definieren*
- *Benutzung von Data Areas*

31

Definition von Local Data

▪ Funktion	178
▪ Einschränkung	178
▪ Syntax-Beschreibung	178

Allgemeine Syntax von DEFINE DATA LOCAL:



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das DEFINE DATA LOCAL-Statement dient zur Definition der Datenelemente, die ausschließlich von einem einzelnen Natural-Modul in einer Anwendung benutzt werden. Diese Elemente oder Felder können auf folgende Weise definiert werden:

entweder innerhalb des DEFINE DATA LOCAL-Statements selbst unter Verwendung der *direct-data-definition*-Syntax (siehe [Direkte Daten-Definition](#))

oder außerhalb des Programms in einer separaten Local Data Area (LDA) oder einer Parameter Data Area (PDA), wobei das DEFINE DATA LOCAL USING-Statement diese Data Area referenziert.

Einschränkung

Die LDA und die sie referenzierenden Objekte müssen in derselben Library (oder in einer Steplib) enthalten sein.

Syntax-Beschreibung

<i>local-data-area</i>	<p>Geben Sie den Namen der zu referenzierenden Local Data Area (LDA) an.</p> <p>Eine Local Data Area wird mit dem <i>Data Area Editor</i> erstellt. Sie enthält vordefinierte Datenelemente, die in das DEFINE DATA LOCAL-Statement übernommen werden können.</p>
------------------------	---

	<p>Sie können mehr als eine Data Area referenzieren; in diesem Fall müssen Sie die reservierten Wörter LOCAL und USING wiederholen, zum Beispiel:</p> <pre>DEFINE DATA LOCAL USING DATX_L LOCAL USING DATX_P ... END-DEFINE ;</pre> <p>Weitere Informationen siehe auch <i>Felder in einer separaten Data Area und Local Data Area</i> und <i>Local Data Area, Beispiel 2</i> im Leitfaden zur Programmierung.</p>
<i>parameter-data-area</i>	<p>Geben Sie den Namen der zu referenzierenden Parameter Data Area (PDA) an.</p> <p>Anmerkung: Eine mit DEFINE DATA LOCAL referenzierte Data Area kann auch eine Parameter Data Area (PDA) sein. Durch Benutzung einer PDA als LDA können Sie sich die zusätzliche Mühe sparen, eine LDA zu erstellen, die dieselbe Struktur wie die PDA hat.</p> <p>Eine Parameter Data Area wird mit dem <i>Data Area Editor</i> erstellt. Sie enthält vordefinierte Datenelemente, die in das DEFINE DATA LOCAL-Statement übernommen werden können.</p>
<i>direct-data-definition</i>	Siehe Direkte Daten-Definition weiter unten.
END-DEFINE	Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

Direkte Daten-Definition

Local Data können direkt in einem Programm oder Subprogramm definiert werden. Für eine direkte Daten-Definition gilt die folgende Syntax:

{	level {	<i>group-name</i> [(<i>array-definition</i>)]	}
		<i>variable-definition</i>	
		<i>view-definition</i>	
		<i>redefinition</i>	
		<i>handle-definition</i>	

Weitere Informationen siehe

- [Beispiel 1 - DEFINE DATA LOCAL](#) (Direkte Daten-Definition)
- *Definition von Feldern in einem DEFINE DATA-Statement* im Leitfaden zur Programmierung
- *Local Data Area, Beispiel 1* im Leitfaden zur Programmierung

Syntax-Elementbeschreibung für die direkte Daten-Definition:

<i>level</i>	<p>Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte 0 ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet.</p> <p>Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren. Bei manchen Statements (CALL, CALLNAT, RESET, WRITE usw.) können Sie den Gruppennamen als Aufrufnamen angeben, um die in der Gruppe enthaltenen Felder zu referenzieren.</p> <p>Eine Gruppe kann aus anderen Gruppen bestehen. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.</p> <p>Eine <i>view-definition</i> muss immer auf Level 1 definiert werden.</p>
<i>group-name</i>	<p>Der Name einer Gruppe. Der Name muss den Regeln zur Definition eines Natural-Variablenamens entsprechen. Siehe auch die folgenden Abschnitte:</p> <ul style="list-style-type: none"> ■ <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen. ■ <i>Datenstrukturen qualifizieren</i> im Leitfaden zur Programmierung.
<i>array-definition</i>	<p>Mit <i>array-definition</i> definieren Sie die untere und obere Grenze einer Dimension in einer Array-Definition. Siehe Definition von Array-Dimensionen.</p>
<i>variable-definition</i>	<p>Die <i>variable-definition</i> dient zur Definition einer einzelnen Variablen (oder Feldes), die aus einem Wert (Skalar) oder mehreren Werten (Array) bestehen kann. Siehe Definition von Variablen.</p>
<i>view-definition</i>	<p>Die <i>view-definition</i> wird benutzt, um eine Datensicht (View) mit Bestandteilen aus einem Datendefinitionsmodul (DDM) zu definieren. Siehe View Definition.</p>
<i>redefinition</i>	<p>Eine <i>redefinition</i> kann zur Redefinition einer Gruppe, eines Views, eines DDM-Felds oder eines einzelnen Feldes oder einer einzelnen Variablen benutzt werden (d.h. Skalar oder Array). Siehe Redefinition.</p>
<i>handle-definition</i>	<p>Eine Handle identifiziert ein Dialog-Element im Code und wird in Handle-Variablen gespeichert. Siehe Handle-Definition.</p>

32

Definition von Global Data

- Funktion 182
- Syntax-Beschreibung 182

Allgemeine Syntax von DEFINE DATA GLOBAL:

```
DEFINE DATA
  GLOBAL USING global-data-area [WITH block [.block...]]
END-DEFINE
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das DEFINE DATA GLOBAL-Statement dient zur Definition von Datenelementen mittels einer **Global Data Area** (GDA).

Syntax-Beschreibung

USING <i>global-data-area</i>	<p>Geben Sie den Namen der zu referenzierenden Global Data Area (GDA) an.</p> <p>Eine Global Data Area wird mit dem <i>Data Area Editor</i> erstellt. Sie enthält vordefinierte Datenelemente, die in das DEFINE DATA LOCAL-Statement übernommen werden können.</p> <p>Im Gegensatz zur LDA können die in einer GDA definierten Datenelemente von mehreren Programmierobjekten referenziert werden.</p> <p>Weitere Informationen siehe Global Data Area im <i>Leitfaden zur Programmierung</i>.</p>
WITH <i>block</i>	<p>Um Datenspeicherplatz einzusparen, können Sie eine Global Data Area mit Datenblöcken erstellen. Datenblöcke können sich bei der Programmausführung gegenseitig überlagern, wodurch Speicherplatz eingespart wird.</p> <p>Die maximale Anzahl der Blockebenen ist 8 (einschließlich des Master-Blocks). Weitere Informationen, siehe <i>Datenblöcke</i> im <i>Leitfaden zur Programmierung</i>.</p>
<i>.block</i>	<p><i>.block</i>-Notationen geben den Block oder die Blöcke an, der bzw. die im Programm benutzt wird bzw. werden.</p>
END-DEFINE	<p>Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.</p>

33

Definition von Parameter Data

- Funktion 184
- Einschränkungen 184
- Syntax-Beschreibung 184

Allgemeine Syntax von DEFINE DATA PARAMETER:

```
[ PARAMETER { USING parameter-data-area } ] ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das DEFINE DATA PARAMETER-Statement wird benutzt, um die Datenelemente zu definieren, die als Eingabeparameter in einem Natural-Subprogramm, einer externen Subroutine, Helprountine, Function oder Dialog verwendet werden sollen. Diese Parameter können innerhalb des Statements selbst definiert werden (siehe [Parameter-Daten-Definition](#) weiter unten); oder sie können außerhalb des Programms in einer Parameter Data Area (PDA) definiert werden, wobei das Statement diese Data Area referenziert.

Einschränkungen

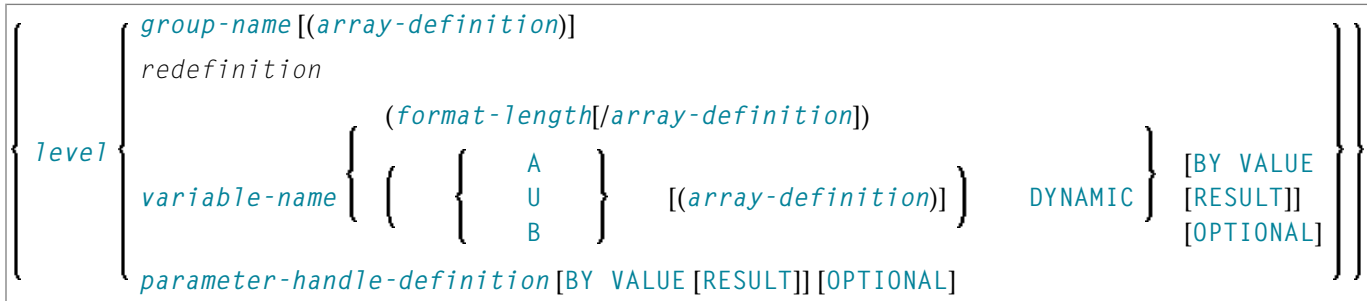
- Parameter-Datenelementen dürfen keine Ausgangswerte oder auch Konstanten-Werte zugewiesen werden, und sie dürfen keine Editiermasken-Definitionen (EM), Kopfzeilen-Definitionen (HD) oder Druckmodus-Definitionen (PM) haben (siehe auch [EM-, HD-, PM-Parameter für Feld/Variable](#)).
- Die Parameter Data Area und die sie referenzierenden Objekte müssen in derselben Library (oder in einer Steplib) enthalten sein.

Syntax-Beschreibung

USING <i>parameter-data-area</i>	Der Name der <i>parameter-data-area</i> , die Datenelemente enthält, die als Parameter in einem Subprogramm, einer externen Subroutine oder einem Dialog benutzt werden.
<i>parameter-data-definition</i>	Anstatt eine Parameter Data Area zu definieren, können Parameter auch direkt in einem Programm oder einer Routine definiert werden; siehe Definition von Parameterdaten weiter unten.
END-DEFINE	Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

Definition von Parameterdaten

Für die direkte Parameter-Daten-Definition gilt die folgende Syntax:



Syntax-Element-Beschreibung:

<i>level</i>	<p>Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte 0 ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächst-niedrigeren Level-Nummer betrachtet.</p> <p>Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren. Bei manchen Statements (CALL, CALLNAT, RESET, WRITE usw.) können Sie den Gruppennamen als Aufrufnamen angeben, um die in der Gruppe enthaltenen Felder zu referenzieren.</p> <p>Eine Gruppe kann aus weiteren Gruppen bestehen. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.</p>
<i>group-name</i>	<p>Der Name einer Gruppe. Der Name muss den Regeln zur Definition eines Natural-Variablenamens entsprechen. Siehe auch die folgenden Abschnitte:</p> <ul style="list-style-type: none"> ■ <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen. ■ <i>Datenstrukturen qualifizieren</i> im <i>Leitfaden zur Programmierung</i>.
<i>array-definition</i>	<p>Mit <i>array-definition</i> definieren Sie die untere und obere Grenze einer Dimension in einer Array-Definition. Siehe Definition von Array-Dimensionen und Variable Arrays in einer Parameter Data Area.</p>

<i>redefinition</i>	<p>Mit <i>redefinition</i> können Sie eine Gruppe oder ein einzelnes Feld oder eine einzelne Variable (d.h. Skalar oder Array) redefinieren. Siehe Redefinition.</p> <p>Anmerkung: In einer Parameter Data Area ist eine Redefinition von Gruppen nur innerhalb eines REDEFINE-Blocks zulässig.</p>
<i>variable-name</i>	<p>Der der Variablen zuzuweisende Name. Es gelten die Regeln für Natural- Variablennamen. Informationen zu Namenskonventionen für Benutzervariablen, siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i>.</p>
<i>format-length</i>	<p>Das Format und die Länge des Feldes. Informationen zu Format und Länge für Benutzervariablen, siehe den Abschnitt <i>Format und Länge von Benutzervariablen</i> im <i>Leitfaden zur Programmierung</i>.</p>
A, U or B	<p>Datentyp: Alphanumerisch (A) oder Binär (B) für dynamische Variable.</p>
DYNAMIC	<p>Ein Parameter kann als DYNAMIC definiert werden. Weitere Informationen zur Verarbeitung von dynamischen Variablen, siehe <i>Dynamische Variablen</i> im <i>Leitfaden zur Programmierung</i>.</p>
	<p>Call-Modus:</p> <p>Je nachdem, ob der Call-By-Reference- oder Call-By-Value-Modus benutzt wird, gilt die betreffende Übertragungsart. Weitere Informationen siehe CALLNAT- Statement.</p>
(without BY VALUE)	<p>Call-By-Reference-Modus:</p> <p>Ohne BY VALUE (gilt standardmäßig) wird ein Parameter durch Referenzierung seiner Adresse („By Reference“) an ein Subprogramm bzw. eine Subroutine übergeben; das bedeutet, dass ein in einem CALLNAT- bzw. PERFORM-Statement als Parameter angegebenes Feld dasselbe Format und dieselbe Länge haben muss wie das betreffende Feld in dem/der aufgerufenen Subprogramm/Subroutine.</p>
BY VALUE	<p>Call-By-Value-Modus:</p> <p>Mit BY VALUE wird ein Parameter direkt als Wert (<i>by value</i>) an ein Subprogramm oder eine Subroutine übergeben; d.h. statt der Adresse des Parameters wird der Wert selbst übergeben. Das bedeutet, das Feld in dem Subprogramm bzw. der Subroutine braucht nicht dasselbe Format und dieselbe Länge zu haben wie der Parameter beimCALLNAT-/PERFORM-Statement. Das Format und die Länge der beiden muss lediglich datenübertragungskompatibel gemäß den <i>Regeln für Datenübertragung/-zuweisung sein</i> (siehe <i>Leitfaden zur Programmierung</i>).</p> <p>Mit BY VALUE können Sie zum Beispiel die Länge eines Feldes in einem Subprogramm bzw. einer Subroutine vergrößern (falls eine Erweiterung des Subprogramms bzw. der Subroutine dies erforderlich machen sollte), ohne deswegen auch die Objekte, die das Subprogramm bzw. die Subroutine aufrufen, anpassen zu müssen.</p>

	<p>Für die Parameterdefinition bei Dialogen gilt Folgendes:</p> <ul style="list-style-type: none"> ■ Wenn Sie <code>BY VALUE</code> weglassen, wird ein Parameter gemäß den Angaben in der Inline-Definition der Parameter Data Area des Dialogs über seine Adresse (d.h. „by reference“) übertragen; das Format und die Länge des Parameters, zum Beispiel in einem <code>OPEN DIALOG</code> oder <code>SEND EVENT</code>-Statement, muss mit dem Format und der Länge des Parameters in der Inline-Definition des Dialogs übereinstimmen. ■ Wenn Sie <code>BY VALUE</code> angeben, wird ein Parameter direkt als Wert (<i>by value</i>) übergeben; Format und Länge brauchen nicht übereinzustimmen. Der Parameter im <code>OPEN DIALOG</code>- oder <code>SEND EVENT</code>-Statement muss mit dem Parameter des Dialogs datenübertragungskompatibel sein. <p>Beispiel für <code>BY VALUE</code>:</p> <table border="1" data-bbox="662 766 1477 1039"> <tr> <td data-bbox="662 766 1063 1039"> <pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre> </td> <td data-bbox="1063 766 1477 1039"> <pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre> </td> </tr> </table>	<pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre>
<pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre>		
<p>BY VALUE RESULT</p>	<p>Call-By-Value-Result-Modus:</p> <p>Während <code>BY VALUE</code> für die Übergabe eines Parameters an ein Subprogramm oder eine Subroutine gilt, bewirkt die Angabe von <code>BY VALUE RESULT</code> die Übergabe des Parameterwertes in beide Richtungen; d.h. der Parameterwert selbst wird vom aufrufenden Objekt an das Subprogramm bzw. die Subroutine übergeben, und bei der Rückkehr zum aufrufenden Objekt wird der Parameterwert selbst von dem Subprogramm bzw. der Subroutine an das aufrufende Objekt zurückgegeben.</p> <p>Wenn Sie <code>BY VALUE RESULT</code> verwenden, müssen Format und Länge der betreffenden Felder in beide Richtungen datenübertragungskompatibel sein.</p> <p>Anmerkung: <code>BY VALUE RESULT</code> kann nicht in Dialogen verwendet werden.</p>		
<p>OPTIONAL</p>	<p>Bei einem <i>ohne</i> <code>OPTIONAL</code> definierten Parameter (Standard) muss ein Wert vom aufrufenden Objekt übergeben werden.</p> <p>Bei einem <i>mit</i> <code>OPTIONAL</code> definierten Parameter muss ein Wert vom aufrufenden Objekt an diesen Parameter nicht unbedingt übergeben werden. Im aufrufenden Objekt wird die Notation <code>nX</code> benutzt, um</p>		

	<p>Parameter anzugeben, die übersprungen werden, d.h. für die keine Werte übergeben werden.</p> <p>Bei der SPECIFIED-Option können Sie zur Laufzeit ermitteln, ob ein optionaler Parameter definiert worden ist oder nicht.</p>
<i>parameter-handle-definition</i>	Siehe Abschnitt Parameter-Handle-Definition weiter unten.

Parameter-Handle-Definition

Syntax der *parameter-handle-definition*:

```
handle-name [(array-definition)] HANDLE OF { dialog-element-type }
                                         OBJECT
```

Syntax-Element-Beschreibung:

<i>handle-name</i>	Der der Handle zuzuweisende Name; es gelten die Namenskonventionen für Benutzervariablen; siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen.
HANDLE OF <i>dialog-element-type</i>	Der Dialog-Element-Typ. Mögliche Werte wie beim TYPE-Attribut. Weitere Informationen siehe <i>Dialog Elements</i> und <i>Attributes</i> in der <i>Dialog Component Reference</i> -Dokumentation.
HANDLE OF OBJECT	Wird benutzt in Zusammenhang mit NaturalX. Siehe <i>NaturalX</i> im <i>Leitfaden zur Programmierung</i> .
<i>array-definition</i>	Bei einer Array-Definition definieren Sie die untere und obere Grenze einer Dimension in einer Array-Dimension. Siehe Definition der Array-Dimension .

34

Definition von anwendungsunabhängigen Variablen

- Funktion 190
- Syntax-Beschreibung 191

Allgemeine Syntax von `DEFINE DATA INDEPENDENT`:

```
DEFINE DATA
  INDEPENDENT AIV-data-definition...
END-DEFINE
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Mit `DEFINE DATA INDEPENDENT` können Sie anwendungsunabhängige Variablen (application-independent variables, AIVs) definieren.

Eine anwendungsunabhängige Variable wird über ihren Namen referenziert, und ihr Inhalt wird von allen innerhalb einer Anwendung ausgeführten Programmierobjekten gemeinsam benutzt, die auf diesen Namen verweisen. Die Variable wird vom ersten ausgeführten Programmierobjekt zugewiesen, das diese Variable referenziert, und sie wird vom `LOGON`-Kommando oder einem `RELEASE VARIABLES`-Statement freigegeben.

Die optionale `INIT`-Klausel wird bei jedem ausgeführten Programmierobjekt ausgewertet, das diese Klausel enthält (nicht nur im Programmierobjekt, das die Variable zuweist).



Anmerkung: In einem RPC-Server werden anwendungsunabhängige Variable (AIVs) nicht implizit freigegeben, sondern bleiben über die RPC-Anforderung hinweg zugewiesen, weil verschiedene Clients Zugriff auf dieselben Variablen auf dem RPC-Server haben können. Das bedeutet, dass diese Variablen explizit mit einem `RELEASE VARIABLES`-Statement freigegeben werden müssen. Siehe *Application-Independent Variables* in der *Natural Remote Procedure Call*-Dokumentation.

Syntax-Beschreibung

INDEPENDENT <i>AIV-data-definition</i>	Das DEFINE DATA INDEPENDENT-Statement kann zur Definition einer einzelnen oder mehrerer anwendungsunabhängiger Variablen (AIVs) benutzt werden. Für jede AIV gilt die weiter unten gezeigte Syntax.
END-DEFINE	Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

AIV Data Definition

<i>level</i>	$\left\{ \begin{array}{l} \textit{variable-definition} \\ \textit{redefinition} \\ \textit{handle-definition} \end{array} \right\}$
--------------	---

Syntax-Element-Beschreibung:

<i>level</i>	Eine applikationsunabhängige Variable muss auf Level 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.
<i>variable-definition</i>	Eine Variablen-Definition wird zur Definition eines einzelnen Feldes oder einer einzelnen Variable benutzt, die einen Wert (Skalar) oder mehrere Werte (Array) haben kann. Siehe Definition von Variablen . Anmerkung: Der Name einer applikationsunabhängigen Variable muss mit einem Plus-Zeichen (+) anfangen.
<i>redefinition</i>	Mit einer <i>redefinition</i> können Sie eine applikationsunabhängige Variable in ein oder mehrere Sub-Felder unterteilen. Siehe Redefinition . Die aus der Redefinition resultierenden Sub-Felder dürfen keine applikationsunabhängigen Variablen sein, d.h. ihr Name darf nicht mit einem Plus-Zeichen (+) anfangen. Diese Felder werden als lokale Variablen behandelt.
<i>handle-definition</i>	Eine Handle identifiziert ein Dialog-Element im Code und wird in Handle-Variablen gespeichert. Siehe Handle Definition .



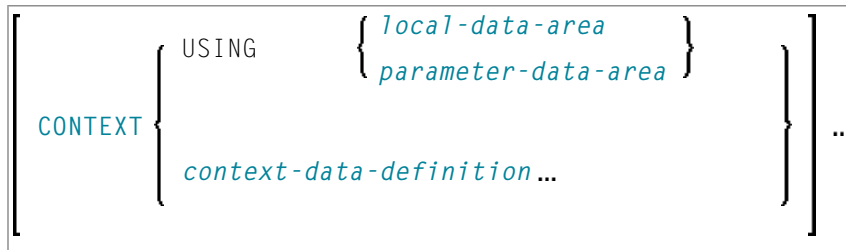
Anmerkung: Das erste Zeichen des Namens muss ein Plus-Zeichen (+) sein. Es gelten die Regeln für Natural-Variablenamen, siehe *Namen von Benutzervariablen* in der Dokumentation *Natural Studio benutzen*.

35

Definition von Kontext-Variablen für den Natural RPC

▪ Funktion	194
▪ Einschränkungen	195
▪ Syntax-Beschreibung	195

Allgemeine Syntax von DEFINE DATA CONTEXT:



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Gehört zur Funktionsgruppe: [Natural Remote Procedure Call](#)

Funktion

Das DEFINE DATA CONTEXT-Statement wird im Zusammenhang mit dem Natural Remote Procedure Call (RPC) verwendet. Es dient dort zur Definition von als Kontext-Variablen bekannte Variablen, die für mehrere Subprogramme auf einem externen (Remote-)Rechner innerhalb einer Konversation zur Verfügung stehen sollen, ohne dass Sie die Variablen explizit als Parameter mit den entsprechenden CALLNAT-Statements übergeben müssen.

Eine Kontext-Variablen wird über ihren Namen referenziert, und ihr Inhalt wird von allen in einer Konversation ausgeführten Programmierobjekten gemeinsam benutzt, die auf diesen Namen verweisen. Die Variable wird vom ersten ausgeführten Programmierobjekt zugewiesen, das die Definition der Variable enthält, und wird freigegeben, wenn die Konversation beendet ist.

Kontext-Variablen können auch in einem nicht-konversationellen CALLNAT benutzt werden. In diesem Fall sind die Kontext-Variablen nur während eines einzigen Aufrufs dieses CALLNAT vorhanden, aber die Variablen können mit allen ihren aufgerufenen Callees gemeinsam benutzt werden.

Eine Kontext-Variablen wird nicht mit Subprogrammen gemeinsam benutzt, die innerhalb der Konversation aufgerufen werden. Wenn ein solches Subprogramm oder eines seiner Callees eine Kontext-Variablen referenziert, wird ein separater Speicherbereich für diese Variable zugewiesen.

Die optionale INIT-Klausel wird bei jedem ausgeführten Programmierobjekt ausgewertet, das diese Klausel enthält (nicht nur im Programmierobjekt, das die Variable zuweist). Bei globalen Variablen funktioniert INIT anders.

Weitere Informationen siehe *Defining a Conversation Context* in der *Natural Remote Procedure Call (RPC)*-Dokumentation.

Einschränkungen

Eine Kontext-Variablen muss auf Level 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.

Syntax-Beschreibung

USING <i>local-data-area</i>	<p>Eine Local Data Area (LDA) enthält Datenelemente, die in einem einzelnen Natural-Modul benutzt werden sollen. Sie können mehr als eine Data Area referenzieren; in diesem Fall müssen Sie die reservierten Wörter <code>CONTEXT</code> und <code>USING</code> wiederholen, zum Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 10px;">DEFINE DATA CONTEXT USING DATX_L CONTEXT USING DATX_P ... END-DEFINE ;</pre> <p>Weitere Informationen siehe <i>Felder in einer separaten Data Area definieren</i> im Leitfaden zur Programmierung.</p>
USING <i>parameter-data-area</i>	<p>Eine Parameter Data Area enthält Datenelemente, die als Parameter in einem Subprogramm, einer externen Subroutine oder in einem Dialog benutzt werden.</p>
<i>context-data-definition</i>	<p>Kontext-Daten können auch direkt innerhalb eines Programms oder einer Routine definiert werden. Bei einer direkten Daten-Definition gilt die weiter unten gezeigte Syntax.</p>
END-DEFINE	<p>Das für Natural reservierte Wort <code>END-DEFINE</code> muss zum Beenden des <code>DEFINE DATA</code>-Statements benutzt werden.</p>

Kontextdaten-Definition

Kontext-Daten können direkt innerhalb eines Programms oder einer Routine definiert werden. Bei einer direkten Daten-Definition gilt die folgende Syntax:

```

level {
  variable-definition
  redefinition
  handle-definition
}
    
```

Weitere Informationen siehe *Felddefinitionen im DEFINE DATA-Statement im Leitfaden zur Programmierung*.

<i>level</i>	Eine anwendungsunabhängige Variable muss auf Stufe (Level) 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.
<i>variable-definition</i>	Eine Variablen-Definition wird zur Definition eines einzelnen Feldes oder einer einzelnen Variable benutzt, die einen Wert (Skalar) oder mehrere Werte (Array) haben kann. Siehe Definition von Variablen . Anmerkung: Die <code>CONSTANT</code> -Klausel darf in diesem Zusammenhang nicht benutzt werden.
<i>redefinition</i>	Mit einer Redefinition redefinieren Sie eine Gruppe, ein View, ein DDM-Feld oder ein einzelnes Feld oder eine einzelne Variable (d.h. ein Skalar oder ein Array). Siehe Redefinition .
<i>handle-definition</i>	Eine Handle identifiziert ein Dialog-Element im Code und wird in Handle-Variablen gespeichert. Siehe Handle-Definition .



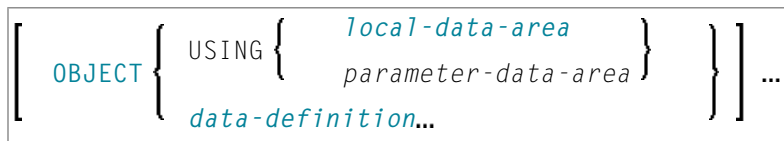
Anmerkung: Die sich aus einer Redefinition ergebenden Felder werden nicht als eine Kontext-Variable angesehen. Diese Felder werden als lokale Variablen behandelt.

36

Definition von NaturalX-Objekten

- Funktion 198
- Syntax-Beschreibung 198

Allgemeine Syntax von DEFINE DATA OBJECT:



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

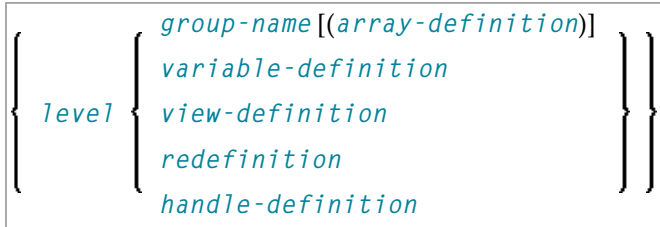
Das DEFINE DATA OBJECT-Statement wird benutzt in einem Subprogramm oder einer Klasse in Zusammenhang mit NaturalX. Weitere Informationen, siehe *NaturalX-Dokumentation im Leitfaden zur Programmierung*.

Syntax-Beschreibung

USING <i>local-data-area</i>	<p>Eine Local Data Area (LDA) enthält Datenelemente, die in einem einzelnen Natural-Modul benutzt werden sollen. Sie können mehr als eine Data Area referenzieren; in diesem Fall müssen Sie die reservierten Wörter OBJECT und USING wiederholen, zum Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 10px;"> DEFINE DATA OBJECT USING DATX_L OBJECT USING DATX_P ... END-DEFINE ; </pre> <p>Weitere Informationen siehe <i>Felddefinitionen im DEFINE DATA-Statement im Leitfaden zur Programmierung</i>.</p>
USING <i>parameter-data-area</i>	<p>Eine mit DEFINE DATA OBJECT definierte Data Area kann eine Parameter Data Area (PDA) sein. Wenn Sie eine PDA als eine Object Data Area benutzen, können Sie sich die zusätzliche Mühe ersparen, eine Object Data Area zu erstellen, die dieselbe Struktur wie die PDA hat.</p>
<i>data-definition</i>	<p>Daten können auch direkt mit der im Abschnitt Direkte Datendefinition weiter unten angegebenen Syntax definiert werden.</p>
END-DEFINE	<p>Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.</p>

Direkte Datendefinition

Daten können auch mit der folgenden Syntax direkt definiert werden:



Weitere Informationen siehe *Felddefinitionen im DEFINE DATA-Statement im Leitfaden zur Programmierung*.

<i>level</i>	<p>Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte 0 ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet.</p> <p>Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren. Bei manchen Statements (CALL, CALLNAT, RESET, WRITE usw.) können Sie den Gruppennamen als Aufrufnamen angeben, um die in der Gruppe enthaltenen Felder zu referenzieren.</p> <p>Eine Gruppe kann aus weiteren Gruppen bestehen. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden. Eine View-Definition muss immer auf Level 1 definiert werden.</p>
<i>group-name</i>	<p>Der Name einer Gruppe. Der Name muss den Regeln zur Definition eines Natural-Variablenamens entsprechen. Siehe auch die folgenden Abschnitte:</p> <ul style="list-style-type: none"> ■ <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen. ■ <i>Datenstrukturen qualifizieren</i> im <i>Leitfaden zur Programmierung</i>.
<i>array-definition</i>	<p>Mit <i>array-definition</i> definieren Sie die Unter- und Obergrenze einer Dimension in einer Array-Definition. Siehe auch Definition einer Array-Dimension.</p>
<i>variable-definition</i>	<p>Eine <i>variable-definition</i> wird zur Definition eines einzelnen Feldes (Skalar) oder einer einzelnen Variablen (Array) verwendet. Siehe Variablen-Definition.</p>
<i>view-definition</i>	<p>Eine <i>view-definition</i> wird benutzt, um ein View mit Bestandteilen aus einem Datendefinitionsmodul (DDM) zu definieren. Siehe View Definition.</p>
<i>redefinition</i>	<p>Eine <i>redefinition</i> kann zur Redefinition einer Gruppe, eines Views, eines DDM-Felds oder eines einzelnen Feldes oder einer einzelnen Variable benutzt werden (d.h. Skalar oder Array). Siehe Redefinition.</p>

handle-definition

Eine Handle identifiziert ein Dialog-Element im Code und wird in Handle-Variablen gespeichert. Siehe [Handle-Definition](#).

37

Definition von Variablen

- Funktion 202
- Syntax-Beschreibung 203

Eine *variable-definition* ist in den Statements `DEFINE DATA LOCAL`, `DEFINE DATA INDEPENDENT`, `DEFINE DATA CONTEXT` und `DEFINE DATA OBJECT` möglich. Sie dient zur Definition eines einzelnen Feldes oder einer einzelnen Variablen, die aus einem einzigen Wert (*scalar-definition*) oder mehreren Werten (*array-definition*) bestehen kann:

```
{ <scalar-definition>
  <array-definition> }
```

<scalar-definition>

```
variable-name { (format-length)
  ( ( { A } ) ) DYNAMIC } [ { { CONSTANT } } init-definition ] [emhdpm]
  ( { U } )
  ( { B } )
```

<array-definition>

```
variable-name { (format-length/ array-definition)
  ( ( { A } ) /array-definition ) DYNAMIC } [ { CONSTANT } } array-init-definition ] [emhdpm]
  ( { U } )
  ( { B } )
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Eine *variable-definition* wird zur Definition eines einzelnen Feldes oder einer einzelnen Variablen verwendet, die aus einem einzigen Wert (Skalar) oder mehreren Werten (Array) bestehen kann.

Syntax-Beschreibung

<i>variable-name</i>	<i>variable-name</i> ist der der Variable zuzuweisende Name. Es gelten die Regeln für Natural-Variablenamen. Bei <code>DEFINE DATA INDEPENDENT</code> muss der Variablenname mit einem Plus-Zeichen (+) beginnen. Informationen zu Namenskonventionen für Benutzervariablen siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen.
<i>format-length</i>	Format und Länge des Feldes. Informationen zu Format/Längen-Definitionen von Benutzervariablen, siehe <i>Format und Länge von Benutzervariablen</i> im Leitfaden zur Programmierung.
A oder B oder U	Datentyp: Alphanumerisch, Binär oder Unicode für dynamische Variablen.
<i>array-definition</i>	Mit <i>array-definition</i> definieren Sie die Unter- und Obergrenze einer Dimension in einer Array-Definition. Siehe Definition einer Array-Dimension .
DYNAMIC	Ein Feld kann als dynamisch definiert werden. Weitere Informationen zur Verarbeitung von dynamischen Variablen siehe <i>Dynamische und große Variablen</i> benutzen.
CONSTANT	<p>Die Variable (bzw. das Array) soll als eine Namens-Konstante behandelt werden. Der bzw. die zugewiesene(n) Konstanten-Wert bzw. -Werte wird jedesmal benutzt, wenn die Variable bzw. das Array referenziert wird. Der bzw. die zugewiesene(n) Wert(e) kann bzw. können bei der Ausführung des Programms nicht geändert werden.</p> <p>Siehe auch <i>Felder definieren, Benutzerkonstanten Namens-Konstanten definieren</i> im Leitfaden zur Programmierung.</p> <p>Anmerkung: Aus Gründen der internen Handhabung ist es nicht zulässig, Variablen-Definitionen und Konstanten-Definitionen innerhalb einer Gruppen-Definition miteinander zu vermischen; d.h. eine Gruppe kann entweder nur Variablen oder nur Konstanten enthalten. Die <code>CONSTANT</code>-Klausel darf nicht mit <code>DEFINE DATA INDEPENDENT</code> und <code>DEFINE DATA CONTEXT</code> benutzt werden. Die <code>CONSTANT</code>-Klausel kann nicht mit X-Arrays benutzt werden.</p>
INIT	<p>Der Variablen bzw. dem Array soll ein Ausgangswert zugewiesen werden. Dieser Wert wird auch benutzt, wenn diese Variable bzw. dieses Array in einem <code>RESET INITIAL</code>-Statement referenziert wird.</p> <p>Wenn <code>INIT</code> nicht angegeben ist, wird ein Feld mit einem standardmäßigen Ausgangswert je nach seinem Format initialisiert (siehe Tabelle Standard-Ausgangswerte).</p> <p>Siehe auch <i>Felder definieren, Ausgangswerte</i> im Leitfaden zur Programmierung.</p> <p>Anmerkung: Bei <code>DEFINE DATA INDEPENDENT</code> und <code>DEFINE DATA CONTEXT</code> wird die <code>INIT</code>-Klausel bei jedem ausgeführten Programmierobjekt ausgewertet, das diese Klausel enthält (nicht nur im Programmierobjekt, das die Variable</p>

	zuweist). INIT funktioniert andes für globale Variablen. Die INIT-Klausel kann nicht für X-Arrays benutzt werden.
<i>init-definition</i>	Mit der Option <i>init-definition</i> definieren Sie die Ausgangswerte oder Konstanten-Werte für eine Variable. Siehe Definition von Ausgangswerten .
<i>array-init-definition</i>	Bei <i>array-init-definition</i> definieren Sie die Ausgangswerte oder Konstanten-Werte für ein Array. Siehe Ausgangswerte/Konstanten-Werte für ein Array .
<i>emhdpm</i>	Mit dieser Option können zusätzliche Parameter definiert werden, die für ein Feld oder eine Variable gelten sollen. Siehe Parameter EM, HD, PM für Feld/Variable .

Standard-Ausgangswerte

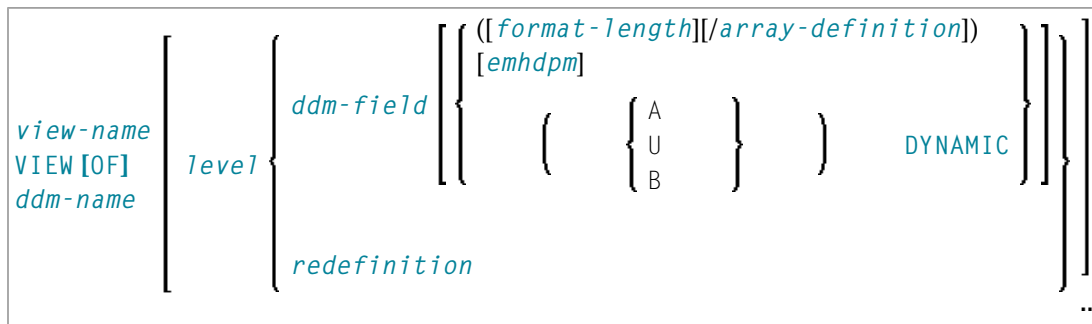
Format	Standard-Ausgangswert
B, F, I, N, P	0
A, U	(leer)
L	FALSE
D	D' '
T	T'00:00:00'
C	(AD=D)
GUI Handle	NULL-HANDLE
Object Handle	NULL-HANDLE

Als dynamisch (DYNAMIC) deklarierte Felder haben keinen Ausgangswert, weil ihre Feldlänge standardmäßig Null ist.

38 View-Definition

▪ Funktion	206
▪ Syntax-Beschreibung	206

Die mit **den Statements** `DEFINE DATA LOCAL` und `DEFINE DATA OBJECT` benutzte Option *view-definition* hat die folgende Syntax:



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Eine *view-definition* stellt einen Ausschnitt eines Datendefinitionsmoduls (DDM) dar.



Anmerkung: In einer Parameter Data Area ist eine *view-definition* nicht erlaubt.

Weitere Informationen siehe Abschnitt *Daten in einer Adabas-Datenbank aufrufen* im Leitfaden zur Programmierung und dort insbesondere die folgenden Themen:

- *Datendefinitionsmodule – DDMs*
- *Datenbank-Arrays*
- *Datenbank-View definieren*

Syntax-Beschreibung

<i>view-name</i>	Der Name, den der View erhalten soll. Es gelten die Namenskonventionen für Natural-Variablen. Siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio</i> benutzen.
VIEW [OF] <i>ddm-name</i>	Der Name des DDMs, aus dem der View gebildet wird.
<i>level</i>	Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte 0 ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil

	<p>einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet.</p> <p>Durch die Definition einer Gruppe (die auch nur aus einem Feld bestehen kann) ist es möglich, durch Angabe lediglich des Gruppennamens eine ganze Reihe von aufeinanderfolgenden Feldern gleichzeitig zu referenzieren. Bei manchen Statements (CALL, CALLNAT, RESET, WRITE usw.) können Sie den Gruppennamen als Kurznamen für die Referenzierung der in der Gruppe enthaltenen Felder angeben.</p> <p>Eine Gruppe kann ihrerseits Teil einer anderen Gruppe sein. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.</p>
<i>dgm-field</i>	<p>Der im verwendeten DDM definierte Name eines Feldes.</p> <p>Bei der Definition eines Views für ein HISTOGRAM-Statement darf dieser View lediglich den Deskriptor enthalten, den das HISTOGRAM-Statement benutzt.</p>
<i>redefinition</i>	<p>Eine <i>redefinition</i> kann zur Redefinition einer Gruppe, eines Views, eines DDM-Felds oder eines einzelnen Feldes oder einer einzelnen Variable benutzt werden (d.h. Skalar oder Array). Siehe <i>Redefinition</i>.</p>
<i>format-length</i>	<p>Format und Länge des definierten Feldes. Werden diese Angaben nicht gemacht, wird die Format-/Längendefinition aus dem DDM übernommen.</p> <p>Im Structured Mode muss die Definition von Format und Länge (wenn angegeben) dieselbe wie die vom DDM sein.</p> <p>Im Reporting Mode muss die Format-/Längendefinition kompatibel mit der im DDM sein.</p>
A, U or B	<p>Datentyp: Alphanumerisch (A), Unicode (U) oder binär (B) für dynamische Variablen.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Bei Adabas für Großrechner steht das Format U für LA-Felder (Länge: <= 16381 Bytes), aber nicht für LB-Felder (Länge: <= 1GB) zur Verfügung 2. Format B kann nicht bei Adabas verwendet werden.
<i>array-definition</i>	<p>Abhängig vom benutzten Modus müssen Arrays (Periodengruppenfelder, multiple Felder) eventuell Informationen über ihre Ausprägungen aufnehmen. Siehe den Abschnitt <i>Array-Definition in einem View</i> weiter unten.</p>
<i>emhdpm</i>	<p>Mit dieser Option können zusätzliche Parameter definiert werden, die für ein Feld oder eine Variable gelten sollen. Siehe <i>Parameter EM, HD, PM für Feld/Variable</i>.</p>
DYNAMIC	<p>Definiert ein View-Feld als dynamisch. Weitere Informationen zur Verarbeitung von dynamischen Variablen siehe den Abschnitt <i>Dynamische und große Variablen benutzen</i>.</p>

Array-Definition in einem View

Abhängig von dem benutzten Programmiermodus müssen Arrays, d.h. Periodengruppenfelder (PE), multiple Felder (MU), in Abhängigkeit vom verwendeten Programmiermodus eventuell Informationen über ihre Ausprägungen aufnehmen.

- [Array-Definition in einem View im Structured Mode](#)
- [Array-Definition in einem View im Reporting Mode](#)

Array-Definition in einem View im Structured Mode

Wenn ein Feld in einem View benutzt wird, das einen Array darstellt, gilt Folgendes:

- Ein Indexwert muss für MU/PE-Felder angegeben werden.
- Wenn kein(e) Format/Länge angegeben ist, werden die Werte aus dem DDM genommen.
- Ist ein(e) Format/Länge angegeben, muss die Angabe mit der im DDM übereinstimmen.

Datenbank-spezifische Anmerkungen zum Structured Mode:

Adabas:	Wenn (in einem DDM definierte) MU/PE-Felder in einem View benutzt werden sollen, müssen diese Felder eine Array-Index-Angabe enthalten. Für ein MU-Feld oder ein normales PE-Feld geben Sie einen eindimensionalen Index-Bereich an, z.B. (1:10). Für ein MU-Feld in einer PE-Gruppe geben Sie einen zweidimensionalen Index-Bereich an, z.B. (1:10,1:5).		
Tamino:	DDM-Definition	zulässig	nicht zulässig
	A(*:X2)	A(*:Y2) Y2=<X2 A(Y1:Y2) Y2>Y1 Y2=<X2 A(Z:Z+Y) Y>=0	A(*:*) A(Y1:*)
	A(X1:*)	A(Y1:*) Y1>=X1 A(Y1:Y2) Y2>=X1, Y1>=X1 A(Z:Z+Y) Y>=0	A(*:*) A(*:Y2)
	A(X1:X2)	A(Y1:Y2) Y2<Y1 A(Z:Z+Y) 0=<Y>=X2-X1+1	A(*:*) A(Y1:*) A(*:Y2)

Beispiele für Structured Mode:

```

DEFINE DATA LOCAL
1 EMP1 VIEW OF EMPLOYEES
  2 NAME(A20)
  2 ADDRESS-LINE(A20 / 1:2)

1 EMP2 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(1:2)

1 EMP3 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(2)

1 #K (I4)
1 EMP4 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(#K:#K+1)
END-DEFINE
END

```

Array-Definition in einem View im Reporting Mode

In diesem Modus gelten dieselben Regeln wie für Structured Mode. Es gibt aber zwei Ausnahmen:

- Ein Indexwert muss nicht angegeben werden. In diesem Fall wird der Index-Bereich für die fehlenden Dimensionen auf (1:1) gesetzt.
- Die Format/Längenangabe kann sich von der Angabe im DDM unterscheiden.

Beispiele:

```

DEFINE DATA LOCAL
1 EMP1 VIEW OF EMPLOYEES
  2 NAME(A30)
  2 ADDRESS-LINE(A35 / 5:10)

1 EMP2 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(A40)          /* ADDRESS LINE (1:1) IS ASSUMED

1 EMP3 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE              /* ADDRESS LINE (1:1) IS ASSUMED

1 #K (I4)
1 EMP4 VIEW OF EMPLOYEES
  2 NAME

```

View-Definition

```
2 ADDRESS-LINE(#K:#K+1)
END-DEFINE
END
```

39

Redefinition

- Funktion 212
- Einschränkungen 212
- Syntax-Beschreibung 213

Die *redefinition*-Option steht in den folgenden Statements zur Verfügung: `DEFINE DATA LOCAL`, `DEFINE DATA PARAMETER`, `DEFINE DATA INDEPENDENT`, `DEFINE DATA CONTEXT` und `DEFINE DATA OBJECT`.

Die *redefinition*-Option hat die folgende Syntax:

```
REDEFINE field-name { level { rgroup  
                           rfield(format-length [array-definition]) } } ...  
                           FILLER nX
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Die *redefinition*-Option kann zur Redefinition einer Gruppe, eines Views, eines DDM-Feldes oder für ein einzelnes Feld oder für eine einzelne Variable (d.h. Skalar oder Array) benutzt werden.



Anmerkungen:

1. Eine Redefinition eines Views oder DDM-Feldes für eine *parameter-data-definition* ist nicht möglich.
2. Unicode-Felder sollten nicht als alphanumerische (A) oder numerische (N) Felder redefiniert werden.

Siehe auch *Felder redefinieren* im *Leitfaden zur Programmierung*.

Einschränkungen

- Handles, X-Arrays und dynamische Variablen können nicht redefiniert werden und können nicht in einer Redefinition-Klausel enthalten sein.
- Eine Gruppe, die eine Handle enthält, ein X-Array oder eine dynamische Variable können nur bis zu dem betreffenden Element – aber nicht einschließlich oder darüber hinaus – redefiniert werden.

Syntax-Beschreibung

<i>field-name</i>	Der Name der Gruppe, des Views, DDM-Feldes oder einzelnen Feldes, der/die/das redefiniert werden soll.
<i>level</i>	Dies ist eine ein- oder zweistellige Zahl im Bereich von 01 bis 99 (die vorangestellte Null ist nicht erforderlich), die in Verbindung mit der Gruppierung von Feldern verwendet wird. Felder mit einer Level-Nummer von 02 an aufwärts werden als Teil einer unmittelbar vorangehenden Gruppe mit einer jeweils nächstniedrigeren Level-Nummer betrachtet.
<i>rgroup</i>	Der Name der Gruppe, die sich aus der Redefinition ergibt. Anmerkung: Bei einer Redefinition innerhalb einer <i>view-definition</i> darf für <i>rgroup</i> kein Name vergeben werden, der schon als Feldname im zugrundeliegenden DDM existiert.
<i>rfield</i>	Der Name des Feldes, das sich aus der Redefinition ergibt. Anmerkung: Bei einer Redefinition innerhalb einer <i>view-definition</i> darf für <i>rfield</i> kein Name vergeben werden, der schon als Feldname im zugrundeliegenden DDM existiert.
<i>format-length</i>	Format und Länge von (<i>rfield</i>).
<i>array-definition</i>	Bei einer Array-Definition definieren Sie die Unter- und Obergrenze einer Dimension in einer Array-Definition. Siehe den Abschnitt <i>Definition der Array-Dimension</i> .
FILLER <i>nX</i>	Mit dieser Notation können Sie in dem Feld, das redefiniert wird, <i>n</i> Füllbytes — d.h. Segmente, die nicht benutzt werden sollen — definieren. Die Definition von nachgestellten Füllbytes ist optional.

Beispiele für die Benutzung von REDEFINE

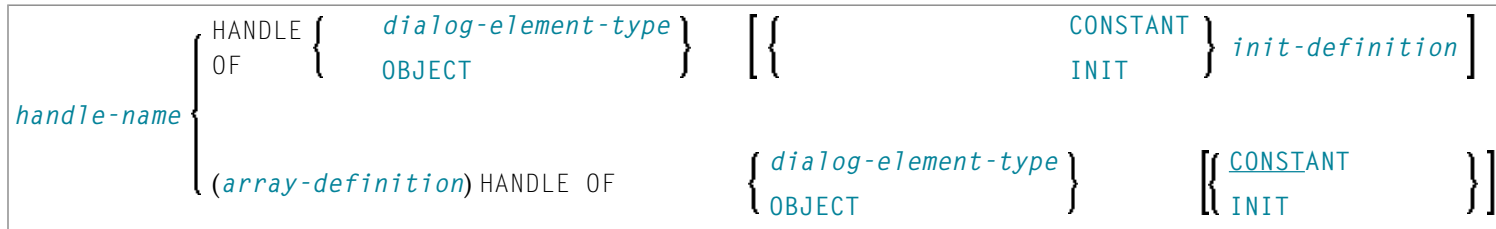
Beispiel 1:	Beispiel 2:	Beispiel 3:
<pre> DEFINE DATA LOCAL 01 #VAR1 (A15) 01 #VAR2 02 #VAR2A (N4.1) INIT <0> 02 #VAR2B (P6.2) INIT <0> 01 REDEFINE #VAR2 02 #VAR2RD (A10) END-DEFINE ... </pre>	<pre> DEFINE DATA LOCAL 01 MYVIEW VIEW OF STAFF 02 NAME 02 BIRTH 02 REDEFINE BIRTH 03 BIRTH-YEAR (N4) 03 BIRTH-MONTH (N2) 03 BIRTH-DAY (N2) END-DEFINE ... </pre>	<pre> DEFINE DATA LOCAL 1 #FIELD (A12) 1 REDEFINE #FIELD 2 #RFIELD1 (A2) 2 FILLER 2X 2 #RFIELD2 (A2) 2 FILLER 4X 2 #RFIELD3 (A2) END-DEFINE ... </pre>

40 Handle-Definition

- Funktion 216
- Syntax-Beschreibung 217

Die Handle-Definition steht in den folgenden Statements zur Verfügung: `DEFINE DATA LOCAL`, `DEFINE DATA OBJECT`, `DEFINE DATA PARAMETER`, `DEFINE DATA INDEPENDENT` und `DEFINE DATA CONTEXT`.

Die *handle-definition* hat die folgende Syntax:



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Eine Handle identifiziert ein Dialog-Element im Code und wird in Handle-Variablen gespeichert. Weitere Informationen siehe Abschnitt *NaturalX* im *Leitfaden zur Programmierung*.

Die Handle-Definition in dem `DEFINE DATA`-Statement wird automatisch bei der Erstellung eines Dialog-Elements oder Dialogs generiert.

Nachdem Sie eine Handle definiert haben, können Sie den Handle-Namen in einem beliebigen Statement benutzen, um Attributwerte für den definierten *dialog-element-type* abzufragen, zu setzen oder zu ändern.

Siehe auch *Event-Driven Programming Techniques* (Ereignisgesteuerte Programmieretechniken) im *Leitfaden zur Programmierung*.

Beispiele für Handle-Definition:

```
1 #SAVEAS-MENUITEM HANDLE OF MENUITEM
1 #OK-BUTTON (1:10) HANDLE OF PUSHBUTTON
```

Syntax-Beschreibung

<i>handle-name</i>	Der der Handle zuzweisende Name. Es gelten die Namenskonventionen für Benutzervariablen. Siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i> .
HANDLE OF <i>dialog-element-type</i>	Der Typ des Dialogelements. Mögliche Werte wie beim TYPE-Attribut. Weitere Informationen siehe <i>Dialog Elements</i> und <i>Attributes</i> in der <i>Dialog Component Reference</i> -Dokumentation.
HANDLE OF OBJECT	Wird im Zusammenhang mit NaturalX benutzt, siehe Abschnitt <i>NaturalX</i> im <i>Leitfaden zur Programmierung</i> .
CONSTANT	<p>Die Variable bzw. das Array ist als Namens-Konstante zu behandeln. Der/Die zugewiesene(n) Konstanten-Wert(e) wird/werden jedesmal benutzt, wenn die Variable/das Array referenziert wird. Der/die zugewiesene(n) Wert(e) können nicht während der Ausführung des Programms geändert werden.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Aus Gründen der internen Behandlung ist es nicht zulässig, Definitionen von Variablen und Definitionen von Konstanten innerhalb einer Gruppen-Definition miteinander zu vermischen, d.h. eine Gruppe kann entweder nur Variablen oder nur Konstanten enthalten. 2. Die CONSTANT-Klausel darf nicht bei den Statements DEFINE DATA INDEPENDENT und DEFINE DATA CONTEXT verwendet werden.
INIT	<p>Der Variable bzw. dem Array soll ein Ausgangswert zugewiesen werden. Dieser Wert wird auch benutzt, wenn diese Variable/dieses Array in einem RESET INITIAL-Statement referenziert wird.</p> <p>Anmerkung: Bei den Statements DEFINE DATA INDEPENDENT und DEFINE DATA CONTEXT wird die INIT-Klausel in jedem ausgeführten Programmierobjekt ausgewertet, das diese Klausel enthält (nicht nur im Programmierobjekt, das die Variable zuweist). Bei globalen Variablen funktioniert INIT anders.</p>
<i>init-definition</i>	Mit der Option <i>init-definition</i> definieren Sie die Ausgangswerte oder Konstanten-Werte für eine Variable. Siehe <i>Definition von Ausgangswerten</i> .
<i>array-definition</i>	Mit <i>array-definition</i> definieren Sie die Untergrenze und Obergrenze einer Dimension in einer Array-Definition. Siehe auch den Abschnitt <i>Definition von Array-Dimensionen</i> weiter unten.
<i>array-init-definition</i>	Dem Array soll ein Ausgangswert zugewiesen werden. Dieser Wert wird auch benutzt, wenn dieses Array in einem RESET INITIAL -Statement referenziert wird.

41

Definition von Array-Dimensionen

- Funktion 220
- Syntax-Beschreibung 220

Die Definition von Array-Dimensionen ist im Statement `DEFINE DATA OBJECT` möglich und außerdem in der *variable-definition*-Option bei folgenden Statements: `DEFINE DATA LOCAL`, `DEFINE DATA INDEPENDENT`, `DEFINE DATA CONTEXT`, `DEFINE DATA OBJECT`. Darüber hinaus ist sie im `DEFINE FUNCTION`-Statement möglich.

Die *array-dimension-definition* hat die folgende Syntax:

```
{bound[:bound]},... 3
```

Dieses Kapitel behandelt folgende Themen:

Funktion

Mit der *array-dimension-definition* können Sie bei einer Array-Definition die Unter- und Obergrenze (*bound*) einer Dimension festlegen.

Sie können bis zu 3 Dimensionen für ein Array definieren.

Siehe auch *Arrays* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung

<i>bound</i>	<p>Als Grenze (<i>bound</i>) kann eines der folgenden Elemente verwendet werden:</p> <ul style="list-style-type: none">■ eine numerische Ganzzahl-Konstante■ eine vorher definierte Namens-Konstante■ (bei Datenbank-Arrays) eine vorher definierte Benutzervariable■ ein Stern (*); dieser definiert einen erweiterbaren <i>bound</i>, auch bekannt als X-Array. <p>Wenn nur ein Bound angegeben ist, stellt der Wert die Obergrenze dar, und die Untergrenze wird als 1 angenommen.</p>
--------------	--

X-Arrays

Wenn mindestens eine Grenze (*bound*) in mindestens einer Dimension eines Arrays als erweiterbar angegeben wird, bezeichnet man dieses Array als X-Array (eXtensible Array). Nur eine Grenze (entweder oberer oder unterer) in einer Dimension kann erweiterbar sein, aber nicht beide. Mehrdimensionale Arrays können eine Mischung von konstanten und erweiterbaren Grenzen haben, z.B. `#a(1:100, 1:*)`.

Beispiel:

```
DEFINE DATA LOCAL
1 #ARRAY1(I4/1:10)
1 #ARRAY2(I4/10)
1 #X-ARRAY3(I4/1:*)
1 #X-ARRAY4(I4/*,1:5)
1 #X-ARRAY5(I4/*:10)
1 #X-ARRAY6(I4/1:10,100:*,*:1000)
END-DEFINE
```

Die folgende Tabelle enthält eine Übersicht über die Grenzen der Arrays aus dem obigen Programm.

	Dimension 1		Dimension 2		Dimension 3	
	Untere Grenze	Obere Grenze	Untere Grenze	Obere Grenze	Untere Grenze	Obere Grenze
#ARRAY1	1	10	-	-	-	-
#ARRAY2	1	10	-	-	-	-
#X-ARRAY3	1	erweiterbar	-	-	-	-
#X-ARRAY4	1	erweiterbar	1	5	-	-
#X-ARRAY5	erweiterbar	10	-	-	-	-
#X-ARRAY6	1	10	100	erweiterbar	erweiterbar	1000

Beispiele für Array-Definitionen:

```
#ARRAY2(I4/10) /* a one-dimensional array with 10 occurrences (1:10)
#X-ARRAY4(I4/*,1:5) /* a two-dimensional array
#X-ARRAY6(I4/1:10,100:*,*:1000) /* a three-dimensional array
```

Variable Arrays in einer Parameter Data Area:

In einer Parameter Data Area können Sie ein Array mit einer variablen Anzahl von Ausprägungen angeben. Dies erfolgt mittels der Index-Notation 1:V.

Beispiel 1: #ARR01 (A5/1:V)

Beispiel 2: #ARR02 (I2/1:V,1:V)

Ein Parameter-Array, das eine variable Index-Notation 1:V enthält, kann nur redefiniert werden in der Länge

- seiner elementaren Feldlänge, wenn der Index 1:V ganz rechts steht, zum Beispiel:

#ARR(A6/1:V) kann bis zu einer Länge von 6 Bytes redefiniert werden;

#ARR(A6/1:2,1:V) kann bis zu einer Länge von 6 Bytes redefiniert werden;

`#ARR(A6/1:2,1:3,1:V)` kann bis zu einer Länge von 6 Bytes redefiniert werden.

- des Produkts der ganz rechts stehenden festen Ausprägungen und der elementaren Feldlänge, zum Beispiel:

`#ARR(A6/1:V,1:2)` kann bis zu einer Länge von $2*6 = 12$ Bytes redefiniert werden;

`#ARR(A6/1:V,1:3,1:2)` kann bis zu einer Länge von $3*2*6 = 36$ Bytes redefiniert werden;

`#ARR(A6/1:2,1:V,1:3)` kann bis zu einer Länge von $3*6 = 18$ Bytes redefiniert werden.

Eine variable Index-Notation `1:V` darf nicht in einem Redefinitionsblock verwendet werden.

Beispiel:

```
DEFINE DATA PARAMETER
  1 #ARR(A6/1:V)
  1 REDEFINE #ARR
    2 #R-ARR(A1/1:V) /* (1:V) is not allowed in a REDEFINE block
END-DEFINE
```

Da die Anzahl der Ausprägungen zur Kompilierungszeit nicht bekannt ist, darf es nicht mit der Index-Notation (*) in den Statements `INPUT`, `WRITE`, `READ WORK FILE` und `WRITE WORK FILE` referenziert werden. Die Index-Notation (*) kann entweder für alle Dimensionen oder für keine Dimension benutzt werden.

Gültige Beispiele:

```
#ARR01 (*)
#ARR02 (*,*)
#ARR01 (1)
#ARR02 (5,#FIELDX)
#ARR02 (1,1:3)
```

Ungültiges Beispiel:

```
#ARRAYY (1,*) /* not allowed
```

Um Laufzeitfehler zu vermeiden, sollte die maximale Anzahl der Ausprägungen eines solchen Arrays über einen anderen Parameter an das Subprogramm/die Subroutine übergeben werden. Als Alternative dazu können Sie die Systemvariable `*OCCURRENCE` benutzen.



Anmerkungen:

1. Wenn eine einen Index $1:V$ enthaltende Parameter Data Area als eine (in einem `DEFINE DATA LOCAL`-Statement angegebene) Local Data Area benutzt wird, muss eine Variable mit Namen V als `CONSTANT` definiert worden sein.
2. In einem Dialog kann ein Index $1:V$ nicht in Zusammenhang mit `BY VALUE` benutzt werden.

42

Definition eines Ausgangswerts

- Funktion 226
- Einschränkung 226
- Syntax-Beschreibung 227

Die Definition eines Ausgangswertes (*init-definition*) ist in der Option *variable-definition* der folgenden Statements möglich: `DEFINE DATA LOCAL`, `DEFINE DATA INDEPENDENT`, `DEFINE DATA CONTEXT` und `DEFINE DATA OBJECT`.

Die *init-definition* hat die folgende Syntax:

```
{ <constant>
  <system-variable>
  FULL LENGTH <character-s>
  LENGTH n <character-s> }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Mit der Option *init-definition* definieren Sie die Ausgangswerte/Konstanten-Werte für eine Variable.



Anmerkung: Wenn in der Option *variable-definition* das Schlüsselwort `INIT` für die Initialisierung benutzt wurde, kann der Wert von einem Statement geändert werden, das den Inhalt einer Variable beeinflusst. Wenn das Schlüsselwort `CONST` für die Initialisierung benutzt wurde, wird jeder Versuch, den Wert zu ändern, vom Compiler zurückgewiesen.

Siehe auch *Felder definieren*, *Ausgangswerte* im *Leitfaden zur Programmierung*.

Einschränkung

Für ein redefiniertes Feld ist eine *init-definition* nicht zulässig.

Syntax-Beschreibung

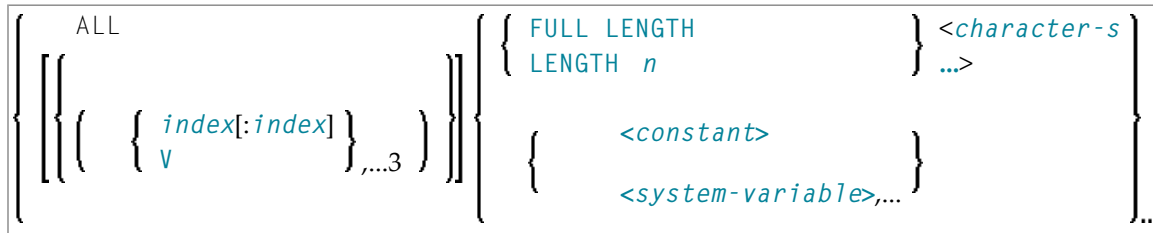
<code><constant></code>	Der Konstanten-Wert, mit der die Variable initialisiert werden soll, bzw. der Konstanten-Wert, der dem Feld fest zugewiesen wird. Informationen zu Konstanten siehe Abschnitt <i>Benutzerkonstanten</i> im <i>Leitfaden zur Programmierung</i> .
<code><system-variable></code>	<p>Als Ausgangswert einer Variablen können Sie auch den Wert einer Natural-Systemvariablen benutzen, zum Beispiel:</p> <pre data-bbox="513 575 1474 709">DEFINE DATA LOCAL 1 #MYDATE (D) INIT <*DATX> END-DEFINE</pre> <p>Anmerkung: Wenn die Variable in einem <code>RESET INITIAL</code>-Statement referenziert wird, wird die Systemvariable neu ausgewertet; d.h. die Variable wird nicht auf den Wert zurückgesetzt, den die Systemvariable zu Beginn der Programmausführung hatte, sondern auf den Wert, den sie zum Zeitpunkt der Ausführung des Statements <code>RESET INITIAL</code> hat.</p>
<p>FULL LENGTH <code><character-s></code></p> <p>LENGTH <i>n</i> <code><character-s></code></p>	<p>Als Ausgangswert können Sie eine Variable auch vollständig oder teilweise mit einem bestimmten Zeichen oder einer Zeichenkette füllen (nur bei alphanumerischen Codepage- oder Unicode-Variablen möglich).</p> <p>Mit der Option <code>FULL LENGTH</code> füllen Sie das gesamte Feld mit dem/den angegebenen Zeichen (<i>character(s)</i>). Im folgenden Beispiel wird das gesamte Feld mit Sternen (*) gefüllt.</p> <pre data-bbox="513 1213 1474 1348">DEFINE DATA LOCAL 1 #FIELD (A25) INIT FULL LENGTH <'*> END-DEFINE</pre> <p>Mit der Option <code>LENGTH <i>n</i></code> füllen Sie die ersten <i>n</i> Stellen des Feldes mit dem bzw. den angegebenen Zeichen (<i>character(s)</i>). <i>n</i> muss eine numerische Konstante sein. Im folgenden Beispiel werden die ersten 4 Stellen des Feldes mit Ausrufungszeichen (!) gefüllt.</p> <pre data-bbox="513 1549 1474 1684">DEFINE DATA LOCAL 1 #FIELD (A25) INIT LENGTH 4 <'!> END-DEFINE</pre>

43 Ausgangswerte/Konstanten-Werte für ein Array

- Funktion 230
- Einschränkung 230
- Syntax-Beschreibung 231

Die *array-init-definition* kann in der Option *variable-definition* der Statements `DEFINE DATA LOCAL`, `DEFINE DATA INDEPENDENT`, `DEFINE DATA CONTEXT` und `DEFINE DATA OBJECT` verwendet werden.

Die *array-init-definition* hat die folgende Syntax:



Dieses Kapitel behandelt folgende Themen:

Funktion

Mit der Option *array-init-definition* definieren Sie die Ausgangswerte oder die Konstanten-Werte für ein Array.



Anmerkung: Wenn in der Option *variable-definition* das Schlüsselwort `INIT` für die Initialisierung benutzt wurde, kann der Wert von einem Statement geändert werden, das den Inhalt einer Variable beeinflusst. Wenn das Schlüsselwort `CONST` für die Initialisierung benutzt wurde, wird jeder Versuch, den Wert zu ändern, vom Compiler zurückgewiesen.

Siehe auch *Felder definieren* im *Leitfaden zur Programmierung*, und zwar die folgenden Abschnitte:

- *Ausgangswerte*
- *Benutzerkonstanten*

Einschränkung

Für ein redefiniertes Feld ist eine *array-init-definition* nicht zulässig.

Syntax-Beschreibung

ALL	Alle Ausprägungen in allen Dimensionen des Arrays werden mit dem gleichen Wert initialisiert.
<i>index</i>	Nur die im <i>index</i> angegebenen Ausprägungen des Arrays werden initialisiert. Wenn Sie einen <i>index</i> angeben, dürfen Sie mit <i>constant</i> nur einen einzigen Wert angeben; d.h. die angegebenen Ausprägungen werden mit dem gleichen Wert initialisiert.
V	<p>Diese Notation ist nur relevant bei mehrdimensionalen Arrays, wenn den Ausprägungen einer Dimension unterschiedliche Werte zugewiesen werden.</p> <p>V bezeichnet einen Indexbereich, der sich über alle Ausprägungen der mit V bezeichneten Dimension erstreckt, d.h. alle Ausprägungen in dieser Dimension werden initialisiert.</p> <p>Pro Array darf höchstens eine Dimension mit V bezeichnet werden.</p> <p>Die Ausprägungen werden Ausprägung für Ausprägung mit den für diese Dimension angegebenen Werten initialisiert.</p> <p>Die Anzahl der Werte darf nicht größer sein als die Anzahl der Ausprägungen der mit V bezeichneten Dimension.</p>
<i>constant</i>	<p>Der Konstanten-Wert, der dem Array entweder als Ausgangswert (INIT) oder als Konstanten-Wert (CONSTANT) zugewiesen wird. Weitere Informationen zur Definition von Konstanten finden Sie im Abschnitt <i>Benutzerkonstanten</i> im <i>Leitfaden zur Programmierung</i>.</p> <p>Anmerkung: Ausprägungen, für die Sie keine Werte angeben, werden mit einem Standardwert initialisiert.</p>
<i>system-variable</i>	<p>Als Ausgangswert können Sie einem Array auch den Wert einer Natural-Systemvariablen zuweisen.</p> <p>Anmerkung: Wenn Sie mehrere Konstante oder Natural-Systemvariable angeben, müssen Sie diese entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander trennen. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (mit dem Session-Parameter DC) definiert ist.</p>
FULL LENGTH LENGTH <i>n</i>	<p>Als Ausgangswert können Sie ein Array auch vollständig oder teilweise mit einem bestimmten Zeichen oder einer Zeichenkette füllen (nur bei alphanumerischen oder Unicode-Variablen möglich):</p> <p>Mit FULL LENGTH füllen Sie die Array-Ausprägungen vollständig mit dem/den angegebenen Zeichen (<i>character(s)</i>).</p> <p>Mit LENGTH <i>n</i> füllen Sie die ersten <i>n</i> Stellen der Array-Ausprägungen mit dem/den angegebenen Zeichen (<i>character(s)</i>).</p>

	Eine Systemvariable (<i>system-variable</i>) darf bei FULL LENGTH oder LENGTH <i>n</i> nicht angegeben werden. Innerhalb einer <i>array-init-definition</i> können Sie nur entweder FULL LENGTH oder LENGTH <i>n</i> angeben, aber nicht beides.
--	--

Beispiel von LENGTH *n* für Array:

```
DEFINE DATA LOCAL
1 #FIELD (A25/1:3) INIT ALL LENGTH 5 <'NO'>
...
END-DEFINE
```

In diesem Beispiel werden die ersten 5 Positionen jeder Ausprägung des Arrays mit NONON gefüllt.

Weitere Beispiele für die Zuweisung von Ausgangswerten zu Arrays finden Sie im *Leitfaden zur Programmierung*.

44

Parameter EM, HD, PM für Feld/Variable

- Funktion 234
- Syntax-Beschreibung 234

Die Option *emhdpm* wird in der Option *view-definition* der folgenden Statements verwendet: **DEFINE DATA LOCAL** und **DEFINE DATA OBJECT**.

Außerdem wird sie in der Option *variable-definition* der folgenden Statements verwendet: **DEFINE DATA LOCAL**, **DEFINE DATA INDEPENDENT**, **DEFINE DATA CONTEXT** and **DEFINE DATA OBJECT**.

Die Option *emhdpm* hat die folgende Syntax:

```
( [ EM=value ] [ EMU=value ] [HD='text'] [PM=value])
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt **Syntax-Symbole**.

Funktion

Mit dieser Option können Sie für ein Feld bzw. eine Variable zusätzliche Parameter definieren.



Anmerkung: Wenn Sie für ein Datenbankfeld weder eine Editiermaske (EM= oder EMU=) noch eine Spaltenüberschrift (HD=) angeben, werden die Standard-Editiermaske und die Standard-Spaltenüberschrift aus dem **DDM** genommen. Wenn sie jedoch eins von beiden angeben, wird für das jeweils andere *nicht* die Standarddefinition aus dem DDM genommen.

Syntax-Beschreibung

EM=value	Mit diesem Parameter können Sie eine Editiermaske definieren, die benutzt wird, wenn das Feld mit einem I/O-Statement angezeigt wird. Siehe Session-Parameter EM in der <i>Parameter-Referenz</i> .
EMU=value	Mit diesem Parameter können Sie eine Unicode-Editiermaske definieren, die benutzt wird, wenn das Feld mit einem I/O-Statement angezeigt wird. Siehe Session-Parameter EM in der <i>Parameter-Referenz</i> .
HD='text'	Mit diesem Parameter können Sie eine Überschrift definieren, die als Standard-Spaltenüberschrift für das Feld ausgegeben wird. Siehe Session-Parameter HD in der <i>Parameter-Referenz</i> .
PM=value	Mit diesem Parameter können Sie den Print-Modus setzen, der definiert, wie Felder ausgegeben werden. Siehe Session-Parameter PM in der <i>Parameter-Referenz</i> .

45

Beispiele für die Benutzung des DEFINE DATA-Statements

▪ Beispiel 1 — DEFINE DATA LOCAL (Direkte Daten-Definition)	236
▪ Beispiel 2 — DEFINE DATA LOCAL (Array-Definition/Initialisierung)	236
▪ Beispiel 3 — DEFINE DATA (View-Definition, Array-Redefinition)	237
▪ Beispiel 4 — DEFINE DATA (Global, Parameter und Local Data Areas)	239
▪ Beispiel 5 — DEFINE DATA (Initialisierung)	240
▪ Beispiel 6 — DEFINE DATA (Variables Array mit (1:V))	240

Die folgenden Themen werden behandelt:

Beispiel 1 — DEFINE DATA LOCAL (Direkte Daten-Definition)

```

** Example 'DDAEX1': DEFINE DATA
*****
DEFINE DATA LOCAL
1 #VAR1      (A15)
1 #VAR2
  2 #VAR2A   (N4.1) INIT <1111>
  2 #VAR2B   (N6.2) INIT <222222>
1 REDEFINE #VAR2
  2 #VAR2C   (A2)
  2 #VAR2D   (A2)
  2 #VAR2E   (A6)
END-DEFINE
*
WRITE NOTITLE '=' #VAR2A / '=' #VAR2B /
              '=' #VAR2C / '=' #VAR2D / '=' #VAR2E
*
END
    
```

Ausgabe des Programms DDAEX1:

```

#VAR2A:  1111.0
#VAR2B:  222222.00
#VAR2C:  11
#VAR2D:  11
#VAR2E:  022222
    
```

Beispiel 2 — DEFINE DATA LOCAL (Array-Definition/Initialisierung)

```

** Example 'DDAEX2': DEFINE DATA (array definition/initialization)
*****
DEFINE DATA LOCAL
1 #VAR1 (A1/1:2,1:2) INIT (1,V)  <'A','B'>
1 #VAR2 (N5/1:2,1:3) INIT (1,2)  <200>
1 #VAR3 (A1/1:4,1:3) INIT (V,2:3) <'W','X','Y','Z'>
END-DEFINE
*
WRITE NOTITLE '=' #VAR1 (1,1) '=' #VAR1 (1,2)
              / '=' #VAR1 (2,1) '=' #VAR1 (2,2)
    
```



```

*
WRITE      /// '=' #VAR2 (1,1) '=' #VAR2 (1,2)
           /  '=' #VAR2 (2,1) '=' #VAR2 (2,2)
*
WRITE      /// '=' #VAR3 (1,1) '=' #VAR3 (1,2) '=' #VAR3 (1,3)
WRITE      /  '=' #VAR3 (2,1) '=' #VAR3 (2,2) '=' #VAR3 (2,3)
WRITE      /  '=' #VAR3 (3,1) '=' #VAR3 (3,2) '=' #VAR3 (3,3)
WRITE      /  '=' #VAR3 (4,1) '=' #VAR3 (4,2) '=' #VAR3 (4,3)
*
END

```

Ausgabe des Programms DDAEX2:

```

#VAR1: A #VAR1: B
#VAR1:  #VAR1:

#VAR2:      0 #VAR2:      200
#VAR2:      0 #VAR2:      0

#VAR3:  #VAR3: W #VAR3: W
#VAR3:  #VAR3: X #VAR3: X
#VAR3:  #VAR3: Y #VAR3: Y
#VAR3:  #VAR3: Z #VAR3: Z

```

Beispiel 3 — DEFINE DATA (View-Definition, Array-Redefinition)

```

** Example 'DDAEX3': DEFINE DATA (view definition, array redefinition)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
*
1 #ARRAY      (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE    (A25/1:4,1:3)
1 #X          (N2) INIT <1>
1 #Y          (N2) INIT <1>

```

```

END-DEFINE
*
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  MOVE NAME                TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE                TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
  PERFORM PRINT
END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END

```

Ausgabe des Programms DDAEX3:

SMITH ENGLANDSVEJ 222 554349	SMITH 3152 SHETLAND ROAD MILWAUKEE 877-4563	SMITH 14100 ESWORTHY RD. MONTERREY 994-2260
SMITH 5 HAWTHORN OAK BROOK 150-9351	SMITH 13002 NEW ARDEN COUR SILVER SPRING 639-8963	

Beispiel 4 — DEFINE DATA (Global, Parameter und Local Data Areas)

```

** Example 'DDAEX4': DEFINE DATA (global and local data area definition)
*****
DEFINE DATA
GLOBAL
  USING DDAEX4G
LOCAL
  1 #FIELD1 (A10)
  1 #FIELD2 (N5)
END-DEFINE
*
MOVE 'HELLO' TO #FIELD1
MOVE 123     TO #FIELD2
*
CALLNAT 'DDAEX4N' #FIELD1 #FIELD2
*
END

```

Vom Programm DDAEX4 benutzte Global Data Area DDAEX4G:

```

1 GLOBAL-FIELD          A    10

```

Vom Programm DDAEX4 aufgerufenes Subprogramm DDAEX4N:

```

** Example 'DDAEX4N': DEFINE DATA PARAMETER (called by DDAEX4)
*****
DEFINE DATA
PARAMETER
  1 #FIELDA (A10)
  1 #FIELDB (N5)
END-DEFINE
*
WRITE '=' #FIELDA '=' #FIELDB
END

```

Ausgabe des Programms DDAEX4:

```
#FIELD A: HELLO      #FIELD B: 123
```

Beispiel 5 — DEFINE DATA (Initialisierung)

```
** Example 'DDAEX5': DEFINE DATA (initialization)
*****
DEFINE DATA LOCAL
1 #START-DATE (D)   INIT <*DATX>
1 #UNDERLINE   (A50) INIT FULL LENGTH <'_'>
1 #SCALE       (A65) INIT LENGTH 65 <'....+..../'>
END-DEFINE
*
WRITE NOTITLE #START-DATE (DF=L)
              / #UNDERLINE
              / #SCALE
END
```

Ausgabe des Programms DDAEX5:

```
2005-01-12
```

```
....+..../....+..../....+..../....+..../....+..../....+..../....+
```

Beispiel 6 — DEFINE DATA (Variables Array mit (1:V))

```
** Example 'DDAEX6': DEFINE DATA (variable array with (1:V))
*****
DEFINE DATA LOCAL
1 #ARRAY   (A1/1:10)
1 #MAX-ARR (P3)
END-DEFINE
*
#ARRAY (1) := 'R'
#ARRAY (2) := 'E'
#ARRAY (3) := 'D'
#MAX-ARR := 4
*
WRITE #ARRAY(*)
*
```

```

CALLNAT 'DDAEX6N' #ARRAY(1:4) #MAX-ARR
*
WRITE #ARRAY(*)
*
*
#MAX-ARR := 5
*
CALLNAT 'DDAEX6N' #ARRAY(1:5) #MAX-ARR
*
WRITE #ARRAY(*)
*
END

```

Vom Programm DDAEX6 aufgerufenes Subprogramm DDAEX6N:

```

** Example 'DDAEX6N': DEFINE DATA (variable array with (1:V))
*****
DEFINE DATA
PARAMETER
1 #STRING (A1/1:V)
1 #MAX (P3)
END-DEFINE
*
IF #MAX = 4
  MOVE 'B' TO #STRING (1)
  MOVE 'L' TO #STRING (2)
  MOVE 'U' TO #STRING (3)
  MOVE 'E' TO #STRING (4)
END-IF
*
IF #MAX = 5
  MOVE 'W' TO #STRING (1)
  MOVE 'H' TO #STRING (2)
  MOVE 'I' TO #STRING (3)
  MOVE 'T' TO #STRING (4)
  MOVE 'E' TO #STRING (5)
END-IF
END

```

Ausgabe des Programms DDAEX4:

```

Page      1                                     05-01-12  09:06:43

R E D
B L U E
W H I T E

```


46

DEFINE FUNCTION

- Funktion 244
- Syntax-Beschreibung 244
- Beispiel 246

```

DEFINE FUNCTION function-name
  [return-data-definition]
  [function-data-definition]
  statement...
END-FUNCTION
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Mit dem `DEFINE FUNCTION`-Statement können Sie benutzerdefinierte Funktionen erstellen, die in den Natural-Statements anstelle von Operanden aufgerufen werden können. Diese Funktionen können nur innerhalb eines Natural-Objekts vom Typ Function definiert werden.

Weitere Informationen finden Sie in den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- Natural-Objecttyp Function
- *Function Call*
- *Benutzerdefinierte Funktionen*

Syntax-Beschreibung

<i>function-name</i>	<p><i>function-name</i> ist der symbolische Name der zu definierenden Natural-Funktion. Es gelten die im Kapitel <i>Namenskonventionen für Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i> aufgeführten Regeln. Das bedeutet, dass der Name maximal 32 Zeichen lang sein und mit einem Buchstaben oder einem Sonderzeichen, z.B. Rautensymbol (#), beginnen darf.</p> <p>Sie dürfen denselben Function-Namen nicht zweimal in einer Library benutzen (einschließlich der Libraries mit dem Steplib-Mechanismus). Funktionsüberladung ist nicht erlaubt. Dies bedeutet, dass alle Funktionsdefinitionen eindeutige Function-Namen haben müssen.</p>
<i>return-data-definition</i>	Siehe Rückgabedatendefinition weiter unten.
<i>function-data-definition</i>	Siehe Funktionsdatendefinition weiter unten.

END-FUNCTION	Das für Natural reservierte Wort END-FUNCTION muss zum Beenden des DEFINE FUNCTION-Statements benutzt werden.
---------------------	---

Rückgabedatendefinition

RETURNS [variable-name]	$\left\{ \begin{array}{l} (format-length[/array-definition]) \\ (\left\{ \begin{array}{l} A \\ U \\ B \end{array} \right\} [/array-definition]) \text{ DYNAMIC} \end{array} \right\}$	[BY VALUE]
----------------------------	--	---------------

Syntax-Element-Beschreibung:

RETURNS	Jede Function darf nur eine Definition der Rückgabevariablen enthalten; d.h. es ist nur <i>eine</i> RETURNS-Klausel zulässig.
<i>variable-name</i>	<p>Der Rückgabewert kann mit <i>variable-name</i> zugewiesen werden. Ist in der Definition kein expliziter Variablenname angegeben, wird der Name der Function als Rückgabevariable verwendet.</p> <p>Der Rückgabewert darf kein Array sein.</p>
BY VALUE	<p>Jeder Parameter kann direkt als Wert (By-Value) oder referenziert über seine Adresse (By-Reference) definiert werden, so dass es möglich ist, Werte über Parameter an den Caller zurückzugeben. Innerhalb einer Funktionsdefinition können rekursive Funktionsaufrufe verwendet werden.</p> <p>Wenn Sie das Schlüsselwort BY VALUE in der RETURNS-Klausel verwenden, wird der Rückgabewert der Function in das/die durch die RETURNS-Klausel festgelegte Format/Länge (<i>format-length</i>) umgesetzt.</p>
<i>format-length</i>	Wenn Sie das Schlüsselwort BY VALUE weglassen, muss die Angabe für <i>format-length</i> bei der RETURNS-Klausel mit der von der zur Laufzeit ausgewerteten Function zurückgegebenen Format/Länge-Angabe übereinstimmen.
<i>array-definition</i>	Mit <i>array-definition</i> legen Sie die untere und obere Grenze einer Dimension bei einer Array-Definition fest. Weitere Informationen siehe DEFINE DATA-Statement, Definition von Array-Dimensionen .
DYNAMIC	Ein Parameter kann als DYNAMIC definiert werden. Informationen zur Verarbeitung von dynamischen Variablen siehe <i>Dynamische Variablen</i> .

Funktionsdatendefinition

Jedes Objekt des Typs Function darf nur eine Funktionsdatendefinition enthalten.

```

DEFINE DATA
[
  PARAMETER { USING parameter-data-area } ] ...
[
  LOCAL { USING { local-data-area } } ] ...
           { parameter-data-area } } ] ...
           { data-definition... } } ] ...
[INDEPENDENT AIV-data-definition...]
END-DEFINE
  
```

Wenn eine Function ein anderes Natural-Objekt aufruft, das eine Global Data (GDA) Area benutzt, dann erstellt sie ihre eigene GDA. Darum ist es nicht möglich, die aktuellen GDA-Daten des aufrufenden Objekts zu verändern. In der Function darf keine GDA angegeben werden.

Beispiel

Objekt vom Typ Function mit Funktionsdefinition:

```

DEFINE FUNCTION GET-FIRST-BYTE
  RETURNS (A1)
  DEFINE DATA PARAMETER
    1 #PARA (A10)
  END-DEFINE
  GET-FIRST-BYTE := #PARA /* return value is assigned
END-FUNCTION
END
  
```

47 DEFINE PRINTER

▪ Funktion	248
▪ Syntax-Beschreibung	248
▪ Beispiele	250

```
DEFINE PRINTER ([logical-printer-name=]n)
  [OUTPUT operand1]
  [PROFILE operand2]
  [DISP operand2]
  [COPIES operand3]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TITLE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `DEFINE PRINTER` dient dazu, einer Report-Nummer einen symbolischen Namen zuzuordnen und die Zuweisung eines Reports zu einem logischen Bestimmungsort (Drucker) zu steuern. Dies bietet zusätzliche Flexibilität bei der Erstellung von Ausgaben für verschiedene logische Drucker-Warteschlangen.

Ist bei der Ausführung dieses Statements der angegebene Drucker bereits offen, bewirkt dieses Statement implizit, dass der Drucker geschlossen wird. Um einen Drucker explizit zu schließen, sollten Sie das Statement `CLOSE PRINTER` verwenden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C	S	A	ja	nein
<i>operand2</i>	C	S	A	ja	nein

Syntax-Element-Beschreibung:

<i>(n)</i>	<p>Druckernummer (Report-Nummer):</p> <p>Die Report-Nummer <i>n</i> kann ein Wert im Bereich von 0 - 31 sein. Dies ist die Nummer, die auch in einem DISPLAY-, WRITE- oder CLOSE PRINTER-Statement verwendet werden soll.</p> <p>Report-Nummer 0 steht für den Ausgabekanal des Haupt-Reports. Nur Ausgabe-Statements wie PRINT, WRITE oder DISPLAY sind betroffen. Das INPUT-Statement ist nicht betroffen.</p>								
<i>logical-printer-name</i>	<p>Logischer Druckername:</p> <p>Als Option können Sie dem Drucker <i>n</i> einen logischen Druckernamen <i>logical-printer-name</i> zuweisen. Dieser Name kann für die <i>rep</i>-Notation in einem DISPLAY- oder WRITE-Statement benutzt werden.</p> <p>Die Namenskonventionen für <i>logical-printer-name</i> sind identisch mit denen für Benutzervariablen. Mehrere logische Namen können ein- und derselben Druckernummer zugewiesen werden. Im Gegensatz zum Wert des OUTPUT-Operanden (siehe unten), wird <i>logical-printer-name</i> zur Kompilierzeit ausgewertet und ist deshalb unabhängig vom Programmkontrollfluss.</p>								
OUTPUT <i>operand1</i>	<p>Druckername:</p> <p>Wenn <i>operand1</i> eine Variable ist, muss dessen Format/Länge entweder A8 oder eines der Folgenden sein. Der Name muss als LPT<i>nn</i> angegeben werden, wobei <i>nn</i> zwischen 1 und 31 liegen kann. Siehe auch Beispiel 1.</p> <p>Anmerkung: Wenn die in einen Report geschriebenen Ausgabedaten an ein Entire Connection-Terminal gesendet und dann in eine NCD-Datei auf dem PC geschrieben werden sollen, muss einer der LPT<i>nn</i>-Druckernamen (wobei <i>nn</i> eine Zahl zwischen 1 und 31 ist) als <i>operand1</i> angegeben werden. Die Gerätezuordnung des logischen Druckers LPT<i>nn</i> muss im Configuration Utility gemacht werden; siehe <i>Device/Report Assignments</i>. Als Device Destination des Physical Output Device muss der Wert "E" (Daten an Entire Connection-Terminal senden) angegeben werden.</p> <p>Zusätzliche Reports können mit den folgenden Namen zugewiesen werden:</p> <table border="1" data-bbox="548 1528 1487 1852"> <thead> <tr> <th>Report</th> <th>Funktion</th> </tr> </thead> <tbody> <tr> <td>DUMMY</td> <td>Ausgabe wird gelöscht.</td> </tr> <tr> <td>INFOLINE</td> <td>Ausgabe in der Natural-Infoline. Näheres zur Infoline siehe Terminalkommando %X in der <i>Terminalkommandos</i>-Dokumentation. Siehe auch Beispiel 2.</td> </tr> <tr> <td>SOURCE</td> <td>Ausgabe in den Arbeitsbereich des Natural-Editors.</td> </tr> </tbody> </table>	Report	Funktion	DUMMY	Ausgabe wird gelöscht.	INFOLINE	Ausgabe in der Natural-Infoline. Näheres zur Infoline siehe Terminalkommando %X in der <i>Terminalkommandos</i> -Dokumentation. Siehe auch Beispiel 2 .	SOURCE	Ausgabe in den Arbeitsbereich des Natural-Editors.
Report	Funktion								
DUMMY	Ausgabe wird gelöscht.								
INFOLINE	Ausgabe in der Natural-Infoline. Näheres zur Infoline siehe Terminalkommando %X in der <i>Terminalkommandos</i> -Dokumentation. Siehe auch Beispiel 2 .								
SOURCE	Ausgabe in den Arbeitsbereich des Natural-Editors.								
PROFILE <i>operand2</i>	<p>Name der Druckersteuerzeigentabelle:</p>								

	<p>Mit der PROFILE-Klausel geben Sie als <i>operand2</i> den Namen der Druckersteuerzeichentabelle an. Die maximal erlaubte Länge für <i>operand2</i> ist 8.</p> <p>Eine solche Tabelle ist im Global Configuration File definiert. Informationen zum Definieren von Druckerprofilen finden Sie unter <i>Printer Profiles</i> (in der <i>Configuration Utility</i>-Dokumentation).</p>						
DISP <i>operand2</i>	<p>Disposition:</p> <p>Maximale Länge des Operanden: 4 Bytes.</p> <p>Mögliche Werte für <i>operand2</i>:</p> <table border="1"> <tr> <td>DEL</td> <td>Das temporäre Spool-File wird gelöscht, nachdem dessen Inhalt gedruckt worden ist. Dies ist der Default-Wert.</td> </tr> <tr> <td>KEEP</td> <td>Das temporäre Spool-File wird <i>nicht</i> gelöscht, nachdem dessen Inhalt gedruckt worden ist.</td> </tr> <tr> <td>HOLD</td> <td>Das temporäre Spool-File wird weder gelöscht noch gedruckt.</td> </tr> </table>	DEL	Das temporäre Spool-File wird gelöscht, nachdem dessen Inhalt gedruckt worden ist. Dies ist der Default-Wert.	KEEP	Das temporäre Spool-File wird <i>nicht</i> gelöscht, nachdem dessen Inhalt gedruckt worden ist.	HOLD	Das temporäre Spool-File wird weder gelöscht noch gedruckt.
DEL	Das temporäre Spool-File wird gelöscht, nachdem dessen Inhalt gedruckt worden ist. Dies ist der Default-Wert.						
KEEP	Das temporäre Spool-File wird <i>nicht</i> gelöscht, nachdem dessen Inhalt gedruckt worden ist.						
HOLD	Das temporäre Spool-File wird weder gelöscht noch gedruckt.						
COPIES <i>operand3</i>	<p>Anzahl der Kopien:</p> <p><i>operand3</i> muss ein Ganzzahlwert sein.</p>						

Beispiele

- [Beispiel 1 - Definition des Druckernamens](#)
- [Beispiel 2 - Druckausgabe an Infoline](#)

Beispiel 1 - Definition des Druckernamens

```

/* PRINTER NAME DEFINED FOR WINDOWS
*
DEFINE PRINTER (REPORT1 = 1) OUTPUT 'LPT1'
WRITE (REPORT1) 'REPORT 1 PRINTED ON PRINTER LPT1'
END

```

Beispiel 2 - Druckausgabe an Infoline

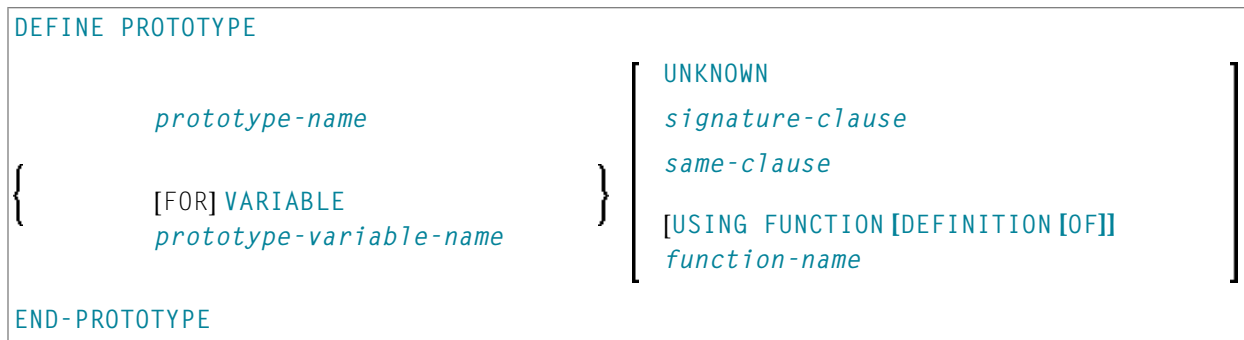
```
** Example 'DPIEX1': DEFINE PRINTER
*****
*
SET CONTROL 'XI+'          /* SWITCH INFOLINE MODE ON
SET CONTROL 'XT'          /* INFOLINE TOP
*
DEFINE PRINTER (1) OUTPUT 'INFOLINE'
WRITE (1) 'EXECUTING' *PROGRAM 'BY' *INIT-USER
WRITE 'TEST OUTPUT'
EJECT                     /* FORCE PHYSICAL I/O
*
SET CONTROL 'X'           /* SWITCH BACK TO NORMAL
*
END
```

Ausgabe des Programms DPIEX1:

```
EXECUTING DPIEX1   BY HTR
Page      1
05-01-13  14:54:33
TEST OUTPUT
```


48 DEFINE PROTOTYPE

▪ Funktion	254
▪ Syntax-Beschreibung	255
▪ Beispiel	257



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandtes Statement: `DEFINE FUNCTION`

Funktion

Die Prototypdefinition kann dazu benutzt werden, eine Signatur gemäß einem bestimmten Function Call anzugeben. Für jeden Function Call müssen der Rückgabebetyp und die Art des Function Calls (`VARIABLE`) bekannt sein. Deshalb müssen diese Daten bei jedem Function Call zur Verfügung stehen. Wenn diese Daten fehlen, muss das Prototypschlüsselwort in der Function Call-Referenz benutzt werden. Wenn es eine Parameterdefinition im Prototyp gibt, werden die Parameterwerte des Function Call mit den Parametern der Prototypdefinition verglichen. Wenn die Parameter nicht geprüft werden sollen, benutzen Sie das Schlüsselwort `UNKNOWN` im `DEFINE DATA PARAMETER`-Statement der Prototypdefinition.

Weitere Informationen finden Sie in den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- Natural-Objecttyp Function
- *Function Call*
- *Benutzerdefinierte Funktionen*

Syntax-Beschreibung

<i>prototype-name</i>	Für den <i>prototype-name</i> gelten dieselben Regeln wie für Benutzervariablen - mit einer Ausnahme: Prototypnamen dürfen Punkte (.) enthalten. Der <i>prototype-name</i> ist frei wählbar. Es ist nicht erforderlich, dass er denselben Namen hat wie die entsprechende Funktionsdefinition. Die maximale Länge des kompletten <i>prototype-name</i> beträgt 32 Zeichen.
VARIABLE <i>prototype-variable-name</i>	Mit dem <i>prototype-variable-name</i> können Sie Functions mit variablen Funktionsnamen aufrufen. Das ist ähnlich wie bei den CALLNAT -Funktionsaufrufen. Der <i>prototype-variable-name</i> ist der Name einer alphanumerischen Variable, die den richtigen Namen der Function enthält, die in der Funktionsreferenz aufgerufen werden soll.
UNKNOWN	Wenn die Parameter nicht geprüft werden sollen, benutzen Sie das Schlüsselwort UNKNOWN im DEFINE DATA PARAMETER -Statement der Prototypdefinition.
<i>signature-clause</i>	Siehe <i>Signature-Klausel</i> unten.
<i>prototype-return-data-definition</i>	Siehe <i>Prototyp-Rückgabewert-Definition</i> unten.
<i>same-clause</i>	Siehe <i>SAME AS-Klausel</i> unten.
USING FUNCTION [DEFINITION [OF]] <i>function-name</i>	Siehe <i>USING FUNCTION-Klausel</i> unten.
END-PROTOTYPE	Das für Natural reservierte Schlüsselwort END-PROTOTYPE muss zum Beenden des DEFINE PROTOTYPE -Statements benutzt werden.

Signature-Klausel

```

[prototype-return-data-definition]
DEFINE DATA
{
  PARAMETER UNKNOWN
  {
    PARAMETER
    {
      USING parameter-data-area
      parameter-data-definition ...
    }
  }
}
END-DEFINE

```

Diese Klausel sieht aus wie ein bestimmter Function Call. Normalerweise stimmt der Prototyp mit der Funktionsdefinition überein. Er muss jedoch nicht exakt derselbe sein. So ist es möglich,

die Parameterdaten wegzulassen und statt dessen das Schlüsselwort `UNKNOWN` zu benutzen. In diesem Fall werden die Parameter zum Kompilierzeitpunkt nicht geprüft.

Der Typ des Rückgabewerts muss in jedem Fall gesetzt werden. Wenn kein Rückgabewert definiert ist, dann ist die Zuweisung vom Function Call an eine Variable nicht erlaubt.

Wenn in einer Prototypdefinition keine Signatur angegeben ist (Signatur ist `UNKNOWN`), muss die entsprechende Signatur eines Function Call mit dem Schlüsselwort `PT` angegeben werden. Weitere Informationen zu `PT` finden Sie unter *Function Call*, Abschnitt *prototype-cast*, im Leitfaden zur Programmierung.

Prototyp-Rückgabewert-Definition

```

RETURNS
[variable-name] { (format-length [/array-definition])
                  ( { A
                    { U } [/array-definition]) DYNAMIC
                    { B }
                  }
                }
    
```

Diese Klausel definiert Format und Länge (*format-length*) des Rückgabewerts, der zum Kompilierzeitpunkt bekannt sein muss.

Der optionale Variablenname (*variable-name*) wird ignoriert. Er wurde eingeführt, damit die Syntaxstruktur ähnlich ist wie bei der `RETURNS`-Klausel des `DEFINE FUNCTION`-Statements.

SAME AS -Klausel

```

SAME AS [PROTOTYPE] { prototype-name
                    { prototype-variable-name }
                    }
    
```

Diese Klausel kann dafür benutzt werden, Signaturen zu benutzen, die vorher definiert wurden, um einen neuen Prototyp zu definieren.

USING FUNCTION-Klausel

```

[USING FUNCTION [DEFINITION [OF]] function-name]
    
```

Diese explizite Klausel bietet Ihnen die Möglichkeit, ein generiertes Objekt auf Funktionsparameterdefinitionen zu analysieren, die dann dazu verwendet werden, ein indirektes `DEFINE PROTOTYPE`-Statement unter dem logischen Namen dieser Funktion zu erstellen. *function-name* ist der logische Name; er ist nicht der Objektname des Function-Objekts. Der logische Function-Name ist im Funktionsrumpf (Body) des entsprechenden Function-Objekts definiert: `DEFINE FUNCTION function-name ... END-FUNCTION`.

Beispiel

- [Beispiel 1 - DEFINE PROTOTYPE](#)
- [Beispiel 2 - DEFINE PROTOTYPE](#)

Beispiel 1 - DEFINE PROTOTYPE

Dies ist eine Prototypdefinition einer Function mit dem Namen `GET-FIRST-BYTE`. Mit dem folgenden Prototyp kann die Function `GET-FIRST-BYTE` als symbolischer Function Call aufgerufen werden.

```
GET-FIRST-BYTE(<#A>)
```

```
DEFINE DATA LOCAL
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE
DEFINE PROTOTYPE GET-FIRST-BYTE
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE
WRITE GET-FIRST-BYTE(<#A>)
END
```

Beispiel 2 - DEFINE PROTOTYPE

Der folgende Natural-Code enthält die Prototypdefinition einer Function, in diesem Fall `GET-FIRST-BYTE`. Damit die Function dynamisch aufgerufen werden kann, muss der Name der Function in der alphanumerischen Variablen `#A` gespeichert sein. Bevor die Variable `#A` benutzt werden kann, muss sie als alphanumerische Variable im `DEFINE DATA`-Statement definiert werden.

```
DEFINE DATA LOCAL
1 FUNCTION-NAME(A32) INIT<'GET-FIRST-BYTE'>
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE
DEFINE PROTOTYPE VARIABLE FUNCTION-NAME
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE
WRITE FUNCTION-NAME(<#A>)
END
```


49

DEFINE SUBROUTINE

- Funktion 260
- Einschränkungen 260
- Syntax-Beschreibung 262
- Welche Daten einer Subroutine zur Verfügung stehen 262
- Beispiele 263

```

DEFINE [SUBROUTINE] subroutine-name
    statement ...
{   END-SUBROUTINE   }
    RETURN (nur im Reporting Mode) }
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `CALL` | `CALL FILE` | `CALL LOOP` | `CALLNAT` | `ESCAPE` | `FETCH` | `PERFORM`

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Das Statement `DEFINE SUBROUTINE` dient dazu, eine Natural-Subroutine zu definieren. Aufgerufen wird eine Subroutine mit einem `PERFORM`-Statement.

Interne und externe Subroutinen

Eine Subroutine kann entweder innerhalb des Natural-Objekts definiert werden, das das sie aufrufende `PERFORM`-Statement enthält (interne Subroutine); oder sie kann in einem anderen Natural-Objekt definiert werden als dem, welches das aufrufende `PERFORM`-Statement enthält (externe Subroutine). Eine interne Subroutine kann entweder vor oder nach dem ersten `PERFORM`-Statement, mit dem sie aufgerufen wird, definiert werden.



Anmerkung: Die Verwendung externer Subroutinen empfiehlt sich zwar, um eine klar gegliederte Anwendungsstruktur zu erhalten; allerdings verursachen externe Subroutinen einen Verarbeitungsmehraufwand. Daher sollten nur größere funktionale Blöcke in externen Subroutinen untergebracht werden.

Einschränkungen

- Eine in einer Subroutine initiierte Verarbeitungsschleife muss vor dem `END-SUBROUTINE`-Statement wieder geschlossen werden.
- Eine interne Subroutine darf ihrerseits kein weiteres `DEFINE SUBROUTINE`-Statement enthalten (siehe *Beispiel 1* unten).

- Eine externe Subroutine (d.h. ein Objekt vom Typ Subroutine) darf nicht mehr als einen DEFINE SUBROUTINE-Statement-Block enthalten (siehe Beispiel 2 unten). Ein externer DEFINE SUBROUTINE-Block darf jedoch seinerseits interne Subroutinen enthalten (siehe *Beispiel 1* unten).

Beispiel 1

Die folgende Konstruktion ist in einem Objekt vom Typ Subroutine möglich, aber nicht in einem anderen Objekt (in dem SUBR01 als interne Subroutine gälte):

```

...
DEFINE SUBROUTINE SUBR01
  ...
  PERFORM SUBR02
  PERFORM SUBR03
  ...
  DEFINE SUBROUTINE SUBR02
  /* inline subroutine...
  END-SUBROUTINE
  ...
  DEFINE SUBROUTINE SUBR03
  /* inline subroutine...
  END-SUBROUTINE
END-SUBROUTINE
END

```

Beispiel 2 (ungültig):

Die folgende Konstruktion ist in einem Objekt vom Typ Subroutine *nicht* erlaubt:

```

...
DEFINE SUBROUTINE SUBR01
...
END-SUBROUTINE
DEFINE SUBROUTINE SUBR02
...
END-SUBROUTINE
END

```

Syntax-Beschreibung

<i>subroutine-name</i>	Für den Namen einer Subroutine (maximal 32 Zeichen lang) gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe Abschnitt <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i> . Der Name einer Subroutine ist unabhängig vom Namen des Moduls, in dem sie definiert ist (beide Namen können, müssen aber nicht, gleich sein).
END-SUBROUTINE	Die Definition einer Subroutine wird mit END-SUBROUTINE beendet.
RETURN	Im Reporting Mode darf eine Subroutine auch mit RETURN beendet werden.

Welche Daten einer Subroutine zur Verfügung stehen

Dieses Abschnitt behandelt folgende Themen:

- [Interne Subroutinen](#)
- [Externe Subroutinen](#)

Interne Subroutinen

An eine interne Subroutine können mit dem **PERFORM**-Statement keine Parameter vom aufrufenden Programm übergeben werden.

Eine interne Subroutine kann auf die aktuelle Global Data Area sowie die vom aufrufenden Programm verwendete Local Data Area zugreifen.

Externe Subroutinen

Eine externe Subroutine kann auf die aktuelle Global Data Area zugreifen. Außerdem können Sie mit dem **PERFORM**-Statement Parameter direkt vom aufrufenden Objekt an die externe Subroutine übergeben; dadurch können Sie die Größe der Global Data Area klein halten.

Eine externe Subroutine kann nicht auf die im aufrufenden Programm definierte Local Data Area zugreifen; allerdings kann eine externe Subroutine eine eigene Local Data Area haben.

Beispiele

- Beispiel 1 — Subroutine definieren
- Beispiel 2 — Beispiel-Struktur für externe Subroutine mittels GDA-Feldern

Beispiel 1 — Subroutine definieren

```

** Example 'DSREX1S': DEFINE SUBROUTINE (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
*
1 #ARRAY (A75/1:4)
1 REDEFINE #ARRAY
  2 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
*
FORMAT PS=20
LIMIT 5
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  MOVE NAME TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
    PERFORM PRINT
  END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END

```

Ausgabe des Programms DSREX1S:

```
SMITH          SMITH          SMITH
ENGLANDSVEJ 222 3152 SHETLAND ROAD 14100 ESWORTHY RD.
554349         MILWAUKEE    MONTERREY
              877-4563      994-2260

SMITH          SMITH
5 HAWTHORN    13002 NEW ARDEN COUR
OAK BROOK     SILVER SPRING
150-9351      639-8963
```

Äquivalentes Reporting-Mode-Beispiel: [DSREX1R](#).

Beispiel 2 — Beispiel-Struktur für externe Subroutine mittels GDA-Feldern

```
** Example 'DSREX2': DEFINE SUBROUTINE (using GDA fields)
*****
DEFINE DATA
GLOBAL
  USING DSREX2G
END-DEFINE
*
INPUT 'Enter value in GDA field' GDA-FIELD1
*
* Call external subroutine in DSREX2S
*
PERFORM DSREX2-SUB
*
END
```

Vom Programm DSREX2 benutzte Global Data Area DSREX2G:

```
1 GDA-FIELD1          A      2
```

Vom Programm DSREX2 aufgerufene Subroutine DSREX2S:

```
** Example 'DSREX2S': SUBROUTINE (external subroutine using global data)
*****
DEFINE DATA
GLOBAL
  USING DSREX2G
END-DEFINE
*
DEFINE SUBROUTINE DSREX2-SUB
```

```
*  
  WRITE 'IN SUBROUTINE' *PROGRAM '=' GDA-FIELD1  
*  
END-SUBROUTINE  
*  
END
```


50

DEFINE WINDOW

▪ Funktion	268
▪ Syntax-Beschreibung	269
▪ Schutz von Eingabefeldern in einem Fenster	273
▪ Aufrufen unterschiedlicher Fenster	273
▪ Beispiel	274

```

DEFINE WINDOW window-name
[
    SIZE {
        AUTO
        QUARTER
        operand1 * operand2
    }
]
[
    BASE {
        CURSOR
        {
            TOP
            BOTTOM
        }
        {
            LEFT
            RIGHT
        }
        operand3 / operand4
    }
]
[REVERSED [(CD=background-color)]]
[TITLE operand5]
[
    CONTROL {
        WINDOW
        SCREEN
    }
]
[
    FRAMED {
        [ON] [(CD=frame-color)] [position-clause]
        OFF
    }
]
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `INPUT` | `REINPUT` | `SET WINDOW`

Gehört zur Funktionsgruppe: *Bildschirmgenerierung für interaktive Verarbeitung*

Funktion

Das `DEFINE WINDOW`-Statement dient dazu, die Größe, Position und Attribute eines Bildschirmfensters (Window) zu definieren.

Ein Fenster ist der Ausschnitt einer von einem Programm erzeugten logischen Seite, der auf dem Bildschirm zu sehen ist. Das Fenster ist ständig vorhanden, auch wenn Sie sich dessen nicht bewusst sind, weil die Größe des Fensters, solange Sie sie nicht anders definieren, mit der Größe Ihres Bildschirms identisch ist.

Mit einem `DEFINE WINDOW`-Statement wird ein Fenster nicht aktiviert; dies geschieht mit einem `SET WINDOW`-Statement oder der `WINDOW`-Klausel eines `INPUT`-Statements.



Anmerkung: Es gibt stets nur *ein* Natural-Fenster, und zwar das jeweils neueste. Vorherige Fenster mögen auf dem Bildschirm noch sichtbar sein, sind aber nicht länger aktiv und werden von Natural ignoriert. Sie können Eingaben nur im jeweils neuesten Fenster machen. Sollte der Platz hierzu nicht ausreichen, müssen Sie das Fenster vorher entsprechend vergrößern.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate												Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>	C	S						N	P	I									ja	nein
<i>operand2</i>	C	S						N	P	I									ja	nein
<i>operand3</i>	C	S						N	P	I									ja	nein
<i>operand4</i>	C	S						N	P	I									ja	nein
<i>operand5</i>	C	S				A	U												ja	nein

Syntax-Element-Beschreibung:

<i>window-name</i>	<p>Der <i>window-name</i> identifiziert das Fenster. Der Name darf bis zu 32 Stellen lang sein.</p> <p>Für Fensternamen gelten die gleichen Namenskonventionen wie für Benutzervariablen (siehe <i>Namen von Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i>).</p>
SIZE	<p>Mit der SIZE-Klausel bestimmen Sie die Größe des Fensters.</p> <p>Anmerkung: Auf Großrechnern benötigt Natural zusätzliche Spalten für sogenannte Attribut-Bytes, um Daten auf dem Bildschirm anzeigen zu können (auf anderen Plattformen sind solche Attribut-Bytes nicht erforderlich). Dadurch ist auf einem Großrechner der von einem Fenster überlagerte Bildschirmbereich breiter und der innerhalb eines Fensters sichtbare Seitenausschnitt schmäler als auf anderen Plattformen.</p> <p>Beispiel: Angenommen, die Größe eines Fensters ist mit <code>SIZE 5 * 15</code> definiert (d.h. mit einer Breite von 15 Spalten):</p> <ul style="list-style-type: none"> ■ Auf Großrechnern ist der vom Fenster überlagerte Bildschirmbereich 16 Spalten breit; der innerhalb des Fensters sichtbare Seitenausschnitt ist 14 Spalten breit ohne Rahmen bzw. 10 Spalten mit Rahmen. ■ Auf anderen Plattformen ist der von dem Fenster überlagerte Bildschirmbereich 15 Spalten breit, die Größe des sichtbaren Bereichs innerhalb des Fensters ist 15 Spalten ohne Rahmen bzw. 13 Spalten mit Rahmen breit.
SIZE AUTO	<p>Die Größe des Fensters wird von Natural automatisch zur Laufzeit festgelegt. Die Größe wird wie folgt durch die in das Fenster hineingenerierten Daten bestimmt:</p>

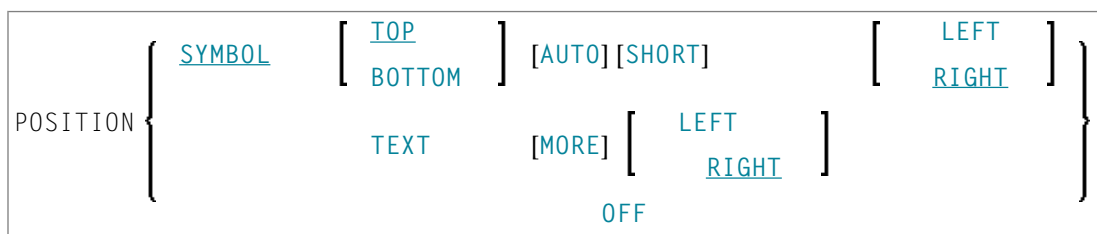
	<ul style="list-style-type: none"> ■ Die Zeilenanzahl des Fensters entspricht der Anzahl der generierten INPUT-Zeilen (plus gegebenenfalls der PF-Tastenleiste, der Meldungszeile und der Statistikzeile/Infoline). ■ Die Spaltenanzahl des Fensters wird durch die längste INPUT-Zeile bestimmt: Natural sucht, ausgehend von den Zeilenenden, nach dem signifikanten Byte, das am weitesten rechts in einer Zeile steht. Daher kann es vorkommen, dass reine Eingabefelder oder modifizierbare Felder (AD=A bzw. AD=M) abgeschnitten werden; um dies zu verhindern, können Sie entweder einen einbuchstabigen Text hinter so ein Feld stellen oder die gewünschte Fenstergröße explizit angeben mit: <code>SIZE operand1 * operand2</code> <p>Wenn Sie die SIZE-Klausel weglassen, gilt standardmäßig <code>SIZE AUTO</code>.</p> <p>Anmerkung: Der Titel ist nicht Bestandteil der Fensterdaten. Deshalb wird er abgeschnitten, wenn die Fenstergröße festgelegt worden ist, wie oben beschrieben, und der Titel länger als das Fenster ist.</p>
SIZE QUARTER	Die Größe des Fensters entspricht einem Viertel der physischen Bildschirmgröße.
SIZE <i>operand1 * operand2</i>	<p>Die Größe des Fensters beträgt <i>n</i> Zeilen mal <i>n</i> Spalten. <i>operand1</i> bestimmt die Anzahl der Zeilen, <i>operand2</i> die Anzahl der Spalten. Die beiden Operanden dürfen keine Dezimalstellen enthalten.</p> <p>Ist das Fenster gerahmt (FRAMED), so schließt die angegebene Größe den Rahmen mit ein.</p> <p>Die kleinstmögliche Fenstergröße ist:</p> <ul style="list-style-type: none"> ■ ohne Rahmen: 2 Zeilen mal 10 Spalten, ■ mit Rahmen: 4 Zeilen mal 13 Spalten. <p>Die größtmögliche Fenstergröße ist die Größe des physischen Bildschirms.</p>
BASE	Mit der BASE-Klausel bestimmen Sie die Position des Fensters auf dem physischen Bildschirm. Lassen Sie die BASE-Klausel weg, gilt standardmäßig <code>BASE CURSOR</code> .
BASE CURSOR	<p>Platziert die obere linke Ecke des Fensters an die aktuelle Cursor-Position. Die Cursor-Position ist die physische Position des Cursors auf dem Bildschirm.</p> <p>Wenn es aufgrund der Größe des Fensters nicht möglich ist, das Fenster an die Cursor-Position zu platzieren, platziert Natural es automatisch so dicht wie möglich an die gewünschte Position.</p>
BASE TOP/BOTTOM LEFT/RIGHT	Platziert das Fenster in die obere linke, untere linke, obere rechte bzw. untere rechte Ecke des physischen Bildschirms.
BASE <i>operand3/operand4</i>	<p>Platziert die obere linke Ecke des Fensters in die angegebene Zeile/Spalte auf dem physischen Bildschirm. <i>operand3</i> bestimmt die Zeilennummer, <i>operand4</i> die Spaltennummer. Die beiden Operanden dürfen keine Dezimalstellen enthalten.</p> <p>Ist es aufgrund der Größe des Fensters nicht möglich, das Fenster an die angegebene Position zu platzieren, erhalten Sie eine Fehlermeldung.</p>

REVERSED	REVERSED bewirkt, dass das Fenster invers angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird REVERSED ignoriert).
REVERSED CD= <i>background-color</i>	Dies bewirkt, dass das Fenster invers und der Fensterhintergrund in der angegebenen Farbe (<i>background-color</i>) angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird die betreffende Angabe ignoriert). Informationen über gültige Farbcodes. Session-Parameter CD in der <i>Parameter-Referenz-Dokumentation</i> .
TITLE <i>operand5</i>	Mit der TITLE-Klausel können Sie eine Überschrift für das Fenster angeben. Die angegebene Überschrift (<i>operand5</i>) wird zentriert in der oberen Rahmenzeile des Fensters angezeigt. Die Überschrift kann entweder als Textkonstante (in Apostrophen) oder als Inhalt einer Benutzervariablen angegeben werden. Ist die Überschrift länger als das Fenster, wird sie abgeschnitten. Die Überschrift wird nur angezeigt, wenn das Fenster gerahmt (FRAMED) ist; wenn FRAMED OFF angegeben ist, wird die TITLE-Klausel ignoriert. Anmerkung: Wenn der Titel nachfolgende Leerzeichen enthält, werden diese entfernt. Wenn das erste Zeichen des Titels ein Leerzeichen ist, wird hinter dem Titel automatisch ein Leerzeichen angehängt.
CONTROL	Mit der CONTROL-Klausel bestimmen Sie, ob die PF-Tastenleiste, die Meldungszeile und die Statistikzeile innerhalb oder außerhalb des Fensters angezeigt werden.
CONTROL WINDOW	CONTROL WINDOW zeigt die Zeilen innerhalb des Fensters an. Wenn Sie die CONTROL-Klausel weglassen, gilt standardmäßig CONTROL WINDOW.
CONTROL SCREEN	CONTROL SCREEN zeigt die Zeilen auf dem vollen physischen Schirm außerhalb des Fensters an.
FRAMED	Standardmäßig, d.h. wenn Sie die FRAMED-Klausel weglassen, wird das Fenster mit Rahmen angezeigt. Die obere und die untere Rahmenlinie sind cursor-sensitiv: Sie können im Fenster vor, zurück, nach links oder rechts blättern, indem Sie einfach den Cursor auf das entsprechende Symbol (<, -, +, oder >; siehe <i>position-clause</i> unten) stellen und FREIG drücken. Wenn keine Symbole angezeigt werden, können Sie im Fenster vor- oder zurückblättern, indem Sie den Cursor in die obere (zum Zurückblättern) bzw. untere (zum Vorblättern) Rahmenlinie platzieren und FREIG drücken. Anmerkung: Falls das Fenster kleiner als 4 Zeilen mal 13 Spalten ist, ist der Rahmen nicht sichtbar.
FRAMED OFF	Wenn Sie FRAMED OFF angeben, wird die Rahmung und alles mit dem Rahmen zusammenhängende (Überschrift für das Fenster und Positionierungsinformationen) ausgeschaltet.
FRAMED (CD= <i>frame-color</i>)	Dies bewirkt, dass der Fensterrahmen in der angegebenen Farbe (<i>frame-color</i>) angezeigt wird (falls ein Farbbildschirm verwendet wird; falls nicht, wird die Farbangabe ignoriert).

	Informationen über gültige Farbcodes siehe Session-Parameter CD in der <i>Parameter-Referenz</i> . Anmerkung: Bei Natural for Windows wird diese Angabe ignoriert.
<i>position-clause</i>	Die POSITION-Klausel wird nur auf Großrechnern ausgewertet: auf allen anderen Plattformen wird sie ignoriert. Einzelheiten, siehe <i>POSITION-Klausel</i> weiter unten.

POSITION-Klausel

Die POSITION-Klausel wird nur auf Großrechnern ausgewertet, auf allen anderen Plattformen wird sie ignoriert.



Mit der POSITION-Klausel steuern Sie die Anzeige von Informationen über die Position des Fensters auf der logischen Seite: Im Rahmen des Fensters wird angezeigt, in welche Richtungen die logische Seite nach oben, unten, links und rechts über das aktuelle Fenster hinausgeht. Dies gilt nur, falls die logische Seite größer als das Fenster ist; falls nicht, wird die POSITION-Klausel ignoriert.

Wenn Sie die POSITION-Klausel nicht angeben, gilt standardmäßig POSITION SYMBOL TOP RIGHT.

POSITION SYMBOL	Bewirkt, dass die Positionsinformationen in Form von Symbolen angezeigt werden: More: < - + > Die Informationen werden in der oberen und/oder unteren Rahmenzeile angezeigt.
TOP/BOTTOM	Bestimmt, ob die Positionsinformationen in der oberen oder unteren Rahmenzeile angezeigt werden.
AUTO	Ist nur relevant, wenn die logische Seite horizontal vollständig im Fenster sichtbar ist, d.h. wenn nur - und/oder + angezeigt werden soll. In diesem Fall schaltet AUTO automatisch von den Symbolen auf die Wörter Top, Bottom bzw. More um.
SHORT	Bewirkt, dass das Wort More: vor den Symbolen < - + > nicht angezeigt wird.
LEFT/RIGHT	Bestimmt, ob die Positionsinformationen im linken oder im rechten Teil der Rahmenzeile angezeigt werden.
POSITION TEXT	Bewirkt, dass die Positionsinformationen in der oberen und/oder unteren Rahmenzeile in Textform angezeigt werden, und zwar mit den Wörtern More, Top und Bottom. Die Wörter sind sprachabhängig und können durch entsprechendes Setzen des Sprachcodes auch in einer anderen Sprache angezeigt werden.

POSITION TEXT MORE	Unterdrückt die Wörter <code>Top</code> und <code>Bottom</code> und zeigt nur das Wort <code>More</code> je nach Situation in der oberen oder unteren Rahmenzeile oder in beiden an.
LEFT/RIGHT	Bestimmt, ob die Positionsinformationen im linken oder rechten Teil der Rahmenzeile angezeigt werden.
POSITION OFF	Bewirkt, dass überhaupt keine Positionsinformationen angezeigt werden.

Schutz von Eingabefeldern in einem Fenster

Für Eingabefelder (`AD=A` oder `AD=M`), die sich nicht vollständig innerhalb des Fensters befinden, gilt folgendes:

- Ein Eingabefeld, dessen Anfang außerhalb des Fensters liegt, wird immer zu einem geschützten Feld gemacht.
- Ein Eingabefeld, das im Fenster beginnt, aber außerhalb des Fensters endet, wird nur dann geschützt, wenn der Wert, den es enthält, nicht vollständig innerhalb des Fensters sichtbar ist. Bitte beachten Sie, dass es hierbei darauf ankommt, ob die *Wertlänge*, nicht die *Feldlänge*, über das Fenster hinausgeht. Füllzeichen (wie mit dem Profilparameter `FC` angegeben) zählen nicht als Teil des Wertes.

Falls Sie in ein derart geschütztes Eingabefeld Eingaben machen möchten, müssen Sie zunächst die Fenstergröße so ändern, dass sich der Anfang des Feldes/das Ende des Feldwertes innerhalb des Fensters befindet.

Aufrufen unterschiedlicher Fenster

Ein `DEFINE WINDOW`-Statement darf nicht innerhalb einer logischen Bedingung stehen. Wollen Sie in Abhängigkeit von einer Bedingung unterschiedliche Fenster aufrufen, verwenden Sie dazu verschiedene `SET WINDOW`-Statements (bzw. `INPUT`-Statements mit `WINDOW`-Klausel) in einer Bedingung.

Beispiel

```

** Example 'DWDEX1': DEFINE WINDOW
*****
DEFINE DATA LOCAL
01 #I (P3)
END-DEFINE
*
SET KEY PF1='%W<<' PF2='%W>>' PF4='%W--' PF5='%W++'
*
DEFINE WINDOW WIND1
    SIZE QUARTER
    BASE TOP RIGHT
    FRAMED ON POSITION SYMBOL AUTO
*
SET WINDOW 'WIND1'
FOR #I = 1 TO 10
    WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
END-FOR
*
END
    
```

Ausgabe des Programms DWDEX1:

```

> r                                     +-----More:      + >+
All      ....+....1....+....2....+....3.. ! Page      1      !
0010 ** Example 'DWDEX1': DEFINE WIND !                1 THIS !
0020 ***** !                2 THIS !
0030 DEFINE DATA LOCAL !                3 THIS !
0040 01 #I (P3) !                4 THIS !
0050 END-DEFINE !                5 THIS !
0060 * !                6 THIS !
0070 SET KEY PF1='%W<<' PF2='%W>>' PF !                7 THIS !
0080 * ! MORE !
0090 DEFINE WINDOW WIND1 +-----+
0100     SIZE QUARTER
0110     BASE TOP RIGHT
0120     FRAMED ON POSITION SYMBOL AUTO
0130 *
0140 SET WINDOW 'WIND1'
0150 FOR #I = 1 TO 10
0160     WRITE 25X #I 'THIS IS SOME LONG TEXT' #I
0170 END-FOR
0180 *
0190 END
    
```


```
0200 .....1.....2.....3.....4.....5..... S 19 L 1
```


51

DEFINE WORK FILE

- Funktion 278
- Syntax-Beschreibung 278

```
DEFINE WORK FILE n { operand1 [TYPE operand2] } [ATTRIBUTES {operand3}...]
```

 **Anmerkung:** Die in eckigen Klammern [...] gezeigten Elemente sind optional, aber mindestens eines muss bei diesem Statement angegeben werden.

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [CLOSE WORK FILE](#) | [READ WORK FILE](#) | [WRITE WORK FILE](#)


Gehört zur Funktionsgruppe: *Verarbeitung von Arbeitsdateien/PC-Dateien*

Funktion

Das Statement `DEFINE WORK FILE` dient dazu, innerhalb einer Natural-Anwendung einer Natural-Arbeitsdateinummer einen Dateinamen zuzuweisen.

Damit können Sie Arbeitsdatei-Zuweisungen innerhalb Ihrer Natural-Session dynamisch vornehmen bzw. ändern, sowie auf anderer Ebene gemachte Arbeitsdatei-Zuweisungen überschreiben.


Ist bei der Ausführung dieses Statements die angegebene Arbeitsdatei bereits offen, bewirkt dieses Statement implizit, dass die Arbeitsdatei geschlossen wird.

 **Anmerkung:** Bezüglich Unicode- und Codepage-Support siehe *Work Files and Print Files on Windows, UNIX and OpenVMS Platforms* in der *Unicode and Code Page Support*-Dokumentation.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S			A	U												yes	no
<i>operand2</i>	C	S			A	U												yes	no
<i>operand3</i>	C	S			A	U												yes	no

 **Anmerkung:** Wenn ein Operand im Format in Unicode (UTF-16) angegeben wird, erfolgt vor der Auswertung eine Umsetzung der Zeichen gemäß der Session-Codepage.

Syntax-Element-Beschreibung:

<p>DEFINE WORK FILE <i>n</i></p>	<p>Nummer der Arbeitsdatei:</p> <p><i>n</i> ist die Nummer der Arbeitsdatei (1 bis 32). Dies ist die Nummer, wie sie in einem READ WORK FILE-, WRITE WORK FILE- oder CLOSE WORK FILE-Statement verwendet wird.</p>	
<p><i>operand1</i></p>	<p>Name der Arbeitsdatei:</p> <p><i>operand1</i> ist der Name der Arbeitsdatei.</p> <p>Der Dateiname (<i>operand1</i>) darf Umgebungsvariablen enthalten. Es ist möglich, die physischen Arbeitsdateinamen zu verwenden. Wenn eine Datei mit dem angegebenen Namen nicht existiert, dann wird sie erstellt.</p> <p>Wenn <i>operand1</i> nicht angegeben ist, wird der Wert von <i>operand1</i> bestimmt, indem der im Configuration Utility (in der Parameterdatei) gespeicherte Arbeitsdateiname für die entsprechende Arbeitsdateinummer genommen wird.</p> <p>Anmerkung: Wenn <i>operand1</i> nicht angegeben ist, dann ist das Verhalten von Natural für Mainframes und Natural für Windows/UNIX/OpenVMS unterschiedlich.</p>	
<p>TYPE <i>operand2</i></p>	<p>TYPE-Klausel:</p> <p><i>operand2</i> gibt den Arbeitsdateityp an.</p> <p>Beim Wert von <i>operand2</i> wird nicht nach Groß- und Kleinbuchstaben unterschieden. Der Wert muss in Anführungszeichen stehen oder in einer alphanumerischen Variable angegeben werden.</p>	
	<p>ATTRIBUTES {<i>operand3</i>} . . .</p>	<p>Bestimmt den Dateityp an Hand der Namensweiterung.</p> <p>Format: Abhängig vom Arbeitsdateityp.</p> <p>Anmerkung: Der Dateityp TRANSFER kann nicht vom Arbeitsdateityp DEFAULT bestimmt werden. Sie müssen ausdrücklich TRANSFER als den Dateityp definieren, den Sie benutzen möchten.</p>
	<p>TRANSFER</p>	<p>Wird zur Datenübertragung zwischen einem PC und Entire Connection benutzt.</p> <p>Dieser Arbeitsdateityp steht für eine Datenverbindung zwischen einer Natural-Session auf UNIX oder OpenVMS und einem Entire Connection-Terminal auf einem PC.</p> <p>Format: ENTIRE CONNECTION</p> <p>Anmerkung:</p> <p>1. Dieser Arbeitsdateityp kann nicht zusammen mit der ATTRIBUTES-Klausel benutzt werden.</p>

		2. Dieser Arbeitsdateityp steht unter Windows nicht zur Verfügung.
	SAG	Format: binär
	ASCII	Dateien im ASCII-Format sind „Text“-Dateien mit Datensätzen, die mit einem Zeilenvorschub [Wagenrücklauf] beendet sind. Format: ASCII
	ASCII-COMPRESSED	Eine Datei im ASCII-Format, bei der alle abschließenden Leerzeichen entfernt wurden. Format: ASCII
	ENTIRECONNECTION	Mit diesem Arbeitsdateityp können Sie direkt aus/in eine/r Arbeitsdatei im Entire Connection-Format auf der lokalen Platte lesen und schreiben (z.B. mit den Statements READ und WRITE). Format: ENTIRE CONNECTION Anmerkung: Dieser Arbeitsdateityp steht auf PCs, UNIX und OpenVMS zur Verfügung. Der Transfer zu einem PC ist nicht möglich. Das Entire Connection-Terminal wird bei diesem Prozess nicht benutzt.
	UNFORMATTED	Eine vollkommen unformatierte Datei. Es werden keine Formatier-Informationen geschrieben (weder für Felder noch für Sätze). Format: UNFORMATTED
	PORTABLE	Eine Datei mit der dynamische Variablen präzise verarbeitet und auch transportiert werden können: zum Beispiel von einer Little Endian-Maschine auf eine Big Endian-Maschine, und umgekehrt. Format: PORTABLE
	CSV	Datei mit „Comma-Separated Values“. Jeder Datensatz wird in eine Zeile der Datei geschrieben. Standardmäßig wird kein Header geschrieben. Das Standardzeichen zum Trennen der Datenfelder ist ein Semikolon (;). Weitere Informationen siehe <i>Work Files</i> in der <i>Configuration Utility</i> -Dokumentation.
ATTRIBUTES { <i>operand3</i> }...}	ATTRIBUTES-Klausel: <i>operand3</i> gibt ein Arbeitsdatei-Attribut an.	

Es können mehrere Attribute angegeben werden, die jeweils durch ein Komma oder Leerzeichen getrennt sind. Beispiel:

```
DEFINE WORK FILE ATTRIBUTES 'APPEND,KEEP'
```

Wenn für denselben Attributtyp mehrere Werte angegeben sind, wird der letzte Wert genommen. Beispiel:

```
DEFINE WORK FILE ATTRIBUTES 'APPEND,NOAPPEND'
```

In diesem Fall wird NOAPPEND genommen.

Beispiel für BOM/NOBOM-Benutzung:

```
...
DEFINE WORK FILE 11 'x.tmp' ATTRIBUTES 'BOM'
*
* write work file with BOM
*
DEFINE WORK FILE 11 'x.tmp' ATTRIBUTES 'NOBOM'
*
* write work file without BOM
...
```

Anmerkung: Wenn *operand3* weggelassen wird, wird der entsprechende Wert genommen, der im Configuration Utility (Parameterdatei) definiert ist.

Folgendes ist eine Übersicht der Attributtypen und ihrer möglichen Werte:

Anfügemodus ein-/ausschalten:

APPEND	Aktiviert den Anfügemodus. In diesem Modus werden neue Datensätze am Ende der Datei angehängt.
NOAPPEND	Deaktiviert den Anfügemodus. Die Datei wird von Anfang an neu geschrieben. Dies ist die Standardeinstellung.

Schreibmodus für BOM ein-/ausschalten:

BOM	Aktiviert den Schreibmodus für BOM (Byte Order Mark). Ein BOM wird vor den Arbeitsdateidaten geschrieben.
NOBOM	Deaktiviert den Schreibmodus für BOM. Ein BOM wird nicht geschrieben. Dies ist die Standardeinstellung.

Datei nach Schließen der Arbeitsdatei behalten/löschen

DELETE	Die Arbeitsdatei wird nach einer Operation, die die Arbeitsdatei schließt, gelöscht.
KEEP	Die Arbeitsdatei wird nach einer Operation, die die Arbeitsdatei schließt, behalten. Dies ist die Standardeinstellung.

Weitere Informationen zu Arbeitsdateien finden Sie unter *Work File Formats* in der *Operations*-Dokumentation.

52 DELETE

▪ Funktion	284
▪ Einschränkung	284
▪ Syntax-Beschreibung	284
▪ Datenbank-spezifische Anmerkungen	285
▪ Beispiele	285

DELETE [RECORD] [IN] [STATEMENT] [(*r*)]

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement DELETE dient dazu, einen Datensatz von der Datenbank zu löschen.

Hold-Status

Das Vorhandensein eines DELETE-Statements bewirkt, dass alle Datensätze, die mit dem betreffenden READ- oder FIND-Statement gelesen werden, in den Hold-Status gestellt werden.

Die Hold-Logik ist im Kapitel *Datenbankzugriffe* im *Leitfaden zur Programmierung* beschrieben.

Einschränkung

Das DELETE-Statement darf nicht mit einem FIND-, READ- oder GET-Statement in derselben Source-code-Zeile stehen.

Syntax-Beschreibung

(<i>r</i>)	Statement-Referenz: Die Notation (<i>r</i>) dient dazu, das Statement zu referenzieren, das verwendet wurde, um den zu löschenden Datensatz auszuwählen/zu lesen. Wenn keine Statement-Referenz angegeben wird, referenziert das DELETE-Statement die jeweils innerste aktive Verarbeitungsschleife, mit der der Datensatz, der gelöscht werden soll, ausgewählt/gelesen wurde.
-------------------	--

Datenbank-spezifische Anmerkungen

SQL-Datenbanken	<p>Mit dem DELETE-Statement können Sie eine Reihe aus einer Datenbank-Tabelle löschen. Das DELETE-Statement entspricht dem SQL-Statement DELETE WHERE CURRENT OF CURSOR-NAME, d.h. nur die zuletzt gelesene Reihe kann gelöscht werden.</p> <p>Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht gelöscht werden.</p>
XML-Datenbanken	<p>Mit dem DELETE-Statement kann ein XML-Objekt aus einer XML-Datenbank gelöscht werden. Das impliziert, dass nur der zuletzt gelesene Datensatz gelöscht werden kann.</p>

Beispiele

- [Beispiel 1 — DELETE-Statement](#)
- [Beispiel 2 — DELETE-Statement](#)

Beispiel 1 — DELETE-Statement

In diesem Beispiel werden alle Datensätze mit Namen ALDEN gelöscht.

```

** Example 'DELEX1': DELETE
**
**
CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
/*
DELETE
END TRANSACTION
/*
AT END OF DATA
  WRITE NOTITLE *NUMBER 'RECORDS DELETED'
END-ENDDATA
END-FIND
END

```

Beispiel 2 — DELETE-Statement

Falls in der VEHICLES-Datei für die Person mit Namen ALDEN keine Datensätze gefunden werden, wird von der EMPLOYEES-Datei der Datensatz von ALDEN gelöscht.

```
** Example 'DELEX2': DELETE
**
**
CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
*
EMPL. FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
/*
  VEHC. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMPL.)
    IF NO RECORDS FOUND
      /*
        DELETE (EMPL.)
      /*
    END TRANSACTION
  END-NOREC
END-FIND
/*
END-FIND
END
```

53 DISPLAY

▪ Funktion	288
▪ Syntax-Beschreibung	288
▪ Standardwerte	301
▪ Beispiele	302

```
DISPLAY [( rep)] [options] {[/... ] [output-format] output-element} ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TITLE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Mit dem DISPLAY-Statement werden die Felder angegeben, deren Werte ausgegeben werden sollen. Die Ausgabe erfolgt in Spaltenform, mit einer Spalte pro Feld und einer Spaltenüberschrift.



Anmerkung: Die Statements WRITE und PRINT können benutzt werden, um Ausgaben in Freiformat (keine Spalten) zu erzeugen.

Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*:

- Steuerung der Ausgabe von Daten
- Statements DISPLAY und WRITE
- Index-Notation (n:n) für multiple Felder und Periodengruppen
- Spaltenüberschriften
- Layout einer Ausgabeseite

Syntax-Beschreibung

(rep)	<p>Report-Spezifikation:</p> <p>Mit der Notation (rep) können Sie einen bestimmten anderen Report angeben, auf den sich das Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls (rep) nicht angegeben wird, bezieht sich das DISPLAY-Statement auf den ersten ausgegebenen Report (Report 0).</p>
--------	--

	<p>Wenn diese Druckdatei für Natural als PC definiert ist, wird der Report auf den PC per Download heruntergeladen, siehe Beispiel 8.</p> <p>Weitere Informationen darüber, wie das Format eines Ausgabe-Reports gesteuert wird, siehe <i>Steuern der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
<i>options</i>	<p>Anzeige-Optionen:</p> <p>Einzelheiten siehe Anzeige-Optionen weiter unten.</p>
<i>output-format</i>	<p>Ausgabeformat-Definitionen:</p> <p>Einzelheiten siehe Ausgabeformat-Definitionen weiter unten.</p>
/	<p>Zeilenvorschub – Schrägstrich-Notation:</p> <p>Ein Schrägstrich (/) innerhalb eines Textelementes bewirkt einen Zeilenvorschub innerhalb des Textes.</p> <p>Ein Schrägstrich (/) zwischen zwei Ausgabeelementen bewirkt, dass das nachfolgende Element in derselben Spalte ausgegeben wird. Die Überschrift der Spalte wird gebildet, indem die Überschriften der beiden Elemente unmittelbar untereinander ausgegeben werden.</p> <p>Siehe auch die folgenden Themen im <i>Leitfaden zur Programmierung</i>:</p> <ul style="list-style-type: none"> ■ <i>Zeilenvorschub – die Schrägstrich-Notation (/)</i> ■ <i>Beispiel für Zeilenvorschub in DISPLAY-Statement</i> ■ <i>Spaltenüberschriften unterdrücken – die Schrägstrich-Notation ('/')</i>
<i>output-element</i>	<p>Ausgabe-Element:</p> <p>Einzelheiten siehe Ausgabe-Element weiter unten.</p>

Anzeige-Optionen

<code>[NOTITLE] [NOHDR] [[AND] [GIVE] [SYSTEM] FUNCTIONS] [(statement-parameters)]</code>

Syntax-Element-Beschreibung:

NOTITLE	<p>Unterdrückung der Standard-Kopfzeile:</p> <p>Natural generiert für jede über ein DISPLAY-Statement ausgegebene Seite eine Standardkopfzeile. Diese Standardkopfzeile enthält die laufende Seitennummer, Datum und Uhrzeit. Die Uhrzeit wird zu Beginn der Programmausführung bzw. zu Beginn des Jobs (Batch-Betrieb) gesetzt. Die Standardkopfzeile kann mit einer eigenen Kopfzeile überschrieben werden, die mit einem <code>WRITE TITLE</code>-Statement angegeben wird. Das Schlüsselwort NOTITLE innerhalb des</p>
----------------	---

	<p>DISPLAY-Statements bewirkt, dass die Generierung einer Standardkopfzeile unterdrückt wird.</p> <p>Beispiele:</p> <ul style="list-style-type: none"> ■ Generierte Standard-Titelzeile wird ausgegeben: <pre>DISPLAY NAME</pre> ■ Eigene Titelzeile wird ausgegeben: <pre>DISPLAY NAME WRITE TITLE 'user-title'</pre> ■ Keine Titelzeile wird ausgegeben: <pre>DISPLAY NOTITLE NAME</pre> <p>Anmerkung: Wird die NOTITLE-Option verwendet, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.</p>
<p>NOHDR</p>	<p>Spaltenüberschriften:</p> <p>Für jede mittels eines DISPLAY-Statements ausgegebene Spalte von Feldwerten wird eine Spaltenüberschrift ausgegeben; hierbei gilt folgendes:</p> <ul style="list-style-type: none"> ■ Sie können mit dem DISPLAY-Statement explizit eine Spaltenüberschrift angeben, und zwar (in Apostrophen) vor dem jeweiligen Feldnamen. Zum Beispiel: <pre>DISPLAY 'EMPLOYEE' NAME 'SALARY' SALARY</pre> ■ Wenn Sie mit dem DISPLAY-Statement keine Spaltenüberschrift angeben, verwendet Natural die im DEFINE DATA-Statement für das Feld angegebene Spaltenüberschrift. ■ Wenn für ein Datenbankfeld im DEFINE DATA-Statement keine Spaltenüberschrift angegeben ist, wird die im betreffenden DDM definierte Standardüberschrift genommen. ■ Ist im DDM keine Spaltenüberschrift definiert, wird stattdessen der (im DDM definierte) Feldname als Überschrift verwendet.

	<ul style="list-style-type: none"> ■ Wenn für eine Benutzervariable im DEFINE DATA-Statement keine Spaltenüberschrift angegeben ist, wird der Variablenname als Überschrift verwendet. Zur Definition von Spaltenüberschriften vgl. DEFINE DATA-Statement. <pre style="background-color: #f0f0f0; padding: 5px;">DISPLAY NAME SALARY #NEW-SALARY</pre> <ul style="list-style-type: none"> ■ Natural unterstreicht die Spaltenüberschriften immer und generiert zwischen Unterstreichung und den ausgegebenen Daten eine Leerzeile. ■ Enthält ein Programm mehrere DISPLAY-Statements, dann bestimmt das erste DISPLAY-Statement die Spaltenüberschriften; dies wird zur Kompilierungszeit ausgewertet. <p>Unterdrücken von Spaltenüberschriften:</p> <p>Um die Spaltenüberschrift für ein einzelnes Feld zu unterdrücken,</p> <ul style="list-style-type: none"> ■ geben Sie vor dem betreffenden Feldnamen die folgenden Zeichen (Apostroph-Schrägstrich-Apostroph) an: <pre style="background-color: #f0f0f0; padding: 5px;">' / '</pre> <p>Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 5px;">DISPLAY ' / ' NAME 'SALARY' SALARY</pre> <p>Sollen gar keine Spaltenüberschriften ausgegeben werden,</p> <ul style="list-style-type: none"> ■ geben Sie das Schlüsselwort NOHDR (für „No Header“) an: <pre style="background-color: #f0f0f0; padding: 5px;">DISPLAY NOHDR NAME SALARY</pre> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. NOHDR gilt nur beim ersten DISPLAY-Statement, da nachfolgende DISPLAY-Statements keine Spaltenüberschriften erzeugen können. 2. Wenn Sie NOTITLE und NOHDR verwenden, müssen Sie sie in der folgenden Reihenfolge angeben: DISPLAY NOTITLE NOHDR NAME SALARY
GIVE SYSTEM FUNCTIONS	<p>Benutzung von Systemfunktionen:</p> <p>Die GIVE SYSTEM FUNCTIONS-Klausel dient zur Auswertung der folgenden Natural-Systemfunktionen:</p> <p>AVER, COUNT, MAX, MIN, NAVER, NCOUNT, NMIN, SUM, TOTAL.</p> <p>Die Systemfunktionen werden ausgewertet, wenn das DISPLAY-Statement, das die GIVE SYSTEM FUNCTIONS-Klausel enthält, ausgeführt wird.</p>

	<p>Die Systemfunktionen können anschließend von einem Statement, das aufgrund einer End-of-Page-Bedingung ausgeführt wird, referenziert werden.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Pro Report darf nur ein DISPLAY-Statement eine GIVE SYSTEM FUNCTIONS-Klausel enthalten. Die Auswertung von Systemfunktionen über DISPLAY GIVE SYSTEM FUNCTIONS geschieht seitenbezogen, d.h. bei Beginn einer neuen Seite werden alle Systemfunktionen außer TOTAL wieder auf Null gesetzt. 2. Bei der Verwendung von Systemfunktionen mit einem DISPLAY-Statement, das sich in einer Subroutine befindet, muss die End-of-Page-Verarbeitung innerhalb derselben Subroutine stattfinden. <p>Siehe auch Beispiel 2 – DISPLAY-Statement mit GIVE SYSTEM FUNCTIONS-Klausel.</p>
<p><i>statement-parameters</i></p>	<p>Parameter-Definition auf Statement-Ebene:</p> <p>Sie können (in Klammern) Session-Parameter auf Statement-Ebene setzen, die dann für das DISPLAY-Statement statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS-Statement (nur im Reporting Mode) oder FORMAT-Statement gesetzten Parameter gelten.</p> <p>Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Anmerkung: Die hier gültigen Parameter-Einstellungen kommen nur für Variablenfelder in Betracht, haben aber keine Auswirkung auf Text-Konstanten. Möchten Sie Feldattribute für eine Text-Konstante setzen, müssen sie explizit für dieses Element gesetzt werden. Siehe Parameter-Definition auf Elementebene (Feldebene).</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Liste der Parameter ■ Beispiel für die Benutzung von Parametern auf Statement- und Elementebene (Feldebene) ■ Beispiel 7 – DISPLAY-Statement mit Parametern auf Statement-/Elementebene (Feldebene)

Liste der Parameter

Session-Parameter, die beim DISPLAY-Statement angegeben werden können		Spezifikation (S = auf Statement-Ebene, E = auf Elementebene)
AD	Attribute Definition	SE
AL	Alphanumeric Length for Output	SE
CD	Color Definition	SE
CV	Control Variable	SE
DF	Date Format	SE
DL	Display Length for Output	SE
DY	Dynamic Attributes	SE
EM	Edit Mask	SE
EMU	Unicode Edit Mask	E
ES	Empty Line Suppression	S
FC	Filler Character	SE
FL	Floating Point Mantissa Length	SE
GC	Filler Character for Group Headers	SE
HC	Header Centering	SE
HW	Heading Width	SE
IC	Insertion Character	SE
IS	Identical Suppress	SE
LC	Leading Characters	SE
LS	Line Size	S
MC	Multiple-Value Field Count	S
MP	Maximum Number of Pages of a Report	S
NL	Numeric Length for Output	SE
PC	Periodic Group Count	S
PM	Print Mode	SE
PS	Page Size	S
SF	Spacing Factor	SE
SG	Sign Position	SE
TC	Trailing Characters	SE
UC	Underlining Character	SE
ZP	Zero Printing	SE

Die einzelnen Parameter sind in der *Parameter-Referenz*-Dokumentation beschrieben.

Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*:

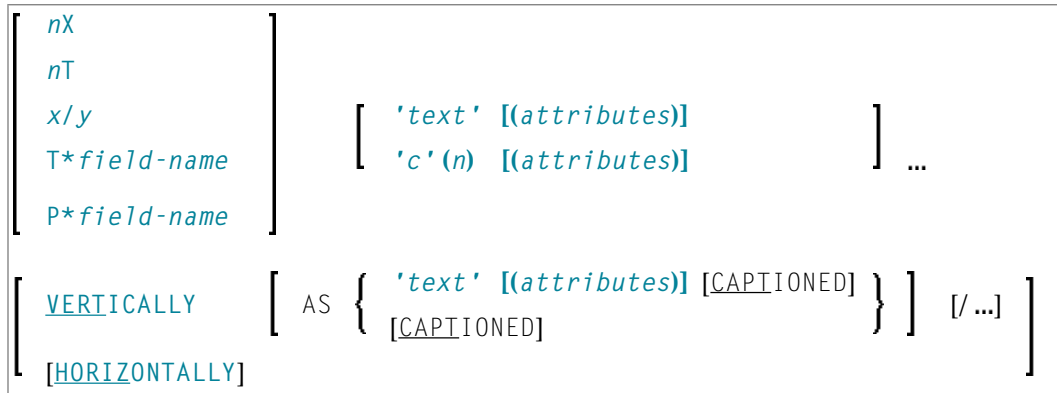
- Spaltenüberschriften zentrieren – der HC-Parameter
- Breite der Spaltenüberschriften – der HW-Parameter
- Füllzeichen für Überschriften – die Parameter FC und GC
- Unterstreichungszeichen für Überschriften und Kopfzeilen – der UC-Parameter

Beispiel für die Benutzung von Parametern auf Statement- und Elementebene (Feldebene)

```

DEFINE DATA LOCAL
1 VARI (A4)      INIT <'1234'>                /*      Output
END-DEFINE                                           /*      Produced
*                                                    /*      -----
DISPLAY NOHDR      'Text'      '='      VARI      /*      Text 1234
DISPLAY NOHDR (AD=U) 'Text'      '='      VARI      /*      Text  1234
DISPLAY NOHDR      'Text' (AD=U) '='      VARI (AD=U) /*      Text 1234
DISPLAY NOHDR      'Text' (AD=U) '='      VARI      /*      Text  1234
END
    
```

Ausgabeformat-Definitionen



Feldpositionierung

nX	<p>Spaltenabstand:</p> <p>Mit dieser Notation fügen Sie zwischen den auszugebenden Spalten n Leerstellen ein.</p>
------	---

	<p>Beispiel:</p> <pre>DISPLAY NAME 5X SALARY</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 1 – DISPLAY-Statement mit Notation nX- und nT (weiter unten) ■ Spaltenabstand – der SF-Parameter und die Notation nX (im Leitfaden zur Programmierung)
<i>nT</i>	<p>Setzen von Tabulatoren:</p> <p>Mit dieser Notation setzen Sie Tabulatoren, d.h. die Ausgabe eines Wertes beginnt ab Spalte <i>n</i>.</p> <p>Zurückpositionieren ist nicht erlaubt.</p> <p>Im folgenden Beispiel wird NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben:</p> <pre>DISPLAY 25T NAME 50T SALARY</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 1 – DISPLAY-Statement mit Notation nX- und nT (weiter unten) ■ Tabulator-Notation nT (im Leitfaden zur Programmierung)
<i>x/y</i>	<p>x/y-Positionierung:</p> <p>Mit dieser Notation erreichen Sie, dass ein Feld <i>x</i> Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte <i>y</i> ausgegeben wird.</p> <p><i>y</i> darf nicht 0 sein.</p> <p>Zurückpositionieren ist nicht erlaubt.</p>
<i>T*field-name</i>	<p>Feldbezogene Positionierung:</p> <p>Mit dieser Notation wird die Position eines Feldes nach der Position eines in einem vorhergehenden DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausgerichtet.</p> <p>Zurückpositionieren ist nicht erlaubt.</p>
<i>P*field-name</i>	<p>Feld- und zeilenbezogene Positionierung:</p> <p>Mit dieser Notation werden Position und Ausgabezeile eines Feldes nach denen eines in einem vorhergehenden DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausgerichtet. Dies wird meist bei vertikalen Ausgaben eingesetzt.</p> <p>Zurückpositionieren ist nicht erlaubt.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 3 – DISPLAY-Statement mit der Notation P* (weiter unten)

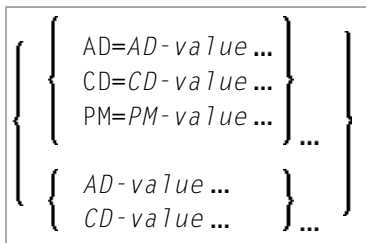
	<ul style="list-style-type: none"> ■ <i>Tab-Notation P*field</i> (im Leitfaden zur Programmierung)
--	---

Spaltenüberschriften

<p>'text'</p>	<p>Textzuweisung:</p> <p>In Apostrophen angegebener Text ('text') vor einem Feld wird als Spaltenüberschrift verwendet. Ein Schrägstrich in Apostrophen '/' vor einem Feldnamen bewirkt, dass für dieses Feld keine Spaltenüberschrift ausgegeben wird. Beispiel:</p> <pre>DISPLAY 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT</pre> <p>Werden vor einem Feldnamen mehrere Textelemente 'text' angegeben, so wird das <i>letzte</i> als Spaltenüberschrift verwendet und die anderen werden in der Ausgabespalte vor dem Feldwert ausgegeben.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ <i>Eigene Spaltenüberschriften definieren</i> (im Leitfaden zur Programmierung) ■ <i>Text Notation, Text-Notation, Text für ein Statement definieren</i> (im Leitfaden zur Programmierung) ■ Beispiel 4 – DISPLAY-Statement mit 'text', 'c'(n) und Attribut-Notation (weiter unten)
<p>'c'(n)</p>	<p>Wiederholungszeichen:</p> <p>Das in Apostrophen (') stehende Zeichen <i>c</i> (character) wird <i>n</i>-mal unmittelbar vor dem Feldwert ausgegeben. Beispiel:</p> <pre>DISPLAY '*' (5) '=' NAME</pre> <p>führt zur Ausgabe von:</p> <pre>***** SMITH</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ <i>Text-Notation, n mal vor einem Feldwert anzuzeigendes Zeichen definieren</i> (im Leitfaden zur Programmierung) ■ Beispiel 4 – DISPLAY-Statement mit 'text', 'c'(n) und Attribut-Notation (weiter unten)

Ausgabe-Attribute

attributes gibt die für die Text-Anzeige zu benutzenden Ausgabe-Attribute an. Es gibt die folgenden Attribute:



Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD - Attribute Definition, Abschnitt Feldanzeige*
- *CD - Color Definition*
- *PM - Print Mode*



Anmerkung: Der Compiler akzeptiert tatsächlich mehr als einen Attributwert für ein Ausgabefeld. Zum Beispiel können Sie Folgendes angeben: AD=BDI. In solch einem Fall gilt allerdings nur der letzte Wert. Im hier gezeigten Beispiel erhält nur der Wert I Gültigkeit, und das Ausgabefeld wird intensiviert (hell hervorgehoben) angezeigt.

Vertikale/Horizontale Ausgabe

Mit `DISPLAY VERT` werden die Werte mehrerer Felder nicht in Spalten nebeneinander sondern in einer Spalte untereinander ausgegeben. Eine neue Spalte wird durch Angabe des Schlüsselwortes `VERT` oder `HORIZ` initialisiert.

Die Ausgabe von Spaltenüberschriften wird beim `DISPLAY VERT` über die `AS`-Klausel gesteuert:

VERTICALLY	<p>Vertikale Spaltenausrichtung. Es wird keine Spaltenüberschrift erzeugt, wenn die <code>AS</code>-Klausel weggelassen wird.</p> <p>Beispiel:</p> <pre>DISPLAY VERT NAME SALARY</pre> <p>Siehe auch Beispiel <i>DISPLAY VERT ohne AS-Klausel</i> im Leitfaden zur Programmierung.</p>
-------------------	--

AS 'text'	<p>Vertikale Spaltenausrichtung. Wenn AS 'text' angegeben wird, wird der in Apostrophen stehende Text als Spaltenüberschrift ausgegeben.</p> <p>Siehe auch Beispiel <i>DISPLAY VERT AS 'text'</i> im Leitfaden zur Programmierung.</p> <p>Wenn Sie einen Schrägstrich (/) in der Zeichenkette 'text' angeben, werden mehrere Zeilen mit Spaltenüberschriften erzeugt.</p> <p>Beispiel:</p> <pre>DISPLAY VERT AS 'LAST/NAME' NAME</pre>
AS 'text' CAPTIONED	<p>Vertikale Spaltenausrichtung. Wenn AS 'text' CAPTIONED angegeben wird, wird 'text' als Spaltenüberschrift ausgegeben, und außerdem wird die Standard-Spaltenüberschrift bzw. der Feldname in jeder Ausgabezeile dem jeweiligen Feldwert vorangestellt.</p> <p>Beispiel:</p> <pre>DISPLAY VERT AS 'PERSONS/SELECTED' CAPTIONED NAME FIRST-NAME</pre> <p>Siehe auch Beispiel <i>DISPLAY VERT AS 'text' CAPTIONED</i> im Leitfaden zur Programmierung.</p>
AS CAPTIONED	<p>Vertikale Spaltenausrichtung. Wenn AS CAPTIONED angegeben wird, wird der standardmäßige Überschriften-Text für das Feld ausgegeben (entweder Überschriften-Text oder den Feldnamen).</p> <p>Beispiel:</p> <pre>DISPLAY VERT AS CAPTIONED NAME FIRST-NAME</pre>
HORIZONTALLY	<p>Horizontale Spaltenausrichtung. Dies ist der standardmäßige Anzeigemodus.</p>

Vertikale und horizontale Ausgaben können miteinander kombiniert verwendet werden, wobei der Wechsel von einer Form zur anderen durch die Angabe des jeweiligen Schlüsselwortes (VERT oder HORIZ) erfolgt.

Um die vertikale Ausgabe für ein einzelnes Ausgabeelement auszusetzen, geben Sie vor dem Element einen Gedankenstrich (-) ein.

Beispiel:

```
DISPLAY VERT NAME - FIRST-NAME SALARY
```

würde bewirken, dass `FIRST-NAME` neben `NAME` ausgegeben wird, während `SALARY` wieder vertikal, d.h. unter `NAME`, ausgegeben wird.

Normalerweise erzeugt ein `DISPLAY`-Statement eine horizontale Ausgabe, d.h. die Ausgabe erfolgt in Spalten, die nebeneinander angeordnet sind.

Bei der Generierung der Spaltenüberschriften hat Natural folgende Prioritäten:

1. Der im `DISPLAY`-Statement für eine Spaltenüberschrift angegebene `'text'`.
2. Bei Datenbankfeldern die im DDM definierte Standardspaltenüberschrift, bei Benutzervariablen der Variablenname.
3. Bei Datenbankfeldern der Name, unter dem das Feld im DDM definiert ist (wenn für das Datenbankfeld kein Überschriftentext definiert wurde).

Bei Gruppennamen wird eine Gruppen-Spaltenüberschrift für die gesamte Gruppe von Feldern erzeugt. Bei Angabe einer Gruppe kann nur diese Standard-Gruppenüberschrift durch eine eigene überschrieben werden.

Es sind bis zu 15 Spaltenüberschriftenzeilen erlaubt.

Die über ein `DISPLAY`-Statement erzeugte Ausgabe darf nicht über das Zeilenende hinausgehen; ist dies doch der Fall, gibt Natural eine entsprechende Fehlermeldung aus.

Weitere Informationen zur Benutzung der vertikalen/horizontalen Ausgabe siehe

- [Beispiel 5 – DISPLAY-Statement mit horizontaler Ausgabe](#)
- [Beispiel 6 – DISPLAY-Statement mit vertikaler und horizontaler Ausgabe](#)
- `DISPLAY VERT AS CAPTIONED` und `HORIZ` im Leitfaden zur Programmierung

Ausgabe-Element

<pre>{ 'text' [(attributes)] 'c'(n) [(attributes)] }...</pre>	<pre>['='] {operand1 [(parameters)]}</pre>
<pre>nX nT x/y</pre>	

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A N P I F B D T L G O	ja	nein

Syntax-Element-Beschreibung:

<i>nX</i>	<p>Spalten-Abstand:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
<i>nT</i>	<p>Setzen von Tabulatoren:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
<i>x/y</i>	<p>x/y-Positionierung:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
' <i>text</i> '	<p>Textzuweisung:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
' <i>c</i> (<i>n</i>)	<p>Wiederholungszeichen:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
' <i>text</i> ' '='	<p>Wird '<i>text</i>' '=' vor einem Feld angegeben, so wird <i>text</i> unmittelbar vor dem Feldwert ausgegeben. Dies gilt analog dazu für '<i>c</i>' (<i>n</i>) '='.</p>
' <i>c</i> ' (<i>n</i>) '='	<p>Beispiel:</p> <pre>DISPLAY '*****' '=' NAME</pre>
<i>attributes</i>	<p>Ausgabe-Attribute:</p> <p>Wie unter <i>Ausgabeformat-Definitionen</i> (siehe oben).</p>
<i>operand1</i>	<p>Das auszugebende Feld.</p>
<i>parameters</i>	<p>Parameter-Definiton auf Elementebene (Feldebene):</p> <p>Unmittelbar nach <i>operand1</i> können Sie auf Elementebene (Feldebene) in Klammern einen oder mehrere Parameter angeben, die dann für das betreffende Feld statt der entsprechenden auf Statement-Ebene oder mit einem GLOBALS-Kommando, SET GLOBALS-Statement (nur Reporting Mode)- oder FORMAT-Statement gesetzten Parameter gelten.</p> <p>Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Siehe auch:</p>

- *Liste der Parameter*
- *Beispiel für die Benutzung von Parametern auf Statement- und Elementebene (Feldebene)*

Standardwerte

Für ein DISPLAY-Statement gelten folgende Standardwerte:

- **Report-Breite:**
Die für Ausgaben gültige Standardbreite wird bei der Installation von Natural festgelegt; in der Regel beträgt sie im Batch-Betrieb 132 Stellen und entspricht im TP-Betrieb der Zeilenlänge des Terminals. Sie kann mit dem Session-Parameter `LS` überschrieben werden. Im TP-Betrieb setzt Natural die Parameter für Zeilenlänge (`LS`) und Seitenlänge (`PS`) unter Berücksichtigung der physischen Charakteristika des verwendeten Terminaltyps.
- **Terminal-Bildschirmausgabe:**
Erfolgt die DISPLAY-Ausgabe auf dem Bildschirm, dann beginnt die Ausgabe in der zweiten physischen Bildschirmspalte (da die erste Spalte für die etwaige Verwendung einer Attributstelle bei einem 3270-Terminal reserviert werden muss).
- **Druckausgabe auf Papier:**
Wird die DISPLAY-Ausgabe auf Papier ausgedruckt, dann beginnt die Ausgabe ganz links, d.h. in Spalte 1.
- **Abstandsfaktor:**
Standardmäßig wird zwischen zwei Ausgabeelementen eine Leerstelle eingefügt. Zwischen Ausgabespalten muss mindestens eine Leerspalte (reserviert für Terminal-Attribute) sein. Dieser Standardwert kann mit dem Session-Parameter `SF` überschrieben werden.
- **Feldausgabe:**
Die Breite einer Ausgabespalte richtet sich nach der Länge des Feldes oder der Spaltenüberschrift, je nachdem, was länger ist (es sei denn, der Parameter `HW` wird verwendet).
 - Ist die Überschrift kürzer als das Feld, wird sie über der Spalte zentriert (es sei denn, mit dem Parameter `HC=L` bzw. `HC=R` wird eine linksbündige bzw. rechtsbündige Ausgabe veranlasst).
 - Ist das Feld kürzer als die Überschrift, wird das Feld linksbündig zur Überschrift ausgerichtet.
 - Bei alphanumerischen Feldern werden die Feldwerte linksbündig im Feld ausgegeben, bei numerischen rechtsbündig.
 - Mit dem Parameter `AD=L` kann auch bei numerischen Feldern eine linksbündige Ausgabe erreicht werden.
 - Mit dem Parameter `AD=R` kann bei alphanumerischen Feldern eine rechtsbündige Ausgabe erreicht werden.
 - Bei vertikalen Ausgaben richtet sich die Breite einer Spalte nach dem längsten Feldwert bzw. der längsten Überschrift (es sei denn, der Parameter `HW` wird verwendet).

■ **Vorzeichen:**

Bei der Ausgabe eines numerischen Feldes wird eine Stelle vor dem Feld für die Ausgabe eines Vorzeichens reserviert. Die Vorzeichenstelle kann mit dem Session-Parameter SG unterdrückt werden.

■ **Seitenumbruch:**

Natural prüft vor der Ausführung eines DISPLAY-Statements, wann ein Seitenumbruch erforderlich ist. Während der Ausführung des DISPLAY-Statements werden keine Kopf- oder Fußzeilen generiert.

Beispiele

- Beispiel 1 — DISPLAY-Statement mit der Notation nX und nT
- Beispiel 2 — DISPLAY-Statement mit GIVE SYSTEM FUNCTIONS-Klausel
- Beispiel 3 — DISPLAY-Statement mit der Notation P*
- Beispiel 4 — DISPLAY-Statement mit 'text', 'c(n)' und Attribut-Notation
- Beispiel 5 — DISPLAY-Statement mit horizontaler Ausgabe
- Beispiel 6 — DISPLAY-Statement mit vertikaler und horizontaler Ausgabe
- Beispiel 7 — DISPLAY-Statement mit Parametern auf Statement-/Elementebene (Feldebene)
- Beispiel 8 — Report-Spezifikation mit für Natural als PC definierter Ausgabedatei

Beispiel 1 — DISPLAY-Statement mit der Notation nX und nT

```

** Example 'DISEX1': DISPLAY (with nX, nT notation)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  DISPLAY NOTITLE 5X NAME 50T JOB-TITLE
END-READ
*
END
    
```

Ausgabe des Programms DISEX1:

NAME	CURRENT POSITION
ABELLAN	MAQUINISTA
ACHIESON	DATA BASE ADMINISTRATOR
ADAM	CHEF DE SERVICE
ADKINSON	PROGRAMMER

Beispiel 2 — DISPLAY-Statement mit GIVE SYSTEM FUNCTIONS-Klausel

```

** Example 'DISEX2': DISPLAY (with GIVE SYSTEM FUNCTIONS)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 15
FORMAT PS=15
*
READ EMPLOY-VIEW
  DISPLAY GIVE SYSTEM FUNCTIONS
    PERSONNEL-ID NAME FIRST-NAME SALARY (1) CURR-CODE (1)
  AT END OF PAGE
    WRITE / 'SALARY STATISTICS:'
      / 7X 'MAXIMUM:' MAX(SALARY(1)) CURR-CODE (1)
      / 7X 'MINIMUM:' MIN(SALARY(1)) CURR-CODE (1)
      / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
END-READ
*
END

```

Ausgabe des Programms DISEX2:

Page 1 05-01-12 09:47:48

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
50005500	BLOND	ALEXANDRE	172000	FRA
50005300	MAIZIERE	ELISABETH	166900	FRA

DISPLAY

```
50004900  CAUDAL      ALBERT      167350 FRA
50004600  VERDIE       BERNARD     170100 FRA
50004200  VAUZELLE    BERNARD     159790 FRA
50004100  CHAPUIS     ROBERT      169900 FRA
50003800  JOUSSELIN   DANIEL      171990 FRA
50006900  BAILLET     PATRICK     188000 FRA
50007600  MARX        JEAN-MARIE  365700 FRA
```

```
SALARY STATISTICS:
  MAXIMUM:    365700 FRA
  MINIMUM:    159790 FRA
  AVERAGE:    192414 FRA
```

Beispiel 3 — DISPLAY-Statement mit der Notation P*

```
** Example 'DISEX3': DISPLAY (with P* notation)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1)
  2 BIRTH
  2 CITY
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY FROM 'N'
  DISPLAY NOTITLE NAME CITY
    VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
  SKIP 1
  AT BREAK OF CITY
    DISPLAY P*SALARY (1) AVER(SALARY (1))
  SKIP 1
  END-BREAK
END-READ
END
```

Ausgabe des Programms DISEX3:

NAME	CITY	BIRTH SALARY
WILCOX	NASHVILLE	1970-01-01 38000
MORRISON	NASHVILLE	1949-07-10 36000

37000

Beispiel 4 — DISPLAY-Statement mit 'text', 'c(n)' und Attribut-Notation

```

** Example 'DISEX4': DISPLAY (with 'c(n)' notation and attribute)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 DEPT
  2 LEAVE-DUE
  2 NAME
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY DEPT FROM 'T'
  IF LEAVE-DUE GT 40
    DISPLAY NOTITLE
      'EMPLOYEE' NAME           /* OVERRIDE STANDARD HEADER
      'LEAVE ACCUMULATED' LEAVE-DUE /* OVERRIDE STANDARD HEADER
      '**'(10)(I)              /* DISPLAY 10 '** INTENSIFIED

  ELSE
    DISPLAY NAME LEAVE-DUE
  END-IF
END-READ
*
END

```

Ausgabe des Programms DISEX4:

EMPLOYEE	LEAVE ACCUMULATED
LAVENDA	33
BOYER	33
CORREARD	45 *****
BOUVIER	19

Beispiel 5 — DISPLAY-Statement mit horizontaler Ausgabe

```

** Example 'DISEX5': DISPLAY (horizontal display)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
  2 SALARY      (1:2)
  2 CURR-CODE  (1:2)
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  DISPLAY NOTITLE NAME JOB-TITLE SALARY (1:2) CURR-CODE (1:2)
  SKIP 1
END-READ
*
END
    
```

Ausgabe des Programms DISEX5:

NAME	CURRENT POSITION	ANNUAL SALARY	CURRENCY CODE
ABELLAN	MAQUINISTA	1450000 1392000	PTA PTA
ACHIESON	DATA BASE ADMINISTRATOR	11300 10500	UKL UKL
ADAM	CHEF DE SERVICE	159980 0	FRA 0
ADKINSON	PROGRAMMER	34500 31700	USD USD

Beispiel 6 — DISPLAY-Statement mit vertikaler und horizontaler Ausgabe

```

** Example 'DISEX6': DISPLAY (vertical and horizontal display)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 JOB-TITLE
  2 SALARY      (1:2)
  2 CURR-CODE (1:2)
END-DEFINE
*
LIMIT 1
READ EMPL-VIEW BY NAME
  DISPLAY NOTITLE VERT AS CAPTIONED
    NAME CITY 'POSITION' JOB-TITLE
    HORIZ 'SALARY' SALARY (1:2) 'CURRENCY' CURR-CODE (1:2)
  /*
  SKIP 1
END-READ
END

```

Ausgabe des Programms DISEX6:

NAME CITY POSITION	SALARY	CURRENCY

ABELLAN	1450000	PTA
MADRID	1392000	PTA
MAQUINISTA		

Beispiel 7 — DISPLAY-Statement mit Parametern auf Statement-/Elementebene (Feldebene)

```

** Example 'DISEX7': DISPLAY (with parameters for statement/element)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 TELEPHONE
    3 AREA-CODE
    3 PHONE
END-DEFINE
*

```

DISPLAY

```
LIMIT 3
READ EMPL-VIEW BY NAME
  DISPLAY NOTITLE (AL=16 GC=+ NL=8 SF=3 UC=)
    PERSONNEL-ID NAME TELEPHONE (LC=< TC=>)
END-READ
END
```

Ausgabe des Programms DISEX7:

PERSONNEL ID	NAME	+++++TELEPHONE+++++	
		AREA CODE	TELEPHONE
=====	=====	=====	=====
60008339	ABELLAN	<1 >	<4356726 >
30000231	ACHIESON	<0332 >	<523341 >
50005800	ADAM	<1033 >	<44864858 >

Beispiel 8 — Report-Spezifikation mit für Natural als PC definierter Ausgabedatei

```
** Example 'PCDIEX1': DISPLAY and WRITE to PC
**
** NOTE: Example requires that Natural Connection is installed.
*****
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'          /* Data selection
WRITE (7) TITLE LEFT 'List of employees in New York' /
DISPLAY (7)          /* (7) designates the output file (here the PC).
  'Location'  CITY
  'Surname'   NAME
  'ID'        PERSONNEL-ID
END-FIND
END
```


54 DIVIDE

▪ Funktion	310
▪ Syntax-Beschreibung	310
▪ Beispiel	314

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [EXAMINE](#) | [MOVE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Mit dem Statement `DIVIDE` können Sie einen Operanden durch einen anderen dividieren.

Division durch Null:

Wird eine Division durch Null (0) versucht, d.h. wenn der Divisor (*operand1*), also die Zahl durch die geteilt wird, 0 ist, wird entweder eine entsprechende Fehlermeldung oder als Ergebnis 0 ausgegeben, je nachdem wie der Session-Parameter `ZD` (der in der *Parameter-Referenz-Dokumentation* beschrieben ist) gesetzt ist.

Syntax-Beschreibung

Für dieses Statement sind verschiedene Strukturen möglich:

- [Syntax 1](#) — `DIVIDE` ohne `GIVING`-Klausel
- [Syntax 2](#) — `DIVIDE`-Statement mit `GIVING`-Klausel
- [Syntax 3](#) — `DIVIDE`-Statement mit `REMAINDER`-Option

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax 1 — `DIVIDE` ohne `GIVING`-Klausel

```
DIVIDE [ROUNDED] operand1 INTO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate				Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S	A	N	N	P	I	F	ja	nein
<i>operand2</i>	C	S	A	M	N	P	I	F	ja	nein

Syntax-Element-Beschreibung:

<p><i>operand1</i> INTO</p> <p><i>operand2</i></p>	<p>Operanden:</p> <p><i>operand1</i> ist der Divisor, <i>operand2</i> ist der Dividend. Das Ergebnis wird in <i>operand2</i> (Ergebnisfeld) ausgegeben, daher sieht das Statement wie folgt aus:</p> <pre><oper2> := <oper2> / <oper1></pre> <p>Das Ergebnisfeld kann ein Datenbankfeld oder eine Benutzervariable sein. Wenn <i>operand2</i> eine Konstante oder eine nicht änderbare Natural- Systemvariable ist, ist die GIVING-Klausel erforderlich. Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. <i>operand2</i>) ausgewertet.</p>
ROUNDED	Wenn Sie das Schlüsselwort ROUNDED angeben, wird das Ergebnis gerundet.

Syntax 2 — DIVIDE-Statement mit GIVING-Klausel

```
DIVIDE [ROUNDED] operand1 INTO operand2 [GIVING operand3]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate							Referenzierung erlaubt	Dynam. Definition					
<i>operand1</i>	C	S	A	N			N	P	I	F						ja	nein
<i>operand2</i>	C	S	A	N			N	P	I	F						ja	nein
<i>operand3</i>		S	A			A	U	N	P	I	F	B*	T			ja	ja

* Format B von *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

<p><i>operand1</i> INTO</p> <p><i>operand2</i> GIVING</p> <p><i>operand3</i></p>	<p>Operanden:</p> <p><i>operand1</i> ist der Divisor, <i>operand2</i> ist der Dividend. Das Ergebnis wird in <i>operand3</i> (Ergebnisfeld) ausgegeben, folglich sieht das Statement wie folgt aus:</p> <pre style="background-color: #f0f0f0; padding: 5px;"><i><operand3> := <operand2> / <operand1></i></pre> <p>Wird ein Datenbankfeld als Ergebnisfeld verwendet, ändert sich durch die Division lediglich der programmintern verwendete Wert des Feldes; der in der Datenbank gespeicherte Wert des Feldes wird davon nicht berührt.</p> <p>Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. <i>operand3</i>) ausgewertet.</p> <p>Informationen zur Genauigkeit der Ergebnisse siehe Abschnitt <i>Genauigkeit von Ergebnissen bei arithmetischen Operationen</i> im Leitfaden zur Programmierung.</p>
<p>ROUNDED</p>	<p>Wenn Sie das Schlüsselwort ROUNDED angeben, wird das Ergebnis gerundet.</p>

Syntax 3 — DIVIDE-Statement mit REMAINDER-Option

```
DIVIDE operand1 INTO operand2 [GIVING operand3] REMAINDER operand4
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate			Referenzierung	Dynam.
	C	S	A	N	N	P I	erlaubt	Definition
<i>operand1</i>	C	S	A	N	N	P I	ja	nein
<i>operand2</i>	C	S	A	N	N	P I	ja	nein
<i>operand3</i>		S	A		A	U N P I F B* T	ja	ja
<i>operand4</i>		S	A		A	U N P I F B* T	ja	ja

* Format B von *operand3* und *operand4* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Divisor:</p> <p><i>operand1</i> ist der Divisor, d.h. die Anzahl oder Menge, durch die der Dividend dividiert werden soll, um den Quotienten zu erhalten.</p>
<i>operand2</i>	<p>Ergebnisfeld:</p> <p>Wird die GIVING-Klausel nicht benutzt, wird das Ergebnis in <i>operand2</i> ausgegeben. Das Ergebnisfeld kann ein Datenbankfeld oder eine Benutzervariable sein.</p> <p>Wenn <i>operand2</i> eine Konstante oder eine nicht änderbare Natural- Systemvariable ist, ist die GIVING-Klausel erforderlich.</p>
ROUNDED	Wenn Sie das Schlüsselwort ROUNDED angeben, wird das Ergebnis gerundet.
GIVING <i>operand3</i>	<p>Wenn das Schlüsselwort GIVING benutzt wird, wird <i>operand2</i> nicht geändert, und das Ergebnis wird in <i>operand3</i> ausgegeben.</p> <p>Wird ein Datenbankfeld als Ergebnisfeld verwendet, ändert sich durch die Division lediglich der programmintern verwendete Wert des Feldes; der in der Datenbank gespeicherte Wert des Feldes wird davon nicht berührt.</p> <p>Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. <i>operand2</i>, wenn keine GIVING-Klausel benutzt wird, oder <i>operand3</i>, wenn die GIVING-Klausel benutzt wird) ausgewertet.</p> <p>Informationen zur Genauigkeit der Ergebnisse siehe Abschnitt <i>Genauigkeit von Ergebnissen bei arithmetischen Operationen</i> im Leitfaden zur Programmierung.</p>
REMAINDER <i>operand4</i>	<p>Das Schlüsselwort REMAINDER bewirkt, dass der nach einer (ungerundeten) Division verbleibende Rest in <i>operand4</i> ausgegeben wird.</p> <p>Wenn GIVING und REMAINDER benutzt werden, kann keiner der vier Operanden ein Array-Bereich sein.</p> <p>Intern wird der Rest wie folgt berechnet:</p> <ol style="list-style-type: none"> 1. Der Quotient der Division von <i>operand1</i> und <i>operand2</i> wird berechnet. 2. Der Quotient wird mit <i>operand1</i> multipliziert. 3. Das Produkt dieser Multiplikation wird von <i>operand2</i> subtrahiert. 4. Das Ergebnis dieser Subtraktion wird <i>operand4</i> zugewiesen. <p>Für jeden dieser Schritte gelten die im Abschnitt <i>Genauigkeit von Ergebnissen bei arithmetischen Operationen</i> im Leitfaden zur Programmierung beschriebenen Regeln.</p>

Beispiel

```

** Example 'DIVEX1': DIVIDE
*****
DEFINE DATA LOCAL
1 #A (N7) INIT <20>
1 #B (N7)
1 #C (N3.2)
1 #D (N1)
1 #E (N1) INIT <3>
1 #F (N1)
END-DEFINE
*
DIVIDE 5 INTO #A
WRITE NOTITLE 'DIVIDE 5 INTO #A' 20X '=' #A
*
RESET INITIAL #A
DIVIDE 5 INTO #A GIVING #B
WRITE 'DIVIDE 5 INTO #A GIVING #B' 10X '=' #B
*
DIVIDE 3 INTO 3.10 GIVING #C
WRITE 'DIVIDE 3 INTO 3.10 GIVING #C' 8X '=' #C
*
DIVIDE 3 INTO 3.1 GIVING #D
WRITE 'DIVIDE 3 INTO 3.1 GIVING #D' 9X '=' #D
*
DIVIDE 2 INTO #E REMAINDER #F
WRITE 'DIVIDE 2 INTO #E REMAINDER #F' 7X '=' #E '=' #F
*
END

```

Ausgabe des Programms DIVEX1:

```

DIVIDE 5 INTO #A           #A:      4
DIVIDE 5 INTO #A GIVING #B #B:      4
DIVIDE 3 INTO 3.10 GIVING #C #C:    1.03
DIVIDE 3 INTO 3.1 GIVING #D #D:     1
DIVIDE 2 INTO #E REMAINDER #F #E: 1 #F: 1

```

55 DO/DOEND

▪ Funktion	316
▪ Einschränkungen	316
▪ Beispiel	317

```
DO statement ... DOEND
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Die Statements `DO` und `DOEND` werden im Reporting Mode verwendet, wenn mehrere Statements in Abhängigkeit von einer logischen Bedingung ausgeführt werden sollen.

- `AT BREAK`
- `AT END OF DATA`
- `AT END OF PAGE`
- `AT START OF DATA`
- `AT TOP OF PAGE`
- `BEFORE BREAK PROCESSING`
- `FIND ... IF NO RECORDS FOUND`
- `IF`
- `IF SELECTION`
- `ON ERROR`
- `READ WORK FILE ... AT END OF FILE`

Einschränkungen

- Die Statements `DO` und `DOEND` gelten nur im Reporting Mode.
- `WRITE TITLE`, `WRITE TRAILER`, und die Bedingungs-Statements, die mit `AT` beginnen (`AT BREAK`, `AT END OF DATA`, `AT END OF PAGE`, `AT START OF DATA`, `AT TOP OF PAGE`) dürfen innerhalb einer `DO/DOEND`-Konstruktion nicht verwendet werden..
- Wenn Sie innerhalb einer `DO/DOEND`-Konstruktion eine Verarbeitungsschleife initiieren, müssen Sie sie vor dem `DOEND`-Statement wieder schließen.

Beispiel

```

** Example 'DOEEX1': DO/DOEND
*****
*
EMP. FIND EMPLOYEES WITH CITY = 'MILWAUKEE'
  VEH. FIND VEHICLES WITH PERSONNEL-ID = PERSONNEL-ID
    IF NO RECORDS FOUND DO
      ESCAPE
    DOEND
    DISPLAY PERSONNEL-ID (EMP.) NAME (EMP.)
      SALARY (EMP.,1)
      MAKE (VEH.) MAINT-COST (VEH.,1)
  AT END OF DATA DO
    WRITE NOTITLE
      / 10X 'AVG SALARY:'
        T*SALARY (1) AVER(SALARY (1))
      / 10X 'AVG MAINTENANCE (ZERO VALUES EXCLUDED):'
        T*MAINT-COST (1) NAVER(MAINT-COST (1))
    DOEND
  /*
LOOP
LOOP
END

```

Ausgabe des Programms DOEEX1:

PERSONNEL ID	NAME	ANNUAL SALARY	MAKE	MAINT-COST
20021100	JONES	31000	GENERAL MOTORS	140
20027800	LAWLER	29000	GENERAL MOTORS	0
20027800	LAWLER	29000	TOYOTA	86
20030600	NORDYKE	47000	FORD	194
	AVG SALARY:	35666		
	AVG MAINTENANCE (ZERO VALUES EXCLUDED):			140

56 EJECT

▪ Funktion	320
▪ Syntax-Beschreibung	320
▪ Verarbeitung	322
▪ Beispiel	322

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [CLOSE PRINTER](#) | [DEFINE PRINTER](#) | [DISPLAY](#) | [FORMAT](#) | [NEWPAGE](#) | [PRINT](#) | [SKIP](#) | [SUSPEND IDENTICAL SUPPRESS](#) | [WRITE](#) | [WRITE TITLE](#) | [WRITE TRAILER](#)

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das EJECT-Statement kann dazu verwendet werden, einen Seitenvorschub auszulösen.

Vgl. auch Natural Profil- und Session-Parameter EJ in der *Parameter-Referenz*.

Syntax-Beschreibung

Zwei verschiedene Strukturen sind für dieses Statement möglich.

- [EJECT - Syntax 1](#)
- [EJECT - Syntax 2](#)

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

EJECT - Syntax 1

EJECT { ON OFF } [(rep)]

Syntax-Element-Beschreibung:

EJECT ON/OFF	Mit Report-Spezifikation – Online- und Batch-Verarbeitung:	
(rep)	EJECT OFF (rep)	Bewirkt, dass für den angegebenen Report kein Seitenvorschub (außer einem mit Syntax 2 des EJECT-Statements angegebenen) ausgeführt wird.
	EJECT ON (rep)	Bewirkt, dass Seitenvorschübe für den angegebenen Report ausgeführt werden.
EJECT ON/OFF	Ohne Report-Spezifikation – nur Batch-Verarbeitung:	
	EJECT ON/OFF – ohne (rep)-Notation – kann im Batch-Betrieb dazu verwendet werden, den Seitenvorschub zwischen den bei der Ausführung eines Programms erzeugten Ausgabelisten zu steuern.	

	EJECT ON	Bewirkt, dass Natural jeweils zwischen der Sourceprogramm- Auflistung, dem Ausgabe-Report und der Meldung EXECUTION COMPLETED einen Seitenvorschub ausführt. Dies ist die Voreinstellung.
	EJECT OFF	Bewirkt, dass keiner der oben genannten Seitenvorschübe ausgeführt wird. EJECT OFF gilt solange, bis es durch ein nachfolgendes EJECT ON-Statement wieder zurückgenommen wird.
(rep)	Report-Spezifikation: Mit der Notation (rep) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das EJECT-Statement auf den ersten Report (Report 0). Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern können, siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i> .	

EJECT - Syntax 2

Diese Form des EJECT-Statements kann dazu verwendet werden, einen Seitenvorschub auszulösen, ohne dass eine End-of-Page- oder Top-of-Page-Verarbeitung durchgeführt oder auf der neuen Seite eine Titel- oder Kopfzeile generiert wird.

```
EJECT [(rep)] [ [ IF ] LESS [THAN] operand1 [LINES] [LEFT] ]
                [ WHEN ]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand1	C S	N P I	ja	nein

Syntax-Element-Beschreibung:

(rep)	Report-Spezifikation: Mit der Notation (rep) kann ein bestimmter anderer Report angegeben werden, auf den sich das EJECT-Statement beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Falls nichts anderes angegeben wird, bezieht sich das EJECT-Statement auf den ersten ausgegebenen Report (Report 0).
-------	--

	Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe- Reports steuern können, siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i> .
IF LESS THAN <i>operand1</i> LINES LEFT	Ein Seitenvorschub wird nur ausgeführt, wenn die aktuelle Zeile für die Seite größer als die Seitenlänge minus <i>operand1</i> ist. <i>operand1</i> kann als numerische Konstante oder als Variable angegeben werden.

Verarbeitung

Die Ausführung eines EJECT-Statements löst keine Ausführung der mit **AT TOP OF PAGE**, **AT END OF PAGE**, **WRITE TITLE** or **WRITE TRAILER** verknüpften Statements aus. Ebenso wenig beeinflusst es die Auswertung von Systemfunktionen in einem **DISPLAY**-Statement mit **GIVE SYSTEM FUNCTIONS**-Klausel.

Das Statement EJECT bewirkt lediglich, dass eine neue physische Ausgabeseite begonnen wird. Es bewirkt außerdem, dass der Wert der Natural-Systemvariablen *LINE-COUNT wieder auf 1 gesetzt wird, hat aber keinen Einfluss auf die Natural-Systemvariable *PAGE-NUMBER.

Beispiel

```

** Example 'EJTEX1': EJECT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
FORMAT PS=15
LIMIT 9
READ EMPLOY-VIEW BY CITY
/*
  AT START OF DATA
    EJECT
    WRITE /// 20T '%' (29) /
              20T '%' /
              20T '%' 3X 'REPORT OF EMPLOYEES' 47T '%' /
              20T '%' 3X ' SORTED BY CITY ' 47T '%' /
              20T '%' /
              20T '%' 47T '%' /
              20T '%' (29) /
    EJECT
  END-START

```

```

EJECT WHEN LESS THAN 3 LINES LEFT
/*
WRITE '*' (64)
DISPLAY NOTITLE NOHDR CITY NAME JOB-TITLE 5X *LINE-COUNT
WRITE '*' (64)
END-READ
END

```

Ausgabe des Programms EJTEX1:

```

          %/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/
          %/%                                         %/%
          %/%   REPORT OF EMPLOYEES   %/%
          %/%   SORTED BY CITY       %/%
          %/%                                         %/%
          %/%                                         %/%
          %/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/%/

```

Nach dem Drücken von EINGABE:

```

*****
AIKEN             SENKO             PROGRAMMER                2
*****
AIX EN OTHE      GODEFROY           COMPTABLE                 5
*****
AJACCIO          CANALE             CONSULTANT                8
*****
ALBERTSLUND     PLOUG             KONTORASSISTENT        11
*****
ALBUQUERQUE     HAMMOND           SECRETARY                14
*****

```

Nach dem Drücken von EINGABE:

***** ALBUQUERQUE ROLLING MANAGER *****	2
***** ALBUQUERQUE FREEMAN MANAGER *****	5
***** ALBUQUERQUE LINCOLN ANALYST *****	8
***** ALFRETON GOLDBERG JUNIOR *****	11

57

END

▪ Funktion	326
▪ Syntax-Beschreibung	326
▪ Beispiele	327



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das Statement `END` dient dazu, das physische Ende eines Natural-Programms zu kennzeichnen. Auf das `END`-Statement können keine Symbole folgen.

Im Reporting Mode werden durch das `END`-Statement alle noch aktiven Verarbeitungsschleifen (die noch nicht durch ein `LOOP`-Statement beendet wurden) geschlossen.

Hinweise zur Programmausführung

Wird ein `END`-Statement in einem Hauptprogramm (einem Programm, das auf Stufe (Level) 1 ausgeführt wird) ausgeführt, so wird eine abschließende End-of-Page-Verarbeitung ausgeführt sowie für alle vom Benutzer ausgelösten Gruppenwechsel (`PERFORM BREAK PROCESSING`), die sich nicht durch Referenzierung (Statement-Label oder Sourcecode-Zeilenummer) auf eine bestimmte Verarbeitungsschleife beziehen, eine abschließende Gruppenwechsel-Verarbeitung.

Die Ausführung eines `END`-Statements in einem Subprogramm oder einem mit `FETCH RETURN` aufgerufenen Programm bewirkt lediglich, dass die Kontrolle wieder an das aufrufende Programm ohne eine endgültige Verarbeitung übergeben wird.

Syntax-Beschreibung

END	<p>Schlüsselwort:</p> <p>Das für Natural reservierte Schlüsselwort <code>END</code> dient normalerweise zum Markieren des physischen Endes eines Natural-Programms.</p>
·	<p>Punkt:</p> <p>Anstatt des für Natural reservierten Schlüsselworts <code>END</code> kann ein Punkt (.) benutzt werden. Falls Sie statt <code>END</code> einen Punkt (.) verwenden und sich in derselben Zeile noch andere Statements befinden, müssen Sie dem Punkt mindestens ein Leerzeichen voranstellen.</p>

Beispiele

Einige typische Beispiele finden Sie im Abschnitt *Beispiele für die Benutzung des DEFINE DATA-Statements*.

58

END TRANSACTION

▪ Funktion	330
▪ Einschränkung	331
▪ Syntax-Beschreibung	331
▪ Betroffene Datenbanken	331
▪ Datenbank-spezifische Anmerkungen	332
▪ Beispiele	332

END [OF] TRANSACTION [*operand1* ...]

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | FIND | GET | GET SAME | GET TRANSACTION DATA | FIND HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement `END TRANSACTION` dient dazu, das Ende einer logischen Transaktion zu markieren. Eine logische Transaktion ist die kleinste (vom Benutzer definierte) logische Arbeitseinheit, die vollständig ausgeführt werden muss, damit die logische Konsistenz der Daten auf der Datenbank gewährleistet ist.

Die erfolgreiche Ausführung eines `END TRANSACTION`-Statements bewirkt, dass alle im Verlaufe der Transaktion durchgeführten Datenänderungen physisch auf der Datenbank durchgeführt worden sind (bzw. werden) und von einem anschließenden Abbruch, sei er durch den Benutzer, Natural, die Datenbank oder das Betriebssystem herbeigeführt, nicht mehr beeinflusst werden können. Wenn das `END TRANSACTION`-Statement nicht erfolgreich ausgeführt wird, d.h. wenn die logische Transaktion nicht vollständig ausgeführt ist, werden alle im Laufe der Transaktion bereits durchgeführten Datenänderungen automatisch wieder rückgängig gemacht.

`END TRANSACTION` bewirkt außerdem, dass alle während der Transaktion im Hold-Status gehaltenen Datensätze wieder freigegeben werden.

Die Ausführung des `END TRANSACTION`-Statements kann an eine logische Bedingung geknüpft werden.

Weitere Informationen hierzu finden Sie im Kapitel *Datenbankzugriffe* im *Leitfaden zur Programmierung*.

Einschränkung

Das Statement `END TRANSACTION` kann nicht mit Entire System Server benutzt werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	N A U N P I F B D T	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Speicherung von Transaktionsdaten:</p> <p>Bei einer Transaktion auf einer Adabas-Datenbank können Sie mit diesem Statement auch transaktionsbezogene Daten speichern. Diese Daten dürfen maximal 2000 Bytes lang sein und können mit einem <code>GET TRANSACTION DATA</code>-Statement wieder gelesen werden.</p> <p>Die Transaktionsdaten werden auf die mit dem Profilparameter <code>ETDB</code> angegebene Datenbank geschrieben.</p> <p>Wenn der <code>ETDB</code>-Parameter nicht gesetzt ist, werden die Transaktionsdaten auf die mit dem Profilparameter <code>UDB</code> angegebene Datenbank geschrieben. Ausnahme: Auf Großrechnern werden die Transaktionsdaten auf die Datenbank geschrieben, auf der sich die Natural-Security-Systemdatei (<code>FSEC</code>) befindet (ist <code>FSEC</code> nicht angegeben, dann ist sie identisch mit der Natural-Systemdatei <code>FNAT</code>; ist Natural Security nicht installiert, dann werden die Transaktionsdaten auf die Datenbank geschrieben, auf der sich <code>FNAT</code> befindet).</p>
-----------------	--

Betroffene Datenbanken

Ein `END TRANSACTION`-Statement ohne Transaktionsdaten (d.h. ohne *operand1*) wird nur ausgeführt, wenn eine Datenbanktransaktion unter Kontrolle von Natural stattgefunden hat. Für welche Datenbanken das Statement ausgeführt wird, hängt davon ab, wie der Natural-Profilparameter `ET` gesetzt ist.

Bei `ET=OFF` wird das Statement nur für die von der Transaktion betroffene Datenbank ausgeführt; bei `ET=ON` wird es für alle Datenbanken ausgeführt, die seit der letzten Ausführung eines `BACKOUT TRANSACTION`- oder `END TRANSACTION`-Statements referenziert wurden.

Ein END TRANSACTION-Statement mit Transaktionsdaten (d.h. mit Angabe von *operand1*) wird immer ausgeführt, und die Transaktionsdaten werden wie unten beschrieben auf einer bestimmten Datenbank gespeichert. Für welche Datenbanken das Statement außerdem ausgeführt wird, hängt vom ET-Parameter (siehe oben) ab.

Datenbank-spezifische Anmerkungen

SQL Databases	Da die meisten SQL-Datenbanken bei Beendigung einer logischen Arbeitseinheit alle Cursor schließen, darf ein END TRANSACTION-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muss nach einer solchen platziert werden.
XML Databases	Ein END TRANSACTION-Statement darf nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muss nach einer solchen platziert werden.

Beispiele

- [Beispiel 1 — END TRANSACTION-Statement](#)
- [Beispiel 2 — END TRANSACTION-Statement mit ET-Daten](#)

Beispiel 1 — END TRANSACTION-Statement

```

** Example 'ETREX1': END TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'BOSTON'
  ASSIGN COUNTRY = 'USA'
  UPDATE
  END TRANSACTION
/*
AT END OF DATA
  WRITE NOTITLE *NUMBER 'RECORDS UPDATED'
END-ENDDATA
/*
END-FIND
END
    
```


Ausgabe des Programms ETREX1:

7 RECORDS UPDATED

Beispiel 2 — END TRANSACTION-Statement mit ET-Daten

```

** Example 'ETREX2': END TRANSACTION (with ET data)
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
*
REPEAT
  INPUT 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  END-IF
  /*
  FIND EMPLOY-VIEW PERSONNEL-ID = #PERS-NR
  INPUT (AD=M)  NAME / FIRST-NAME / CITY
  UPDATE
  END TRANSACTION #PERS-NR
  END-FIND
  /*
END-REPEAT
END

```

Ausgabe des Programms ETREX2:

ENTER PERSONNEL NUMBER TO BE UPDATED: 20027800

Nach Änderung und Bestätigung der Personalnummer:

END TRANSACTION

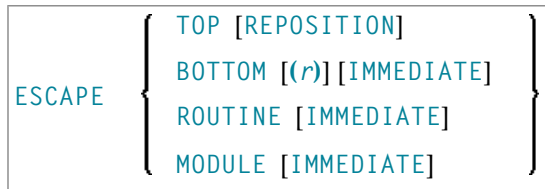
NAME LAWLER
FIRST-NAME SUNNY
CITY MILWAUKEE

59

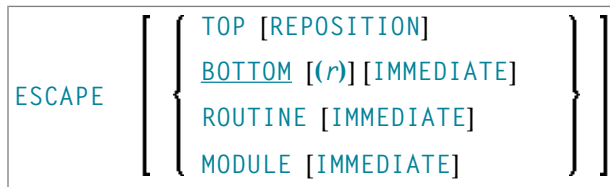
ESCAPE

- Funktion 336
- Syntax-Beschreibung 337
- Beispiel 338

Structured Mode-Syntax



Reporting Mode-Syntax



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements:

- FOR | REPEAT | PROCESS PAGE MODAL
- CALL | CALL FILE | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | FETCH | PERFORM

Gehört zur Funktionsgruppe:

- *Schleifenverarbeitung*
- *Aufruf von Programmen und Unterprogrammen*

Funktion

Das Statement `ESCAPE` dient dazu, den linearen Ablauf der Ausführung einer Verarbeitungsschleife oder eines Unterprogramms zu unterbrechen.

Mit den Schlüsselwörtern `TOP`, `BOTTOM` und `ROUTINE` geben Sie an, wo die Verarbeitung nach dem `ESCAPE`-Statement fortgesetzt werden soll.

Ein `ESCAPE TOP`- bzw. `ESCAPE BOTTOM`-Statement bezieht sich immer auf die innerste gerade aktive Verarbeitungsschleife. Das `ESCAPE`-Statement muss nicht unbedingt innerhalb der Schleife stehen.

Befindet sich das `ESCAPE TOP`- bzw. `ESCAPE BOTTOM`-Statement in einem Unterprogramm (Subroutine, Subprogramm oder mit `FETCH RETURN` aufgerufenes Programm), werden die innerhalb der Verarbeitungsschleife aufgerufenen Unterprogramme automatisch beendet.

Anmerkungen

In einer Verarbeitungsschleife können mehrere `ESCAPE`-Statements enthalten sein.

Die Ausführung eines `ESCAPE`-Statements kann an eine logische Bedingung geknüpft werden. Befindet sich das `ESCAPE`-Statement in einem `AT END OF DATA`-, `AT BREAK`- oder `AT END OF PAGE`-Statement-Block, so wird die Verarbeitung des betreffenden Blocks abgebrochen und die `ESCAPE`-Verarbeitung wie angegeben fortgesetzt.

Wird das `ESCAPE`-Statement während einer `IF NO RECORDS FOUND`-Bedingung ausgeführt, werden keine schleifenabschließenden Verarbeitungen durchgeführt (entspricht `ESCAPE IMMEDIATE`).

Syntax-Beschreibung

ESCAPE TOP	TOP bedeutet, dass die Verarbeitung am Anfang der Verarbeitungsschleife fortgesetzt werden soll, d.h. die Schleife wird erneut von Anfang an durchlaufen.
REPOSITION	Wenn ein <code>ESCAPE TOP REPOSITION</code> -Statement ausgeführt wird, fährt Natural sofort mit der Verarbeitung am Anfang der aktiven <code>READ</code> -Schleife fort und benutzt dabei den aktuellen Wert der Suchvariable als neuen Startwert. Gleichzeitig setzt <code>ESCAPE TOP REPOSITION</code> die Systemvariable <code>*COUNTER</code> auf Null (0). <code>ESCAPE TOP REPOSITION</code> kann innerhalb einer <code>READ</code> -Statementschleife angegeben werden, die auf Adabas-Datenbanken zugreift. Das betreffende <code>READ</code> -Statement muss die Option <code>WITH REPOSITION</code> enthalten.
ESCAPE BOTTOM	BOTTOM bedeutet, dass die Verarbeitung mit dem ersten Statement nach der Verarbeitungsschleife fortgesetzt werden soll. Die Schleife wird beendet, und schleifenabschließende Verarbeitungen (abschließendes <code>BREAK</code> und <code>END DATA</code>) werden für alle zu beendenden Schleifen ausgeführt. Im Reporting Mode ist <code>ESCAPE BOTTOM</code> die Voreinstellung.
(r)	Notation (<i>r</i>): Wenn auf <code>BOTTOM</code> ein Label oder eine Referenznummer folgt, wird die Verarbeitung mit dem ersten Statement nach der Verarbeitungsschleife fortgesetzt, das durch das Label oder die Referenznummer identifiziert wird. Ein Label oder eine Referenznummer kann jedoch nur dann angegeben werden, wenn das <code>ESCAPE BOTTOM</code> -Statement innerhalb der referenzierten Verarbeitungsschleife steht.
IMMEDIATE	Wenn Sie das Schlüsselwort <code>IMMEDIATE</code> angeben, werden keine abschließenden schleifenbeendenden Verarbeitungen durchgeführt.

ESCAPE ROUTINE	<p>Diese Option bewirkt, dass das aktive Natural-Unterprogramm, das entweder über PERFORM, CALLNAT, FETCH RETURN oder als Hauptprogramm aufgerufen wurde, die Kontrolle abgibt.</p> <p>Im Falle einer Subroutine wird die Verarbeitung mit dem ersten Statement fortgesetzt, das auf das Statement folgt, mit dem die Subroutine aufgerufen wurde.</p> <p>Im Falle eines Hauptprogramms gelangt Natural in den Kommando-Modus. Alle aktiven Schleifen innerhalb des Unterprogramms werden beendet und schleifenabschließende Verarbeitungen (BREAK und END OF DATA) sowie vom Benutzer bestimmte Gruppenwechsel-Verarbeitungen (PERFORM BREAK) durchgeführt, und zwar für alle betroffenen Verarbeitungsschleifen. Steht das ESCAPE ROUTINE-Statement in einem auf Stufe (Level) 1 ausgeführten Hauptprogramm, wird außerdem eine abschließende End-of-Page-Verarbeitung durchgeführt.</p>
ESCAPE MODULE	<p>Diese Option bewirkt, dass die gesamte aktive Programmebene, einschließlich aller internen Subroutinen, die Kontrolle abgibt. Die Kontrolle wird dann an das Objekt der vorherigen Programmebene zurückgegeben.</p> <p>Wenn ESCAPE MODULE in einer Hierarchie interner Subroutinen benutzt wird, ermöglicht es diese Option, alle auf dieser Ebene laufenden Unterprogramme sofort zu verlassen.</p> <p>Wenn keine interne Subroutine aktiv ist, führt ESCAPE MODULE zum gleichen Ergebnis wie ESCAPE ROUTINE.</p> <p>ESCAPE MODULE ist nur bei internen Subroutinen von Bedeutung. Bei externen Subroutinen, Subprogrammen und aufgerufenen Programmen hat diese Option denselben Effekt wie ESCAPE ROUTINE.</p> <p>Wie bei ESCAPE ROUTINE wird eine schleifenabschließende Verarbeitung ausgeführt. Geben Sie aber das Schlüsselwort IMMEDIATE an, wird keine schleifenbeendende Verarbeitung ausgeführt.</p>

Beispiel

```

** Example 'ESCEX1': ESCAPE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 FIRST-NAME
  2 NAME
  2 AREA-CODE
  2 PHONE
*
1 #CITY (A20) INIT <' '>
1 #CNTL (A1) INIT <' '>
END-DEFINE
*

```

```

REPEAT
  INPUT 'ENTER VALUE FOR CITY: ' #CITY
    / 'OR '.' TO TERMINATE '
  IF #CITY = '.'
    ESCAPE BOTTOM
  END-IF
/*
  FND. FIND EMPLOY-VIEW WITH CITY = #CITY
  /*
  IF NO RECORDS FOUND
    WRITE 'NO RECORDS FOUND'
    ESCAPE BOTTOM (FND.)
  END-NOREC
  AT START OF DATA
    INPUT (AD=0) 'RECORDS FOUND:' *NUMBER //
      'ENTER 'D' TO DISPLAY RECORDS' #CNTL (AD=M)
    IF #CNTL NE 'D'
      ESCAPE BOTTOM (FND.)
    END-IF
  END-START
  /*
  DISPLAY NOTITLE NAME FIRST-NAME PHONE
  END-FIND
END-REPEAT

```

Ausgabe des Programms ESCEX1:

```

ENTER VALUE FOR CITY:  PARIS
(OR '.' TO TERMINATE)

```

Nach Eingabe und Bestätigung des Namens der Stadt:

```

RECORDS FOUND:          26
ENTER 'D' TO DISPLAY RECORDS D

```

Ergebnis nach Eingabe und Bestätigung von D:

NAME	FIRST-NAME	TELEPHONE
MAIZIERE	ELISABETH	46758304
MARX	JEAN-MARIE	40738871
REIGNARD	JACQUELINE	48472153
RENAUD	MICHEL	46055008
REMOUE	GERMAINE	36929371
LAVENDA	SALOMON	40155905
BROUSSE	GUY	37502323
GIORDA	LOUIS	37497316

ESCAPE

SIECA	FRANCOIS	40487413
CENSIER	BERNARD	38070268
DUC	JEAN-PAUL	38065261
CAHN	RAYMOND	43723961
MAZUY	ROBERT	44286899
FAURIE	HENRI	44341159
VALLY	ALAIN	47326249
BRETON	JEAN-MARIE	48467146
GIGLEUX	JACQUES	40477399
KORAB-BRZOZOWSKI	BOGDAN	45288048
XOLIN	CHRISTIAN	46060015
LEGRIS	ROGER	39341509
VVVV		

60 EXAMINE

▪ Syntax 1 — EXAMINE	342
▪ Syntax 2 — EXAMINE TRANSLATE	351
▪ Syntax 3 — EXAMINE für Unicode-Grapheme	353
▪ Beispiele	356

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [MOVE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Syntax 1 — EXAMINE

```
EXAMINE [DIRECTION-clause]  
  
    [FULL [VALUE [OF]]] { operand1  
                        SUBSTRING  
                        (operand1,operand2,operand3) }  
  
    [POSITION-clause]  
    [FOR] [FULL [VALUE [OF]]] [PATTERN] operand4  
    [DELIMITERS-option]  
    {[DELETE-REPLACE-clause] [GIVING-clause]}
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax-Beschreibung — Syntax 1

Das Statement `EXAMINE` dient dazu, den Inhalt eines alphanumerischen oder binären Feldes (oder eines Bereiches von Feldern innerhalb eines Arrays) nach einer bestimmten Zeichenkette abzusuchen und um

- zu zählen, wie oft eine bestimmte Zeichenkette vorkommt;
- die Byte-Position zurückzugeben, an der eine gesuchte Zeichenkette zuerst erscheint;
- die signifikante Länge des Inhalts eines Feldes zurückzugeben, d.h. die Feldlänge ohne nachfolgende Leerzeichen;
- die Ausprägungsnummer (Indizes) eines Array-Feldes zurückzugeben, wo eine Zeichenkette zuerst gefunden wurde;
- eine Zeichenkette auszutauschen;
- eine Zeichenkette zu löschen.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C*	S	A		A	U				B				ja	nein
<i>operand2</i>	C	S					N	P	I	B*				ja	nein
<i>operand3</i>	C	S					N	P	I	B*				ja	nein
<i>operand4</i>	C	S	A		A	U				B				ja	nein

* *operand1* darf nur eine Konstante sein, wenn Sie die GIVING-Klausel verwenden, aber nicht, wenn Sie die DELETE-REPLACE-Klausel verwenden.

* Format B von *operand2* und *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

<i>DIRECTION-clause</i>	<p>Suchrichtung:</p> <p>Mit dieser Klausel legen Sie die Suchrichtung fest. Weitere Informationen siehe <i>DIRECTION-Klausel</i> weiter unten.</p>
<i>operand1</i>	<p>Zu untersuchendes Feld:</p> <p><i>operand1</i> ist das Feld, dessen Inhalt untersucht werden soll.</p> <p>Ist <i>operand1</i> eine dynamische Variable, kann deren Länge über eine REPLACE-Operation auf einen höheren oder niedrigeren Wert gesetzt werden; durch eine DELETE-Operation kann deren Länge auf "0" gesetzt werden. Die aktuelle Länge einer dynamischen Variablen kann über die Systemvariable *LENGTH ermittelt werden.</p> <p>Sie finden allgemeine Informationen über dynamische Variablen im Abschnitt <i>Große und dynamische Variablen/Felder</i>.</p>
<i>POSITION-clause</i>	<p>POSITION-Klausel:</p> <p>Mit dieser Klausel können Sie für die Untersuchung eine Start- und Endeposition innerhalb von <i>operand1</i> angeben (oder den Substring von <i>operand1</i>). Weitere Informationen siehe <i>POSITION-Klausel</i> weiter unten.</p>
<i>operand4</i>	<p>Suchwert:</p> <p><i>operand4</i> ist der für die Untersuchung zu verwendende Wert.</p> <p>Weitere Informationen zu <i>operand4</i> und <i>operand6</i> siehe <i>operand6</i>, der in der unten beschriebenen <i>DELETE REPLACE-Klausel</i> benutzt wird.</p>
FULL	<p>FULL-Option:</p>

	<p>Wenn Sie für einen Operanden FULL angeben, so wird der gesamte Wert, einschließlich nachfolgender Leerstellen, verarbeitet; ohne FULL werden dem Wert nachfolgende Leerstellen bei der Verarbeitung ignoriert.</p>
SUBSTRING	<p>SUBSTRING-Option:</p> <p>Normalerweise wird der ganze Inhalt des Feldes untersucht, und zwar vom Anfang des Feldes bis zum Ende bzw. bis zum letzten signifikanten Zeichen.</p> <p>Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes zu untersuchen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (<i>operand1</i>) zunächst die erste Stelle (<i>operand2</i>) und dann die Länge (<i>operand3</i>) des Feldteils, der untersucht werden soll, an.</p> <p>Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A zu untersuchen, geben Sie folgendes an:</p> <pre>EXAMINE SUBSTRING(#A,5,8).</pre> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn Sie <i>operand2</i> weglassen, wird ab Anfang des Feldes untersucht. 2. Wenn Sie <i>operand3</i> weglassen, wird ab der angegebenen Stelle (<i>operand2</i>) bis zum Ende des Feldes untersucht. 3. Wenn SUBSTRING in Verbindung mit einer dynamischen Variable benutzt wird, verhält sich das Feld wie eine Variable fester Länge, d.h. die Länge (*LENGTH) ändert sich nicht als Ergebnis der EXAMINE-Operation, ungeachtet der Tatsache, ob eine DELETE- oder REPLACE-Operation ausgeführt wurde oder nicht.
PATTERN	<p>PATTERN-Option:</p> <p>Wenn Sie das Feld nach einem Wert absuchen möchten, der Variablen enthält, d.h. Platzhalter für Stellen, die bei der Suche nicht berücksichtigt werden sollen, verwenden Sie die Option PATTERN. <i>operand4</i> kann dann die folgenden Platzhalter für nicht zu untersuchende Stellen enthalten:</p> <ul style="list-style-type: none"> ■ Ein Punkt (.), Fragezeichen (?) oder Unterstrich (_) steht für eine einzelne Stelle, die nicht untersucht werden soll. ■ Ein Stern (*) oder Prozentzeichen (%) steht für eine beliebige Anzahl von Stellen, die nicht untersucht werden sollen. <p>Beispiel:</p> <p>Mit PATTERN 'NAT*AL' könnten Sie ein Feld nach jedem Wert, in dem NAT und AL vorkommt, absuchen, ganz gleich, welche und wieviele andere Zeichen zwischen NAT und AL stehen (dies würde z.B. auf die Werte NATURAL und NATIONAL zutreffen, aber auch auf NATAL).</p>

<i>DELIMITERS-option</i>	DELIMITERS-Option: Diese Option wird zum Suchen eines Wertes benutzt, der Delimiter darstellt. Einzelheiten, siehe <i>DELIMITERS-Option</i> weiter unten.
<i>DELETE-REPLACE-clause</i>	DELETE REPLACE-Klausel: Die DELETE-Option dieser Klausel wird zum Löschen jedes Suchwertes (<i>operand4</i>) benutzt, der in <i>operand1</i> gefunden wird. Die REPLACE-Option wird zum Austauschen jedes in <i>operand1</i> gefundenen Suchwertes (<i>operand4</i>) durch den in <i>operand6</i> angegebenen Wert benutzt. Siehe <i>DELETE REPLACE-Klausel</i> weiter unten.
<i>GIVING-clause</i>	GIVING-Klausel: Siehe <i>GIVING-Klausel</i> weiter unten.

DIRECTION-Klausel

Diese Klausel bestimmt die Suchrichtung.

DIRECTION	{ FORWARD BACKWARD <i>operand8</i> }
-----------	--

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand8</i>	C S	A1	ja	nein

Syntax-Element-Beschreibung:

FORWARD	Wenn Sie FORWARD angeben, wird der Feldinhalt von links nach rechts untersucht.
BACKWARD	Wenn Sie BACKWARD angeben, wird der Feldinhalt von rechts nach links untersucht.
<i>operand8</i>	Wenn Sie <i>operand8</i> angeben, wird die Suchrichtung durch den Inhalt von <i>operand8</i> bestimmt. <i>operand8</i> muss mit Format/Länge A1 definiert werden. Wenn <i>operand8</i> ein "F" enthält, dann ist die Suchrichtung FORWARD; wenn <i>operand8</i> ein "B" enthält, dann ist die Suchrichtung BACKWARD. Alle anderen Werte sind ungültig und werden zurückgewiesen: wenn <i>operand8</i> eine Konstante ist, wird der Wert zur Kompilierzeit zurückgewiesen; wenn <i>operand8</i> eine Variable ist, wird der Wert zur Laufzeit zurückgewiesen.



Anmerkung: Wenn die DIRECTION-Klausel nicht angegeben ist, wird die Standardrichtung FORWARD benutzt.

POSITION-Klausel

Mit dieser Klausel können Sie für die Untersuchung eine Start- und Endeposition innerhalb von *operand1* angeben (oder den Substring von *operand1*).

```
[[[STARTING] FROM [POSITION] operand9] [ { ENDING AT } [POSITION] operand10 ] ]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate										Referenzierung erlaubt	Dynam. Definition				
<i>operand9</i>	C	S															ja	nein
<i>operand10</i>	C	S															ja	nein

Syntax-Element-Beschreibung:

FROM <i>operand9</i>	<i>operand9</i> definiert die Startposition für die Untersuchung.
ENDING AT / THROUGH <i>operand10</i>	<i>operand10</i> definiert die Endeposition für die Untersuchung.


Startposition (*operand9*) und Endeposition (*operand10*) sind relativ zu *operand1* oder dem Substring von *operand1* und werden beide verarbeitet.

Die Suche beginnt an der Startposition und endet an der Endeposition.

Wenn Start- und/oder Endeposition nicht angegeben sind, gelten die Standardwerte für die Position. Der Wert wird durch die Suchrichtung bestimmt:

Richtung	Standardstartposition	Standardendeposition
FORWARD	1 (erstes Zeichen)	Länge von <i>operand1</i> (letztes Zeichen)
BACKWARD	Länge von <i>operand1</i> (letztes Zeichen)	1 (erstes Zeichen)

Mit dieser Lösung ist `EXAMINE BACKWARD ...` identisch mit `EXAMINE BACKWARD ... FROM *LENGTH(...) THRU 1` und funktioniert wie erwartet.

 **Anmerkung:** Eine Suche wird nicht durchgeführt wenn die Suchrichtung FORWARD ist und die Startposition größer als die Endeposition ist, oder wenn die Suchrichtung BACKWARD ist und die Startposition kleiner als die Endeposition ist.

DELIMITERS-Option

```

{
  ABSOLUTE
  [WITH DELIMITERS]
  [WITH DELIMITERS] operand5
}
    
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand5</i>	C S	A B	ja	nein

Syntax-Element-Beschreibung:

ABSOLUTE	<p>Absolute Suche:</p> <p>Standardmäßig gilt die Option ABSOLUTE; d.h. die zu suchende Zeichenkette wird auch gefunden, wenn sie von anderen Zeichen umgeben und Teil einer längeren Zeichenkette ist.</p>
WITH DELIMITERS	<p>Suchwert mit beliebigen Begrenzungszeichen:</p> <p>Mit WITH DELIMITERS wird ein Wert gesucht, dem je ein Leerzeichen oder irgendein anderes Zeichen, das weder ein Buchstabe noch eine Ziffer ist, vor- und nachgestellt ist.</p>
WITH DELIMITERS <i>operand5</i>	<p>Suchwert mit bestimmten Begrenzungszeichen:</p> <p>Mit WITH DELIMITERS <i>operand5</i> wird ein Wert gesucht, der von dem/den in <i>operand5</i> angegebenen Zeichen eingegrenzt ist.</p>

DELETE REPLACE-Klausel

```

[AND] {
  DELETE [FIRST]
  REPLACE [FIRST] [WITH] [FULL [VALUE [OF]]] operand6
}
    
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand6</i>	C S A	A U B	ja	nein

Syntax-Element-Beschreibung:

DELETE	<p>Lösch-Option:</p> <p>Dient zum Löschen der ersten (oder aller) Ausprägung(en) des Suchwertes (<i>operand4</i>) im Inhalt von <i>operand1</i>.</p>
REPLACE	<p>Austausch-Option:</p> <p>Wird benutzt zum Austauschen der ersten (oder aller) Ausprägung(en) des Suchwertes (<i>operand4</i>) in <i>operand1</i> durch den in <i>operand6</i> angegebenen Austauschwert.</p>
FIRST	<p>Löschen/Austauschen des ersten identischen Werts:</p> <p>Wenn Sie das Schlüsselwort FIRST angeben, wird nur der erste identische Wert gelöscht/ausgetauscht.</p>



Anmerkungen:

1. Wenn die REPLACE-Operation zur Generierung von mehr Zeichen führt als in *operand1* passen, erhalten Sie eine Fehlermeldung.
2. Wenn *operand1* eine dynamische Variable ist, kann eine REPLACE-Operation dazu führen, dass seine Länge vergrößert oder verkleinert wird; eine DELETE-Operation kann dazu führen, dass seine Länge auf Null (0) gesetzt wird. Die aktuelle Länge einer dynamischen Variablen kann mittels der Systemvariable *LENGTH ermittelt werden. Allgemeine Informationen zu dynamischen Variablen siehe *Dynamische Variablen benutzen*.
3. Falls ein Laufzeitfehler auftritt, bleibt das geprüfte Feld unverändert.

Suchen und Ersetzen mit mehreren Werten

Der Suchwert (*operand4*) und der Ersetzungswert (*operand6*) können auch als Array-Felder definiert werden. Dadurch ist es mit nur einem EXAMINE-Statement möglich, mehrere unterschiedliche Muster in dem geprüften Feld (*operand1*) zu ersetzen. Der Such- und der Ersetzungsoperand brauchen nicht die gleiche Anzahl an Ausprägungen zu haben. Es muss lediglich die Übertragungskompatibilität zwischen diesen Feldern gewährleistet sein, d.h. *operand4:=operand6* muss eine gültige Operation sein; siehe auch *Zuweisungen bei Arrays im Leitfaden zur Programmierung*.

Die Operationslogik für die Suche mit mehreren Werten arbeitet wie folgt:

- Das zu prüfenden Feld (*operand6*) wird nur einmal durchlaufen, entweder von links nach rechts für Richtung FORWARD oder von rechts nach links für Richtung BACKWARD.

- Die Werte im Such-Array (*operand4*) werden, beginnend mit der ersten Position, auf Übereinstimmung geprüft, und zwar einer nach dem anderen, wobei mit der Array-Ausprägung mit dem niedrigsten Index begonnen wird.
- Wird kein Suchwert gefunden, wird der Vergleich auf der nächsten Feldposition fortgesetzt.
- Wenn eines der gesuchten Muster in einem geprüften Feld (*operand1*) gefunden wird, dann wird es durch den Wert des Ersetzungs-Arrays (*operand6*) ersetzt, das das übereinstimmende Muster in *operand4* belegt, wenn eine Operation *operand4:=operand6* ausgeführt würde.
- Nachdem die Ersetzung eines Musters erfolgt ist, wird der Vergleichsvorgang unmittelbar nach dem eingefügten Wert mit der ersten Ausprägung für das Such-Array fortgesetzt. Das bedeutet, dass ein schon einmal ersetztes Muster übersprungen wird und kein zweites Mal mehr ersetzt werden kann.

Beispiel 1:

Dieses Beispiel zeigt die Ersetzung des Kleiner-als-Zeichens (<), des Größer-als-Zeichens (>) und des Zeichens für das Kaufmännische Und (&) durch die entsprechenden HTML-Zeichen '<', '>', und '&'.

```
DEFINE DATA LOCAL
1 #HTML (A/1:3) DYNAMIC INIT <'&lt;', '&gt;', '&amp;'>
1 #TAB (A/1:3) DYNAMIC INIT <'<', '>', '&'>
1 #DOC(A) DYNAMIC /* document to be replaced
END-DEFINE
#DOC := 'a&lt;&lt;b&amp;b&gt;c&gt;'
WRITE #DOC (AL=30) 'before'
/* Replace #DOC using #HTML to #TAB (n:1 replacement)
EXAMINE #DOC FOR #HTML(*) REPLACE #TAB(*)
/* '&lt;' is replaced by '<' (4:1 replacement)
/* '&gt;' is replaced by '>' (4:1 replacement)
/* '&amp;' is replaced by '&' (5:1 replacement)
WRITE #DOC (AL=30) 'after'
END
```

Beispiel 2:

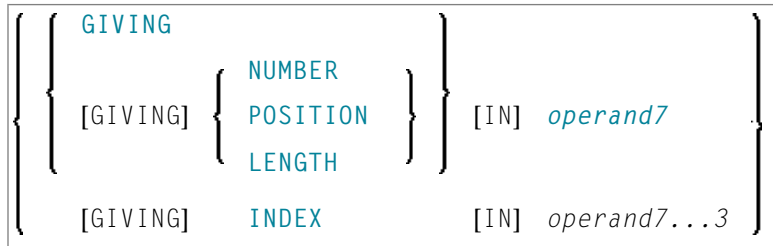
Dieses Beispiel zeigt die Ersetzung des Musters 'AA', 'Aa' und 'aA' durch '++', des Musters 'BB', 'Bb' und 'bB' durch '--' und des Musters 'CC', 'Cc' und 'cC' durch '**'.

```
DEFINE DATA LOCAL
1 #SV (A2/1:3,1:3) INIT (1,V) <'AA', 'BB', 'CC'>
(3,V) <'Aa', 'Bb', 'Cc'>
(2,V) <'aA', 'bB', 'cC'>
1 #RV (A2/1:3) INIT <'++', '--', '**'>
1 #STRING (A20) INIT <'AAABbbbbBCCCcccCaaa'>
END-DEFINE
DISPLAY #STRING /* shows 'AAABbbbbBCCCcccCaaa'
EXAMINE #STRING FOR #SV(*,*)
```

```

AND REPLACE WITH #RV(*)
DISPLAY #STRING /* shows '++A--bb--****c**aa++'
END
    
```

GIVING-Klausel



Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand7</i>	S	N P I	ja	ja

Syntax-Element-Beschreibung:

GIVING	Wenn nur das Schlüsselwort GIVING angegeben wird, entspricht dies GIVING NUMBER (Voreinstellung).
NUMBER	Wird benutzt, um die Zahl zu erhalten, wie oft der zu suchende Wert (<i>operand4</i>) in dem Feld (<i>operand1</i>) gefunden wird, dessen Inhalt überprüft werden soll.
POSITION	Mit GIVING POSITION erhalten Sie die Byte-Position, die der erste gefundene Wert (<i>operand4</i>) innerhalb von <i>operand1</i> (bzw. des Substrings von <i>operand1</i>) innehat.
LENGTH	Mit GIVING LENGTH erhalten Sie die Länge von <i>operand1</i> (bzw. des Substrings von <i>operand1</i>), nachdem alle DELETE- bzw. REPLACE-Operationen abgeschlossen sind. Nachfolgende Leerzeichen werden ignoriert.
<i>operand7</i>	Die Anzahl der Ausprägungen des Suchwertes. Wenn auch die Option REPLACE FIRST oder DELETE FIRST benutzt wird, ist die Zahl nicht größer als 1.
INDEX <i>operand7...3</i>	Siehe GIVING INDEX-Option weiter unten.

GIVING INDEX-Option

```
[GIVING] INDEX [IN] operand7 ... 3
```

Diese Option steht nur zur Verfügung, wenn das zu überprüfende zugrundeliegende Feld ein Array-Feld ist.

Syntax-Element-Beschreibung:

INDEX	Mit GIVING INDEX erhalten Sie die Nummer der Ausprägung (Index) von <i>operand1</i> , in der der erste gefundene Wert (<i>operand4</i>) enthalten ist.
<i>operand7...3</i>	<i>operand7</i> muss genauso oft angegeben werden wie <i>operand1</i> Dimensionen hat (höchstens dreimal). <i>operand7</i> enthält Null, wenn der gesuchte Wert in keiner der Ausprägungen enthalten ist.



Anmerkung: Falls der Indexbereich von *operand1* die Ausprägung Null enthält (z.B. 0:5), ist der Wert 0 in *operand7* zweideutig. In diesem Falle sollte eine zusätzliche GIVING NUMBER-Klausel verwendet werden, um festzustellen, ob der gesuchte Wert tatsächlich vorkommt oder nicht.

Syntax 2 — EXAMINE TRANSLATE

```
EXAMINE { operand1
           SUBSTRING (operand1,operand2,operand3) } [AND]
           TRANSLATE { INTO { UPPER
                          LOWER } [CASE]
                     USING [INVERTED] operand4 }
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax-Beschreibung — Syntax 2

Das Statement EXAMINE TRANSLATE dient dazu, die in einem Feld enthaltenen Zeichen in Groß- oder Kleinschreibung oder in andere Zeichen umzusetzen.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>		S	A	A B	ja	nein
<i>operand2</i>	C	S		N P I B*	ja	nein
<i>operand3</i>	C	S		N P I B*	ja	nein
<i>operand4</i>		S	A	A B	ja	nein

*Format B von *operand2* und *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

EXAMINE <i>operand1</i>	<p>Umsetzung des kompletten Feldinhalts:</p> <p><i>operand1</i> ist das Feld, dessen Inhalt umgesetzt werden soll.</p>
<p>EXAMINE SUBSTRING</p> <p><i>operand1 operand2</i></p> <p><i>operand3</i></p>	<p>Umsetzung von Teilen des Feldinhalts:</p> <p>Normalerweise wird der Inhalt des gesamten Feldes umgesetzt.</p> <p>Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes umzusetzen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (<i>operand1</i>) zunächst die erste Stelle (<i>operand2</i>) und dann die Länge (<i>operand3</i>) des Feldteils, der umgesetzt werden soll, an.</p> <p>Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A umzusetzen, geben Sie folgendes an:</p> <pre>EXAMINE SUBSTRING(#A,5,8) AND TRANSLATE ...</pre> <p>Anmerkung: Wenn Sie <i>operand2</i> weglassen, wird ab Anfang des Feldes umgesetzt. Wenn Sie <i>operand3</i> weglassen, wird ab der ersten Stelle bis zum Ende des Feldes umgesetzt.</p>
TRANSLATE INTO UPPER CASE	<p>Umsetzung in Großbuchstaben:</p> <p>Der Inhalt von <i>operand1</i> wird in Großbuchstaben umgesetzt.</p>
TRANSLATE INTO LOWER CASE	<p>Umsetzung in Kleinbuchstaben:</p> <p>Der Inhalt von <i>operand1</i> wird in Kleinbuchstaben umgesetzt.</p>
TRANSLATE USING	<p>Zu benutzende Umsetzungstabelle:</p> <p><i>operand4</i> ist die Umsetzungstabelle, die für die Zeichenumsetzung verwendet werden soll. Die Tabelle muss Format/Länge A2 oder B2 haben.</p>

	Anmerkung: Falls für ein umzusetzendes Zeichen in der Umsetzungstabelle mehr als eine Umsetzung definiert ist, gilt die jeweils letzte Umsetzung.
INVERTED	Wenn Sie das Schlüsselwort <code>INVERTED</code> angeben, wird die Umsetzungstabelle (<i>operand4</i>) in umgekehrter Richtung verwendet, d.h. die Umsetzungsrichtung wird umgekehrt.

Syntax 3 — EXAMINE für Unicode-Grapheme

<pre> EXAMINE [FULL [VALUE [OF]]] [<i>POSITION-clause</i>] [FOR] [CHARPOSITION <i>operand4</i>] [CHARLENGTH <i>operand5</i>] [GIVING] POSITION IN <i>operand6</i> [[GIVING] LENGTH IN <i>operand7</i>] </pre>	<pre> <i>operand1</i> SUBSTRING (<i>operand1, operand2, operand3</i>) </pre>
---	--

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Syntax-Beschreibung - Syntax 3

Unter einem Graphem versteht ein Benutzer normalerweise ein Zeichen. In den meisten Fällen ist eine UTF-16 Code-Einheit (= U-Formatzeichen) ein Graphem, allerdings kann ein Graphem auch aus mehreren Code-Einheiten bestehen. Beispiele sind: eine Folge von einem Basiszeichen gefolgt von Kombinationszeichen oder einem Ersatz-Paar. Weitere Informationen zu Graphemen und anderen Unicode-Begriffen entnehmen Sie dem Dokument *The Unicode Standard* unter <http://www.unicode.org/>.

Das Statement `EXAMINE` für U-Format-Operanden spricht im Allgemeinen Code-Einheiten an. Allerdings ist es bei `CHARPOSITION`- und `CHARLENGTH`-Klauseln möglich, die Startposition und Länge (als Code-Einheiten) einer Graphem-Sequenz zu erhalten. Die zurückgegeben Code-Einheitswerte können dann in anderen Statements/Klauseln benutzt werden, für die Code-Einheitsoperanden erforderlich sind (zum Beispiel in einer `SUBSTRING`-Klausel).

Weitere Informationen zur Syntax des `EXAMINE`-Statements, siehe auch *Unicode and Code Page Support in Natural Programming Language*, Abschnitt *Statements, EXAMINE*.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition								
<i>operand1</i>	C	S	A			U																		ja	nein
<i>operand2</i>	C	S				N	P	I	B*															ja	nein
<i>operand3</i>	C	S				N	P	I	B*															ja	nein
<i>operand4</i>	C	S	A			N	P	I																ja	nein
<i>operand5</i>	C	S	A			N	P	I																ja	nein
<i>operand6</i>	C	S				N	P	I																ja	nein
<i>operand7</i>	C	S				N	P	I																ja	nein

* Format B von *operand2* und *operand3* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

FULL	<p>FULL-Option:</p> <p>Wenn FULL für einen Operanden angegeben wird, wird der gesamte Wert, einschließlich der nachfolgenden Leerzeichen abgearbeitet. Wenn FULL nicht angegeben wird, werden nach folgende Leerzeichen im Operanden ignoriert.</p>
SUBSTRING	<p>SUBSTRING-Option:</p> <p>Normalerweise wird der ganze Inhalt des Feldes untersucht, und zwar vom Anfang des Feldes bis zum Ende bzw. bis zum letzten signifikanten Zeichen.</p> <p>Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes zu untersuchen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (<i>operand1</i>) zunächst die erste Stelle (<i>operand2</i>) und dann die Länge (<i>operand3</i>) des Feldteils an, der untersucht werden soll. <i>operand2</i> und <i>operand3</i> werden als Code-Einheiten angegeben.</p> <p>Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A zu untersuchen, geben Sie folgendes an:</p> <pre style="background-color: #e0e0e0;">EXAMINE SUBSTRING (#A,5,8)</pre> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn Sie <i>operand2</i> weglassen, wird ab Anfang des Feldes (Position 1) untersucht. 2. Wenn Sie <i>operand3</i> weglassen, wird ab der angegebenen Stelle bis zum Ende des Feldes untersucht. 3. Wenn SUBSTRING in Verbindung mit einer dynamischen Variable benutzt wird, verhält sich das Feld wie eine Variable fester Länge, d.h. die Länge (*LENGTH) ändert

	sich nicht als Ergebnis der EXAMINE-Operation, ungeachtet der Tatsache, ob eine DELETE- oder REPLACE-Operation ausgeführt wurde oder nicht.
<i>POSITION-clause</i>	POSITION-Klausel: Bereichswerte für FROM und THRU werden in Form von Code-Einheiten angegeben. Weitere Informationen siehe <i>POSITION-Klausel</i> unter Syntax 1.
CHARPOSITION <i>operand4</i>	CHARPOSITION-Klausel: <i>operand4</i> legt die Startposition (als Unicode-Grapheme) der Graphem-Sequenz fest. Die entsprechende Position wird in <i>operand6</i> in Form von Code-Einheiten zurückgegeben. Diese Klausel kann weggelassen werden, wenn die CHARLENGTH-Klausel angegeben wird; in diesem Fall ist die Startposition 1.
CHARLENGTH <i>operand5</i>	CHARLENGTH-Klausel: <i>operand5</i> legt die Länge der Graphem-Sequenz (als Unicode-Grapheme) fest. Die Länge der Graphem-Sequenz wird in Form von Code-Einheiten in <i>operand7</i> zurückgegeben. Diese Klausel kann weggelassen werden, wenn die CHARPOSITION-Klausel angegeben wird; in diesem Fall wird die Länge von der Startposition bis zum Ende der Zeichenkette zurückgegeben.
GIVING POSITION IN <i>operand6</i>	GIVING POSITION-Klausel: <i>operand6</i> enthält (als Code-Einheiten) die Startposition der von <i>operand4</i> und <i>operand5</i> definierten Graphem-Sequenz. Wenn <i>operand1</i> weniger als <i>operand4</i> Grapheme hat, wird Null (0) zurückgegeben. Diese Klausel kann weggelassen werden, wenn die GIVING LENGTH-Klausel angegeben wird.
GIVING LENGTH IN <i>operand7</i>	GIVING LENGTH-Klausel: <i>operand7</i> enthält (als Code-Einheiten) die Länge der mit <i>operand4</i> und <i>operand5</i> definierten Graphem-Sequenz. Wenn <i>operand1</i> weniger als <i>operand4+operand5</i> Grapheme hat, wird 0 zurückgegeben. Diese Klausel kann weggelassen werden, wenn die GIVING POSITION-Klausel angegeben wird.

**Anmerkungen:**

1. Es muss entweder die CHARPOSITION- oder die CHARLENGTH-Klausel oder beide angegeben werden.
2. Es muss entweder die GIVING POSITION- oder die GIVING LENGTH-Klausel oder beide angegeben werden.

Beispiele

- Beispiel 1 — EXAMINE
- Beispiel 2 — EXAMINE SUBSTRING, PATTERN, TRANSLATE
- Beispiel 3 — EXAMINE TRANSLATE
- Beispiel 4 — EXAMINE für Unicode-Grapheme

Beispiel 1 — EXAMINE

```

** Example 'EXMEX1': EXAMINE
*****
DEFINE DATA LOCAL
1 #TEXT   (A40)
1 #A      (A1)
1 #START  (N2)
1 #NMB1   (N2)
1 #NMB2   (N2)
1 #NMB3   (N2)
1 #NMBEX2 (N2)
1 #NMBEX3 (N2)
1 #NMBEX4 (N2)
1 #POSEX5 (N2)
1 #LGHEX6 (N2)
1 #NMBEX7 (N2)
1 #NMBEX8 (N2)
END-DEFINE
*
WRITE 'EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)'
MOVE 'ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C- ' TO #TEXT
ASSIGN #A = 'A'
EXAMINE #TEXT FOR #A GIVING NUMBER #NMB1
EXAMINE #TEXT FOR #A WITH DELIMITER GIVING NUMBER #NMB2
EXAMINE #TEXT FOR #A WITH DELIMITER '.' GIVING NUMBER #NMB3
WRITE NOTITLE '=' #NMB1 '=' #NMB2 '=' #NMB3
*
WRITE / 'EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR #A WITH DELIMITER '-' REPLACE WITH '*'
      GIVING NUMBER #NMBEX2
WRITE '=' #TEXT '=' #NMBEX2
*
WRITE / 'EXAMPLE 3 (REPLACE, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX3
WRITE '=' #TEXT '=' #NMBEX3
*
WRITE / 'EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)'

```



```

WRITE '=' #TEXT
EXAMINE FULL #TEXT ' ' REPLACE WITH '+' GIVING NUMBER #NMBEX4
WRITE '=' #TEXT '=' #NMBEX4
*
WRITE / 'EXAMPLE 5 (DELETE, GIVING POSITION)'
WRITE '=' #TEXT
EXAMINE #TEXT '+' DELETE GIVING POSITION #POSEX5
WRITE '=' #TEXT '=' #POSEX5
*
WRITE / 'EXAMPLE 6 (DELETE, GIVING LENGTH)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR 'A' DELETE GIVING LENGTH #LGHEX6
WRITE '=' #TEXT '=' #LGHEX6
*
*
NEWPAGE
*
MOVE 'ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-  ' TO #TEXT
*
ASSIGN #A = 'A B C'
ASSIGN #START = 6
*
WRITE / 'EXAMPLE 7 (SUBSTRING, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE SUBSTRING(#TEXT,#START,9) FOR #A GIVING NUMBER #NMBEX7
WRITE '=' #TEXT '=' #NMBEX7
*
WRITE / 'EXAMPLE 8 (PATTERN, GIVING NUMBER)'
WRITE '=' #TEXT
EXAMINE #TEXT FOR PATTERN '-A-' GIVING NUMBER #NMBEX8
WRITE '=' #TEXT '=' #NMBEX8
*
END

```

Ausgabe des Programms EXMEX1:

```

EXAMPLE 1 (GIVING NUMBER, WITH DELIMITER)
#NMB1:  4 #NMB2:  3 #NMB3:  1

EXAMPLE 2 (WITH DELIMITER, REPLACE, GIVING NUMBER)
#TEXT: ABC  A B C  .A.  .B.  .C.  -A-  -B-
#TEXT: ABC  A B C  .A.  .B.  .C.  -*  -B- #NMBEX2:  1

EXAMPLE 3 (REPLACE, GIVING NUMBER)
#TEXT: ABC  A B C  .A.  .B.  .C.  -*  -B-
#TEXT: ABC+++A+B+C+++A.++.B.++.C.++++-*+++B- #NMBEX3:  18

EXAMPLE 4 (FULL, REPLACE, GIVING NUMBER)
#TEXT: ABC+++A+B+C+++A.++.B.++.C.++++-*+++B-
#TEXT: ABC+++A+B+C+++A.++.B.++.C.++++-*+++B-+ #NMBEX4:  1

```

EXAMINE

EXAMPLE 5 (DELETE, GIVING POSITION)

```
#TEXT: ABC+++A+B+C+++ .A. ++.B. ++.C. ++++ -*---B--+
#TEXT: ABCABC.A..B..C.-*--B- #POSEX5: 4
```

EXAMPLE 6 (DELETE, GIVING LENGTH)

```
#TEXT: ABCABC.A..B..C.-*--B-
#TEXT: BCBC...B..C.-*--B- #LGHEX6: 18
```

EXAMPLE 7 (SUBSTRING, GIVING NUMBER)

```
#TEXT: ABC A B C .A. .B. .C. -A- -B-
#TEXT: ABC A B C .A. .B. .C. -A- -B- #NMBEX7: 1
```

EXAMPLE 8 (PATTERN, GIVING NUMBER)

```
#TEXT: ABC A B C .A. .B. .C. -A- -B-
#TEXT: ABC A B C .A. .B. .C. -A- -B- #NMBEX8: 1
```

Beispiel 2 — EXAMINE SUBSTRING, PATTERN, TRANSLATE

```
** Example 'EXMEX2': EXAMINE TRANSLATE
*****
DEFINE DATA LOCAL
1 #TEXT (A50)
1 #TAB (A2/1:10)
1 #START (N2)
END-DEFINE
*
MOVE 'ABC A B C .A. .B. .C. -A- -B- -C- ' TO #TEXT
*
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
*
*
WRITE 'EXAMPLE 1 (USING TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 3 (USING SUBSTRING, LOWER CASE)'
WRITE '=' #TEXT
ASSIGN #START = 13
EXAMINE SUBSTRING(#TEXT,#START,15) TRANSLATE INTO LOWER CASE
WRITE '=' #TEXT
END
```

Ausgabe des Programms EXMEX2:

```

EXAMPLE 1 (USING TRANSLATION TABLE)
#TEXT: ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-
#TEXT: XYZ  X Y Z  .X.  .Y.  .Z.  -X-  -Y-  -Z-

EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)
#TEXT: XYZ  X Y Z  .X.  .Y.  .Z.  -X-  -Y-  -Z-
#TEXT: ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-

EXAMPLE 3 (USING SUBSTRING, LOWER CASE)
#TEXT: ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C-
#TEXT: ABC  A B C  .a.  .b.  .c.  -A-  -B-  -C-

```

Beispiel 3 — EXAMINE TRANSLATE

```

** Example 'EXMEX2': EXAMINE TRANSLATE
*****
DEFINE DATA LOCAL
1 #TEXT (A50)
1 #TAB (A2/1:10)
1 #START (N2)
END-DEFINE
*
MOVE 'ABC  A B C  .A.  .B.  .C.  -A-  -B-  -C- ' TO #TEXT
*
MOVE 'AX' TO #TAB(1)
MOVE 'BY' TO #TAB(2)
MOVE 'CZ' TO #TAB(3)
*
*
WRITE 'EXAMPLE 1 (USING TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)'
WRITE '=' #TEXT
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE NOTITLE '=' #TEXT
*
WRITE / 'EXAMPLE 3 (USING SUBSTRING, LOWER CASE)'
WRITE '=' #TEXT
ASSIGN #START = 13
EXAMINE SUBSTRING(#TEXT,#START,15) TRANSLATE INTO LOWER CASE
WRITE '=' #TEXT
END

```

Ausgabe des Programms EXMEX2:

```

EXAMPLE 1 (USING TRANSLATION TABLE)
#TEXT: ABC   A B C   .A. .B. .C.   -A- -B- -C-
#TEXT: XYZ   X Y Z   .X. .Y. .Z.   -X- -Y- -Z-

EXAMPLE 2 (USING INVERTED TRANSLATION TABLE)
#TEXT: XYZ   X Y Z   .X. .Y. .Z.   -X- -Y- -Z-
#TEXT: ABC   A B C   .A. .B. .C.   -A- -B- -C-

EXAMPLE 3 (USING SUBSTRING, LOWER CASE)
#TEXT: ABC   A B C   .A. .B. .C.   -A- -B- -C-
#TEXT: ABC   A B C   .a. .b. .c.   -A- -B- -C-

```

Beispiel 4 — EXAMINE für Unicode-Grapheme

Dieses Beispiel veranschaulicht die Analyse einer Unicode-Zeichenkette mit den Zeichen ä und ü. Beide Zeichen sind als Basiszeichen, gefolgt von einem Kombinationszeichen festgelegt: ä ist als U+0061, gefolgt von U+0308 kodiert, und ü ist als U+0075, gefolgt von U+0308 kodiert.

```

DEFINE DATA LOCAL
1 #U (U20)
1 #START (I2)
1 #POS (I2)
1 #LEN (I2)
END-DEFINE
#U := U'AB'-UH'00610308'-U'CD'-UH'00750308'-U'EF'
*
REPEAT
  #START := #START + 1
  EXAMINE #U FOR CHARPOSITION #START
                CHARLENGTH      1
                GIVING POSITION IN #POS
                LENGTH IN #LEN
*
  INPUT (AD=0) MARK POSITION #POS IN FIELD *#U
  '          UNICODE-STRING:' #U      (AD=MI)
// '          CHARACTER NO.:' #START (EM=9)
/ 'STARTS AT BYTE POSITION:' #POS      (EM=9)
/ '          AND THE LENGTH IS:' #LEN  (EM=9)
WHILE #POS NE 0
END-REPEAT
END

```

Ausgabe:

Großrechner-Umgebungen:	Windows-, UNIX- und OpenVMS-Umgebungen (mit Natural Web I/O Interface):
UNICODE-STRING: ABa?CDu?EF CHARACTER NO.: 1 STARTS AT BYTE POSITION: 1 AND THE LENGTH IS: 1	UNICODE-STRING: ABäCDüEF CHARACTER NO.: 1 STARTS AT BYTE POSITION: 1 AND THE LENGTH IS: 1
Drücken Sie die Eingabetaste um fortzufahren.	Drücken Sie die Eingabetaste um fortzufahren.
UNICODE-STRING: ABa?CDu?EF CHARACTER NO.: 2 STARTS AT BYTE POSITION: 2 AND THE LENGTH IS: 1	UNICODE-STRING: ABäCDüEF CHARACTER NO.: 2 STARTS AT BYTE POSITION: 2 AND THE LENGTH IS: 1
Drücken Sie die Eingabetaste um fortzufahren.	Drücken Sie die Eingabetaste um fortzufahren.
Beachten Sie, dass das Zeichen in Position 3 eine Kombinationszeichenfolge ist und zwei Code-Einheiten lang ist.	
UNICODE-STRING: ABa?CDu?EF CHARACTER NO.: 3 STARTS AT BYTE POSITION: 3 AND THE LENGTH IS: 2	UNICODE-STRING: ABäCDüEF CHARACTER NO.: 3 STARTS AT BYTE POSITION: 3 AND THE LENGTH IS: 2
Und so weiter.	Und so weiter.

61 EXPAND

▪ Funktion	364
▪ Syntax-Beschreibung	364

EXPAND	$\left\{ \begin{array}{l} \textit{dynamic-clause} \\ \textit{array-clause} \end{array} \right\}$	[GIVING <i>operand5</i>]
--------	--	---------------------------

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [REDUCE](#) | [RESIZE](#)

Gehört zur Funktionsgruppe: [Speicherverwaltungskontrolle für dynamische Variablen/X-Arrays](#)

Funktion

Das Statement EXPAND dient dazu,

- die zugewiesene Länge einer dynamischen Variable (*dynamic-clause*) oder
- die Anzahl der Ausprägungen von X-Arrays (*array-clause*)

Weitere Informationen entnehmen Sie den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- *Dynamische Variablen benutzen*
- *Hauptspeicherplatz für eine dynamische Variable zuweisen/freigeben*
- *X-Arrays*
- *Speicherverwaltung von X-Gruppen-Arrays*

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate												Referenzierung erlaubt	Dynam. Definition													
<i>operand1</i>	S	A				A	U							B											nein	nein			
<i>operand2</i>	C	S																								nein	nein		
<i>operand3</i>			A	G			A	U	N	P	I	F	B	D	T	L	C	G	O									ja	nein
<i>operand4</i>	C	S								N	P	I														nein	nein		
<i>operand5</i>	S																									nein	ja		

Syntax-Element-Beschreibung:

<i>dynamic-clause</i>	<p>DYNAMIC-Klausel:</p> <p>Mit dem Statement <code>EXPAND DYNAMIC VARIABLE</code> können Sie die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variable (<i>operand1</i>) auf den mit <i>operand2</i> angegebenen Wert erweitern.</p> <p>Siehe DYNAMIC-Klausel weiter unten.</p>
<i>operand1</i>	<p>Zu erweiternde Variable:</p> <p><i>operand1</i> ist die dynamische Variable, für die die zugewiesene Länge erweitert werden soll.</p>
<i>operand2</i>	<p>Erweiterungswert:</p> <p><i>operand2</i> dient dazu, die Länge anzugeben, auf die die dynamische Variable erweitert werden soll. Der angegebene Wert muss eine nicht negative, numerische Ganzzahl-Konstante oder eine Variable des Typs Integer4 (I4) sein.</p>
<i>array-clause</i>	<p>Array-Klausel:</p> <p>Mit dem Statement <code>EXPAND ARRAY</code> können Sie Anzahl der Ausprägungen des X-Arrays (<i>operand3</i>) auf die mit (<i>dim[, dim[, dim]]</i>) angegebene Ober- und Untergrenze erweitern.</p> <p>Siehe Array-Klausel weiter unten.</p>
<i>operand3</i>	<p>Zu erweiterndes X-Array:</p> <p><i>operand3</i> ist das X-Array, für das die Anzahl der Ausprägungen erweitert werden kann. Die Index-Notation des Arrays ist optional. Als Index-Notation ist nur die Stern-Notation (*) für den vollständigen Bereich für jede Dimension zulässig.</p>
dim <i>operand4</i>	<p>Ober-/Untergrenze der Erweiterung:</p> <p>Die Notation für die Ober- und Untergrenze (<i>operand4</i> oder Stern-Notation), auf die das X-Array erweitert werden sollte, wird hier angegeben. Wenn der aktuelle Wert für die Ober- oder Untergrenze benutzt werden sollte, kann ein Stern-Notation (*) anstatt <i>operand4</i> angegeben werden.</p> <p>Siehe Dimension weiter unten.</p>
GIVING <i>operand5</i>	<p>GIVING-Klausel</p> <p>Wenn diese Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt.</p> <p>Wenn die Klausel angegeben wird, enthält <i>operand5</i> die Natural- Fehlernummer, wenn vorher ein Fehler aufgetreten ist, oder Null (0) bei Erfolg.</p>

DYNAMIC-Klausel

```
[SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
```

Mit dem Statement `EXPAND DYNAMIC VARIABLE` können Sie die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variable (*operand1*) auf den mit *operand2* angegebenen Wert erweitern.

Ist *operand2* kleiner als die aktuell zugewiesene Länge von *operand1*, wird das Statement für diese dynamische Variable ignoriert. Die aktuell benutzte Länge (`*LENGTH`) der dynamischen Variable wird nicht geändert.

Array-Klausel

```
[AND RESET [OCCURRENCES OF] ARRAY operand3 TO (dim[,dim[,dim]])]
```

Mit dem Statement `EXPAND ARRAY` können Sie die Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit `TO (dim [,dim [,dim])` angegebene Ober- und Untergrenze erweitern, wobei jede Angabe von *dim* sich auf eine Dimension bezieht, die mittels der weiter unten beschriebenen Syntax definiert wird.

Mit der `RESET`-Option setzen Sie alle Ausprägungen des größtmäßig angepassten X-Arrays auf ihren standardmäßigen Nullwert zurück. Als Voreinstellung (keine `RESET`-Option) werden die Direktwerte beibehalten und die größtmäßig angepassten (neuen) Ausprägungen zurückgesetzt.

Verwenden Sie das `EXPAND`-Statement, ist es nur möglich, die Anzahl der Ausprägungen zu erhöhen. Wenn die erforderliche Anzahl kleiner ist als die aktuell zugewiesene Anzahl der Ausprägungen, wird dies einfach ignoriert.

Eine bei einem `EXPAND`-Statement eingesetzte Ober- oder Untergrenze muss genau der betreffenden, für das Array definierten Ober- oder Untergrenze entsprechen.

Beispiel:

```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...
/* allocating #a(1:10)
EXPAND ARRAY #a TO (1:10)          /* #a is allocated 10
EXPAND ARRAY #a TO (*:10)         /* occurrences.
```

```

/* allocating #ga(1:10,1:20)
EXPAND ARRAY #g TO (1:10)      /* 1st dimension is set to (1:10)
EXPAND ARRAY #ga TO (*:*,1:20) /* 1st dimension is dependent and
                               /* therefore kept with (*:*)
                               /* 2nd dimension is set to (1:20)

EXPAND ARRAY #a TO (5:10)      /* This is rejected because the lower index
                               /* must be 1 or *
EXPAND ARRAY #a TO (#i:10)     /* This is rejected because the lower index
                               /* must be 1 or *

EXPAND ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                               /* because the dimension is dependent and
                               /* must be specified with (*:*).

```

Weitere Informationen siehe

- *Speicherverwaltung von X-Arrays*
- *Speicherverwaltung von X-Gruppen-Arrays*

Dimension

Jede der in der *Array-Klausel* angegebenen Dimensionen (*dim*) wird mittels der folgenden Syntax definiert:

$$\left\{ \begin{array}{c} \textit{operand4} \\ * \end{array} \right\} : \left\{ \begin{array}{c} \textit{operand4} \\ * \end{array} \right\}$$

Die Notation für Ober- und Untergrenzen (*operand4* oder Stern-Notation), auf die das X-Array erweitert werden sollte, wird hier angegeben. Wenn der aktuelle Wert der Ober- oder Untergrenze benutzt werden soll, kann ein Stern (*) anstelle von *operand4* angegeben werden. Anstatt *: * können Sie auch einen einzelnen Stern verwenden.

Die Anzahl der Dimensionen (*dim*) muss genau mit der definierten Anzahl der Dimensionen des X-Arrays (1, 2 oder 3) übereinstimmen.

Wenn die Anzahl der Ausprägungen für eine angegebene Dimension kleiner ist als die Anzahl der aktuell zugewiesenen Ausprägungen, wird die Anzahl der Ausprägungen nicht für die betreffende Dimension aktualisiert.

62 FETCH

▪ Funktion	370
▪ Syntax-Beschreibung	371
▪ Beispiel	372

FETCH [{ REPEAT }] operand1 [operand2 [(parameter)]] ...
--

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: CALL | CALL FILE | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | ESCAPE | FETCH | PERFORM

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Das Statement `FETCH` dient dazu, ein Natural-Objektprogramm auszuführen, welches als Hauptprogramm geschrieben wurde. Das zu ladende Programm muss vorher mit einem `STOW-` oder `CATALOG-`Kommando in der Natural-Systemdatei in Objektform gespeichert worden sein. Ein im Arbeitsbereich des Editors befindliches Sourceprogramm wird durch die Ausführung eines `FETCH`-Statements nicht überschrieben.

Für Natural RPC: Siehe *Notes on Natural Statements on the Server* in der *Natural Remote Procedure Call (RPC)*-Dokumentation.

Zusätzliche Anmerkungen

Zusätzlich zu den explizit mit dem `FETCH`-Statement übergebenen Parametern hat das aufgerufene Programm Zugang zu der Global Data Area des aufrufenden Programms.

Je nachdem, wie Ihr Natural-Administrator den Natural-Profilparameter `OPRB` (*Database Open/Close Processing*) gesetzt hat, kann es sein, dass das `FETCH`-Statement die Ausführung eines internen `END TRANSACTION`-Statements auslöst. Soll eine logische Transaktion mehrere Programme einschließen, so wenden Sie sich bitte vorher an Ihren Natural-Administrator, um sicherzustellen, dass der `OPRB`-Parameter entsprechend gesetzt ist.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	ja	nein
<i>operand2</i>	C S A G	A U N P I F B D T L G	ja	ja

Syntax-Element-Beschreibung:

REPEAT	<p>Wegfall der Notwendigkeit einer Benutzer-Interaktion:</p> <p>REPEAT bewirkt, dass bei der Ausführung des aufgerufenen Programms der Benutzer bei INPUT-Statements keine Eingaben machen muss. REPEAT kann auch dazu eingesetzt werden, Informationen über die Ausführung des Programms am Bildschirm anzuzeigen, ohne dass der Benutzer daraufhin EINGABE drücken muss.</p>
RETURN	<p>Aufrufen und Ausführen eines Objekts des Typs Programm als Routine:</p> <p>Wird RETURN nicht angegeben, so wird das aufrufende Programm, welches das FETCH-Statement enthält, augenblicklich beendet, und das aufgerufene Programm wird als Hauptprogramm (Stufe 1) aktiviert.</p> <p>Geben Sie FETCH RETURN an, wird das aufrufende Programm nicht beendet, sondern nur unterbrochen, während das aufgerufene Programm als Unterprogramm auf einer höheren Stufe ausgeführt wird. Ein END- oder ESCAPE ROUTINE-Statement im aufgerufenen Programm bewirkt, dass die Kontrolle wieder an das aufrufende Programm übergeben wird, dessen Ausführung dann mit dem auf das FETCH RETURN folgende Statement fortgesetzt wird.</p>
<i>operand1</i>	<p>Programm-Name:</p> <p>Der Name des aufgerufenen Programms (maximal 8 Zeichen lang) kann entweder als alphanumerische Konstante oder als Inhalt einer alphanumerischen Variablen der Länge 1 bis 8 angegeben werden.</p> <p>Natural sucht das Programm zunächst in der zum Zeitpunkt der Ausführung des FETCH-Statements gerade aktiven Library. Wird es dort nicht gefunden, sucht Natural in den Steplibs. Wird das Programm auch dort nicht gefunden, gibt Natural eine entsprechende Fehlermeldung aus.</p> <p>Der Name des Programms darf ein Und-Zeichen (&) enthalten; zur Laufzeit wird dieses Zeichen durch den aus einem Zeichen bestehenden Code ersetzt, der dem aktuellen Wert der Systemvariablen *LANGUAGE entspricht. Dadurch ist es beispielsweise möglich, je nachdem in welcher Sprache eine Eingabe gemacht wird, zur Verarbeitung der Eingabe unterschiedliche Programme aufzurufen.</p>

<i>operand2</i>	<p>Zu übergebende Parameter-Felder:</p> <p>Mit dem <code>FETCH</code>-Statement können auch Parameterfelder an das aufgerufene Programm übergeben werden. Ein Parameterfeld kann mit einem beliebigen Format definiert werden; das Format wird dem jeweiligen <code>INPUT</code>-Feld entsprechend umgesetzt. Sämtliche Parameter werden oben auf dem Natural-Stack abgelegt.</p> <p>Das aufgerufene Programm liest die übergebenen Parameterfelder über ein <code>INPUT</code>-Statement. Das erste <code>INPUT</code>-Statement bewirkt, dass die Werte aller Parameterfelder in die mit dem <code>INPUT</code>-Statement angegebenen Felder übertragen werden. Da jedes mit einem numerischen Format definierte Parameterfeld eine Stelle für das Vorzeichen erhält, wenn sein Wert negativ ist, muss beim <code>INPUT</code>-Statement der Session-Parameter <code>SG</code> für die Parameterfelder auf <code>SG=ON</code> gesetzt werden.</p> <p>Werden mehr Parameter übergeben als vom <code>INPUT</code>-Statement gelesen werden können, so werden überschüssige Parameter ignoriert. Die Anzahl der Parameter kann mit Hilfe der Natural-Systemvariablen <code>*DATA</code> ermittelt werden.</p> <p>Anmerkung: Wenn <i>operand2</i> eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.</p>
<i>parameter</i>	<p>Datumsformat für Datumsvariable:</p> <p>Wenn <i>operand2</i> eine Datumsvariable ist, können Sie den Session-Parameter <code>DF</code> (Date Format) als <i>parameter</i> für diese Variable angeben.</p>

Beispiel

Aufrufendes Programm FETEX1:

```

** Example 'FETEX1': FETCH (with parameter)
*****
DEFINE DATA LOCAL
1 #PNUM (N8)
1 #FNC (A1)
END-DEFINE
*
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (35) //
      10X 'ADD (A)' /
      10X 'UPDATE (U)' /
      10X 'DELETE (D)' /
      10X 'STOP (.)' //
      10X 'PLEASE ENTER FUNCTION: ' #FNC ///
      10X 'PERSONNEL NUMBER:' #PNUM
*
DECIDE ON EVERY VALUE OF #FNC
VALUE 'A', 'U', 'D'

```



```

    IF #PNUM = 0
        REINPUT 'PLEASE ENTER A VALID NUMBER' MARK *#PNUM
    END-IF
    VALUE 'A'
        FETCH 'FETEXAD' #PNUM
    VALUE 'U'
        FETCH 'FETEXUP' #PNUM
    VALUE 'D'
        FETCH 'FETEXDE' #PNUM
    VALUE '.'
        STOP
    NONE
        REINPUT 'PLEASE ENTER A VALID FUNCTION' MARK *#FNC
END-DECIDE
*
END

```

Aufgerufenes Programm FETEXAD:

```

** Example 'FETEXAD': FETCH (called by FETEX1)
*****
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record added with personnel number:' #PERS-NR
*
END

```

Aufgerufenes Programm FETEXUP:

```

** Example 'FETEXUP': FETCH (called by FETEX1)
*****
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record updated with personnel number:' #PERS-NR
*
END

```

Aufgerufenes Programm FETEXDE:

```
** Example 'FETEXDE': FETCH (called by FETEX1)
*****
DEFINE DATA LOCAL
1 #PERS-NR (N8)
END-DEFINE
*
INPUT #PERS-NR
*
WRITE *PROGRAM 'Record deleted with personnel number:' #PERS-NR
*
END
```

Ausgabe des Programms FETEX1:

```
SELECTION MENU FOR EMPLOYEES SYSTEM
-----
ADD      (A)
UPDATE  (U)
DELETE  (D)
STOP    (.)

PLEASE ENTER FUNCTION: D

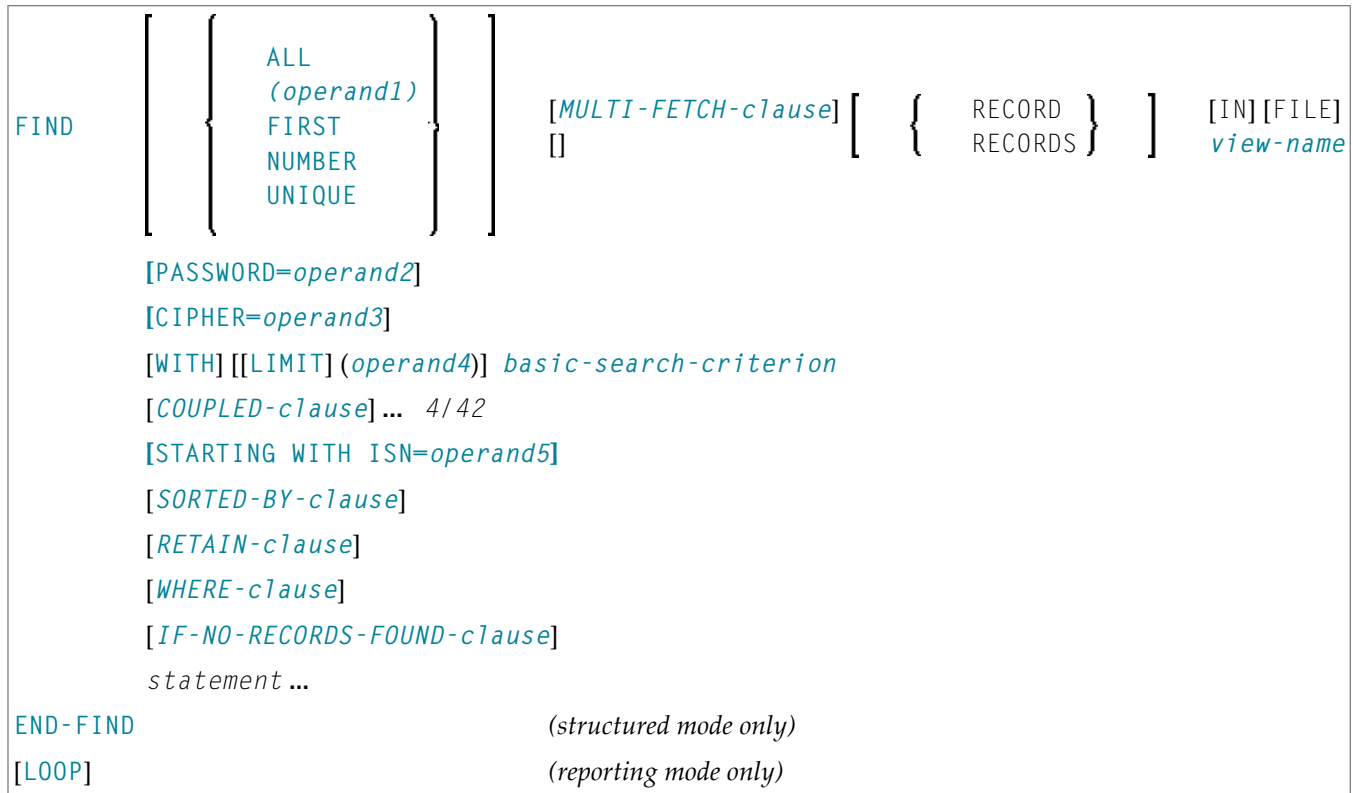
PERSONNEL NUMBER: 1150304
```

Nach Eingabe und Bestätigung der Funktion und Personalnummer:

```
Page      1                                05-01-13  11:58:46
FETEXDE Record deleted with personnel number: 1150304
```

63 FIND

▪ Funktion	376
▪ Einschränkungen	378
▪ Syntax-Beschreibung	378
▪ Beispiele	400



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement FIND dient dazu, Datensätze von der Datenbank auszuwählen, und zwar anhand eines Suchkriteriums, d.h. des Wertes eines Schlüsselfeldes (Deskriptors).

Mit dem FIND-Statement wird eine Verarbeitungsschleife initiiert, die für jeden gefundenen Datensatz durchlaufen wird. Innerhalb der FIND-Schleife kann jedes Feld eines gefundenen Datensatzes referenziert werden, und zwar ohne dass hierzu ein zusätzliches READ-Statement erforderlich wäre.

Siehe auch *FIND Statement* im *Leitfaden zur Programmierung*.

Datenbank-spezifische Anmerkungen

SQL	<p>FIND FIRST sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt.</p> <p>FIND UNIQUE ist nicht erlaubt. (Ausnahme: Auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)</p> <p>Die SORTED BY-Klausel entspricht der SQL-Klausel ORDER BY. Das Suchkriterium (Basic-Search-Criterion) für eine SQL-Datenbank-Tabelle kann genauso angegeben werden wie für eine Adabas-Datei. Der Begriff <i>Datensatz</i> (Record) ist in diesem Zusammenhang dem SQL-Begriff <i>Reihe</i> (Row) gleichzusetzen.</p>
XML	<p>FIND FIRST sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt.</p> <p>FIND UNIQUE ist nicht erlaubt.</p> <p>Das Suchkriterium (Basic-Search-Criterion) für eine XML-Datenbank kann genauso angegeben werden wie für eine Adabas-Datei. Der Begriff <i>Datensatz</i> (Record) ist in diesem Zusammenhang dem SQL-Begriff <i>XML-Objekt</i> (XML Object) gleichzusetzen.</p>

Systemvariablen beim FIND-Statement

Die Natural-Systemvariablen *ISN, *NUMBER und *COUNTER werden automatisch für jedes FIND-Statement erzeugt. Wird eine Systemvariable außerhalb der aktuellen Verarbeitungsschleife oder über ein FIND FIRST-, FIND NUMBER- oder FIND UNIQUE-Statement referenziert, muss mittels Statement-Label oder Sourcecode-Zeilenummer referenziert werden. Alle drei Systemvariablen haben Format/Länge P10; diese(s) Format/Länge kann nicht geändert werden.

*ISN	Adabas	Bei Adabas-Datenbanken enthält *ISN die Adabas-ISN (Interne Satz-Nummer) des gerade verarbeiteten Datensatzes. *ISN kann nicht bei FIND NUMBER verwendet werden.
	Tamino	*ISN enthält die XML-Objekt-ID.
	SQL	*ISN ist nicht verfügbar.
	Entire System Server	*ISN ist nicht verfügbar.
*NUMBER	Adabas	*NUMBER enthält die Anzahl der Datensätze, die das in der WITH-Klausel angegebene primäre Suchkriterium erfüllt haben.
	Tamino	Siehe *NUMBER für SQL Databases in the Systemvariablen-Dokumentation.
	Entire System Server	*NUMBER ist nicht verfügbar.
*COUNTER	*COUNTER enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen worden ist.	

Siehe auch [Beispiel 13 - Systemvariablen mit dem FIND-Statement benutzen](#).

Mehrere FIND-Statements

Es ist möglich, mehrere FIND-Schleifen ineinander zu verschachteln. Hierbei wird eine jeweils innere Schleife für jeden Datensatz, der mit der jeweils äußeren Schleife ausgewählt wurde, durchlaufen. Siehe auch [Beispiel 14 – Mehrere FIND-Statements](#).

Einschränkungen

Bei Entire System Server sind FIND NUMBER und FIND UNIQUE sowie die Klauseln PASSWORD, CIPHER, COUPLED und RETAIN nicht zulässig.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C	S	N P I B*	ja	nein
<i>operand2</i>	C	S	A	ja	nein
<i>operand3</i>	C	S	N	ja	nein
<i>operand4</i>	C	S	N P I B*	ja	nein
<i>operand5</i>	C	S	N P I B*	ja	nein

* Format B von *operand1*, *operand4* und *operand5* kann nur mit einer Länge von kleiner oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

ALL <i>operand1</i>	<p>Begrenzung der Verarbeitung:</p> <p>Sie können die Anzahl der ausgewählten Datensätze, die in der FIND-Schleife verarbeitet werden sollen, auf eine bestimmte Zahl begrenzen. Diese Zahl (<i>operand1</i>) geben Sie in Klammern unmittelbar nach dem Schlüsselwort FIND an, und zwar entweder als numerische Konstante (im Bereich von 0 bis 4294967295) oder in Form einer numerischen Variablen.</p> <p>Andernfalls werden alle gefundenen Sätze weiterverarbeitet, was Sie zusätzlich durch das Schlüsselwort ALL hervorheben können.</p>
----------------------------	---

	<p>Geben Sie mit <i>operand1</i> ein Limit an, so gilt dieses Limit für die initiierte FIND-Schleife, wobei allerdings Datensätze, die aufgrund einer WHERE-Klausel abgelehnt werden, nicht mitgezählt werden.</p> <pre>FIND (5) IN EMPLOYEES WITH ... MOVE 10 TO #CNT(N2) FIND (#CNT) EMPLOYEES WITH ...</pre> <p>Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.</p> <p>Ist mit dem LT-Parameter ein kleineres Limit gesetzt, so gilt das LT-Limit.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn Sie eine vierstellige Anzahl von Datensätzen verarbeiten möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement. 2. <i>operand1</i> hat keinen Einfluss auf die Größe eines ISN-Set, der mittels einer RETAIN-Klausel erzeugt wird. <i>operand1</i> wird zu Beginn des ersten FIND-Schleifendurchlaufs ausgewertet. Wird der Wert von <i>operand1</i> innerhalb der FIND-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der verarbeiteten Datensätze.
<p>FIND FIRST FIND NUMBER FIND UNIQUE</p>	<p>Suchoptionen:</p> <p>Diese Optionen dienen dazu</p> <ul style="list-style-type: none"> ■ nur den ersten der gefundenen Datensätze auszuwählen (FIND FIRST), ■ die Anzahl der gefundenen Datensätze zu ermitteln (FIND NUMBER), ■ bzw. sicherzustellen, dass nur ein Datensatz das Suchkriterium erfüllt (FIND UNIQUE). <p>Näheres hierzu finden Sie in den entsprechenden Abschnitten weiter unten.</p>
<p><i>MULTI-FETCH-clause</i></p>	<p>Multi-Fetch-Klausel:</p> <p>Bei Adabas-Datenbanken: Im Standard-Modus liest Natural nicht mehrere Datensätze auf einmal bei einem einzigen Datenbankaufruf, sondern jeweils nur einen. Diese stabile Betriebsart kann aber einige Zeit in Anspruch nehmen, wenn eine größere Anzahl von Datenbanksätzen verarbeitet wird.</p>

	<p>Um die Leistung dieser Programme zu verbessern, bietet Natural die MULTI-FETCH-Klausel, die es ermöglicht, mehr als einen Datensatz pro Datenbankzugriff zu lesen.</p> <p>Weitere Informationen, siehe MULTI-FETCH-Klausel weiter unten.</p>
<i>view-name</i>	<p>View-Angabe:</p> <p>Der Name eines Views, der entweder in einem DEFINE DATA-Statement-Block oder in einer separaten Global oder Local Data Area definiert ist.</p> <p>Im Reporting Mode ist <i>view-name</i> der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.</p>
PASSWORD= <i>operand2</i>	<p>PASSWORD-Klausel:</p> <p>Die PASSWORD-Klausel gilt nur für Zugriffe auf Adabas-Datenbanken.</p> <p>Mit Entire System Server ist diese Klausel nicht erlaubt.</p> <p>Die PASSWORD-Klausel dient dazu, ein Passwort (<i>operand2</i>) anzugeben, um auf Daten einer passwortgeschützten Adabas-Datei zugreifen zu können. Wollen Sie auf eine passwortgeschützte Datei zugreifen, sollten Sie sich bezüglich des Passwortes mit Ihrem Datenbank-Security-Administrator in Verbindung setzen.</p> <p>Wenn das Passwort als Konstante angegeben wird, sollte die PASSWORD-Klausel ganz am Anfang einer Sourcecode-Zeile stehen und es sollte sich zwischen dem Schlüsselwort PASSWORD und dem Gleichheitszeichen kein Leerzeichen befinden; dadurch ist gewährleistet, dass das Passwort im Sourcecode nicht sichtbar ist.</p> <p>Im TP-Modus können Sie die PASSWORD-Klausel unsichtbar machen, indem Sie dann zuerst das Terminalkommando %* eingeben, bevor Sie das Passwort eintippen.</p> <p>Wenn Sie die PASSWORD-Klausel weglassen, gilt das mit dem PASSW-Statement angegebene Passwort.</p> <p>Während der Ausführung einer Verarbeitungsschleife kann das Passwort nicht geändert werden.</p> <p>Siehe auch Beispiel 1 – PASSWORD-Klausel.</p>
CIPHER= <i>operand3</i>	<p>CIPHER-Klausel:</p> <p>Die CIPHER-Klausel gilt nur für Zugriffe auf Adabas-Dateien.</p> <p>Mit Entire System Server ist diese Klausel nicht erlaubt.</p> <p>Die CIPHER-Klausel dient dazu, einen Chiffrierschlüssel (<i>operand3</i>) anzugeben, um in chiffrierter Form gespeicherte Daten von Adabas-Dateien in entschlüsselter Form zu erhalten. Wollen Sie auf eine</p>

	<p>chiffrierte Datei zugreifen, sollten Sie sich bezüglich des Chiffrierschlüssels mit Ihrem Datenbank-Security-Administrator in Verbindung setzen.</p> <p>Der Chiffrierschlüssel kann als numerische Konstante (8 Stellen lang) oder als Inhalt einer Benutzervariablen (Format/Länge N8) angegeben werden.</p> <p>Wird der Chiffrierschlüssel als Konstante angegeben, sollte die CIPHER-Klausel ganz am Anfang einer Sourcecode-Zeile stehen; dadurch ist gewährleistet, dass der Chiffrierschlüssel im Sourcecode nicht sichtbar ist.</p> <p>Im TP-Modus können Sie die CIPHER-Klausel unsichtbar machen, indem Sie zuerst das Terminalkommando %* eingeben, bevor Sie den Chiffrierschlüssel eintippen.</p> <p>Während der Ausführung der FIND-Verarbeitungsschleife kann der Chiffrierschlüssel nicht geändert werden.</p> <p>Siehe auch Beispiel 2 – CIPHER-Klausel.</p>		
<p>WITH LIMIT <i>operand4</i></p> <p><i>basic-search-criterion</i></p>	<p>WITH-Klausel:</p> <p>Die WITH-Klausel ist unbedingt erforderlich. Mit ihr wird das Suchkriterium (basic-search-criterion) in Form des Wertes eines als Schlüsselfeld (Deskriptor) definierten Datenbankfeldes angegeben. Siehe Suchkriterien für Adabas-Dateien.</p> <p>Datenbank-spezifischer Hinweis:</p> <table border="1" data-bbox="646 1161 1477 1486"> <tr> <td data-bbox="646 1161 1010 1486"></td> <td data-bbox="1010 1161 1477 1486"> <p>Sie können in der WITH-Klausel einen Deskriptor, einen Subdeskriptor, einen Superdeskriptor, einen Hyperdeskriptor oder einen phonetischen Deskriptor angeben. Auf Großrechnern kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit N markiert ist) angegeben werden.</p> </td> </tr> </table> <p>Die Anzahl der Datensätze, die anhand des in der WITH-Klausel definierten Suchkriteriums ausgewählt werden sollen, können Sie begrenzen, indem Sie das Schlüsselwort LIMIT und dahinter in Klammern eine Zahl oder eine Benutzervariable angeben (<i>operand4</i>). Übersteigt die Anzahl der ausgewählten Datensätze diese Zahl, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen.</p> <p>Anmerkung: Wenn das Limit eine vierstellige Zahl sein soll, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.</p>		<p>Sie können in der WITH-Klausel einen Deskriptor, einen Subdeskriptor, einen Superdeskriptor, einen Hyperdeskriptor oder einen phonetischen Deskriptor angeben. Auf Großrechnern kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit N markiert ist) angegeben werden.</p>
	<p>Sie können in der WITH-Klausel einen Deskriptor, einen Subdeskriptor, einen Superdeskriptor, einen Hyperdeskriptor oder einen phonetischen Deskriptor angeben. Auf Großrechnern kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit N markiert ist) angegeben werden.</p>		

<i>COUPLED-clause</i>	<p>COUPLED-Klausel:</p> <p>Diese Klausel gilt nur für Such-Zugriffe auf Adabas-Dateien. Siehe COUPLED-Klausel.</p>
STARTING WITH ISN=operand5	<p>STARTING WITH-Klausel:</p> <p>Diese Klausel kann zum Repositionieren innerhalb einer FIND-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden.</p> <p>Siehe STARTING WITH-Klausel.</p>
<i>SORTED-BY-clause</i>	<p>SORTED BY-Klausel:</p> <p>Diese Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren.</p> <p>Siehe SORTED BY-Klausel.</p>
<i>RETAIN-clause</i>	<p>RETAIN-Klausel:</p> <p>Mit dieser Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen.</p> <p>Siehe RETAIN-Klausel.</p>
<i>WHERE-clause</i>	<p>WHERE-Klausel:</p> <p>Diese Klausel dient dazu, ein zusätzliches Selektionskriterium (<i>logical-condition</i>) anzugeben.</p> <p>Siehe WHERE-Klausel.</p>
<i>IF-NO-RECORDS-FOUND-clause</i>	<p>IF NO RECORDS FOUND-Klausel:</p> <p>In dieser Klausel können Sie eine Schleife angeben, die mit einem FIND-Statement ausgeführt werden soll für den Fall, dass kein Datensatz die in der WITH- und WHERE-Klausel des FIND-Statements angegebenen Selektionskriterien erfüllt.</p> <p>Siehe IF NO RECORDS FOUND-Klausel.</p>
END-FIND	<p>Das für Natural reservierte Schlüsselwort END-FIND muss zum Beenden des FIND-Statements verwendet werden.</p>

FIND FIRST

Das `FIND FIRST`-Statement dient dazu, den ersten Datensatz, der die `WITH`- und `WHERE`-Selektionskriterien erfüllt, auszuwählen und zu verarbeiten.

Bei Adabas-Dateien wird derjenige der ausgewählten Datensätze verarbeitet, der die niedrigste Adabas-ISN (Interne Satznummer) hat.

`FIND FIRST` initiiert *keine* Verarbeitungsschleife.

Einschränkungen bei FIND FIRST

- `FIND FIRST` darf nur im Reporting Mode verwendet werden.
- `FIND FIRST` ist beim Zugriff auf SQL-Datenbanken nicht möglich.
- Ein `FIND FIRST`-Statement darf keine `IF NO RECORDS FOUND`-Klausel enthalten.

Systemvariablen bei FIND FIRST

Beim `FIND FIRST`-Statement stehen folgende Natural-Systemvariablen zur Verfügung:

*ISN	Die Systemvariable <code>*ISN</code> enthält die Adabas-ISN (Interne Satznummer) des ausgewählten Datensatzes. <code>*ISN</code> enthält den Wert Null (0), wenn kein Datensatz die <code>WITH</code> - und <code>WHERE</code> -Selektionskriterien erfüllt. Mit Entire System Server kann <code>*ISN</code> nicht verwendet werden.
*NUMBER	Die Systemvariable <code>*NUMBER</code> enthält die Anzahl der Datensätze, die vor Auswertung des <code>WHERE</code> -Kriteriums das in der <code>WITH</code> -Klausel angegebene primäre Selektionskriterium erfüllt haben. <code>*NUMBER</code> enthält den Wert Null (0), wenn kein Datensatz das <code>WITH</code> -Kriterium erfüllt. Mit Entire System Server kann <code>*NUMBER</code> nicht verwendet werden.
*COUNTER	Die Systemvariable <code>*COUNTER</code> enthält 1, wenn ein Datensatz gefunden wurde, sie enthält 0, wenn kein Datensatz gefunden wurde.

Beispiel für `FIND FIRST` siehe Programm `FNDFIR` (Reporting Mode).

FIND NUMBER

Mit dem Statement `FIND NUMBER` können Sie ermitteln, wieviele Datensätze die `WITH`- und `WHERE`-Kriterien erfüllen. `FIND NUMBER` löst *keine* Verarbeitungsschleife aus *und liest auch keine* Datensätze von der Datenbank.



Anmerkung: Eine `WHERE`-Klausel kann hierbei einen beträchtlichen Verarbeitungsmehraufwand verursachen.

Einschränkungen bei FIND NUMBER

- Ein FIND NUMBER-Statement darf keine SORTED BY- oder IF NO RECORDS FOUND-Klausel enthalten.
- Im Structured Mode darf keine WHERE-Klausel benutzt werden.
- Mit Entire System Server kann FIND NUMBER nicht verwendet werden.

Systemvariablen bei FIND NUMBER

Bei FIND NUMBER stehen folgende Natural-Systemvariablen zur Verfügung:

*NUMBER	Die Systemvariable *NUMBER enthält die Anzahl der Datensätze, die das in der WITH-Klausel angegebene primäre Selektionskriterium erfüllt haben.
*COUNTER	Die Systemvariable *COUNTER enthält die Anzahl der Datensätze, die die Selektionskriterien der WHERE-Klausel erfüllt haben. *COUNTER steht nur zur Verfügung, wenn das FIND NUMBER-Statement eine WHERE-Klausel enthält.

Beispiel für FIND NUMBER siehe das Programm FNDNUM (Reporting Mode).

FIND UNIQUE

Dieses Statement gewährleistet, dass nur ein einziger Datensatz die Selektionskriterien erfüllt. FIND UNIQUE initiiert keine Verarbeitungsschleife. Enthält das FIND UNIQUE-Statement eine WHERE-Klausel, wird zur Auswertung dieser Klausel eine automatische interne Verarbeitungsschleife durchlaufen.

Wird kein oder mehr als ein Datensatz gefunden, wird eine entsprechende Fehlermeldung ausgegeben; dieser Fall kann mit einem ON ERROR-Statement abgefangen werden.

Einschränkungen bei FIND UNIQUE

- FIND UNIQUE darf nur im Reporting Mode verwendet werden.
- FIND UNIQUE ist beim Einsatz von Entire System Server nicht möglich.
- Bei SQL-Datenbanken ist FIND UNIQUE nicht erlaubt. (Ausnahme: auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)
- Ein FIND UNIQUE-Statement darf keine SORTED BY- oder IF NO RECORDS FOUND-Klausel enthalten.

Systemvariablen bei FIND UNIQUE

*ISN	Die Systemvariable *ISN enthält die eindeutige ISN-Nummer des Datensatzes, der selbst wiederum eindeutig sein muss.
*NUMBER	Die Systemvariable *NUMBER enthält bei einer gültigen Ausführung des FIND UNIQUE-Statements immer eine 1. *NUMBER kann einen anderen positiven Wert (=0 oder >=2) enthalten, wenn ein Fehler aufgetreten ist. Diese Fehlerbedingung kann vom ON ERROR-Statement benutzt werden. *NUMBER ist nicht zulässig, wenn die WHERE-Klausel fehlt.
*COUNTER	Die Systemvariable *COUNTER enthält die Anzahl der Datensätze nach Auswertung des WHERE-Kriteriums. *COUNTER ist nicht zulässig, wenn die WHERE-Klausel fehlt.

Beispiel für FIND UNIQUE siehe Programm FNDUNQ (Reporting Mode).

MULTI-FETCH-Klausel



Anmerkung: Diese Klausel kann nur bei Adabas-Datenbanken benutzt werden.

$\left[\text{MULTI-FETCH} \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \\ \text{OF } \textit{multi-fetch-factor} \end{array} \right\} \right]$

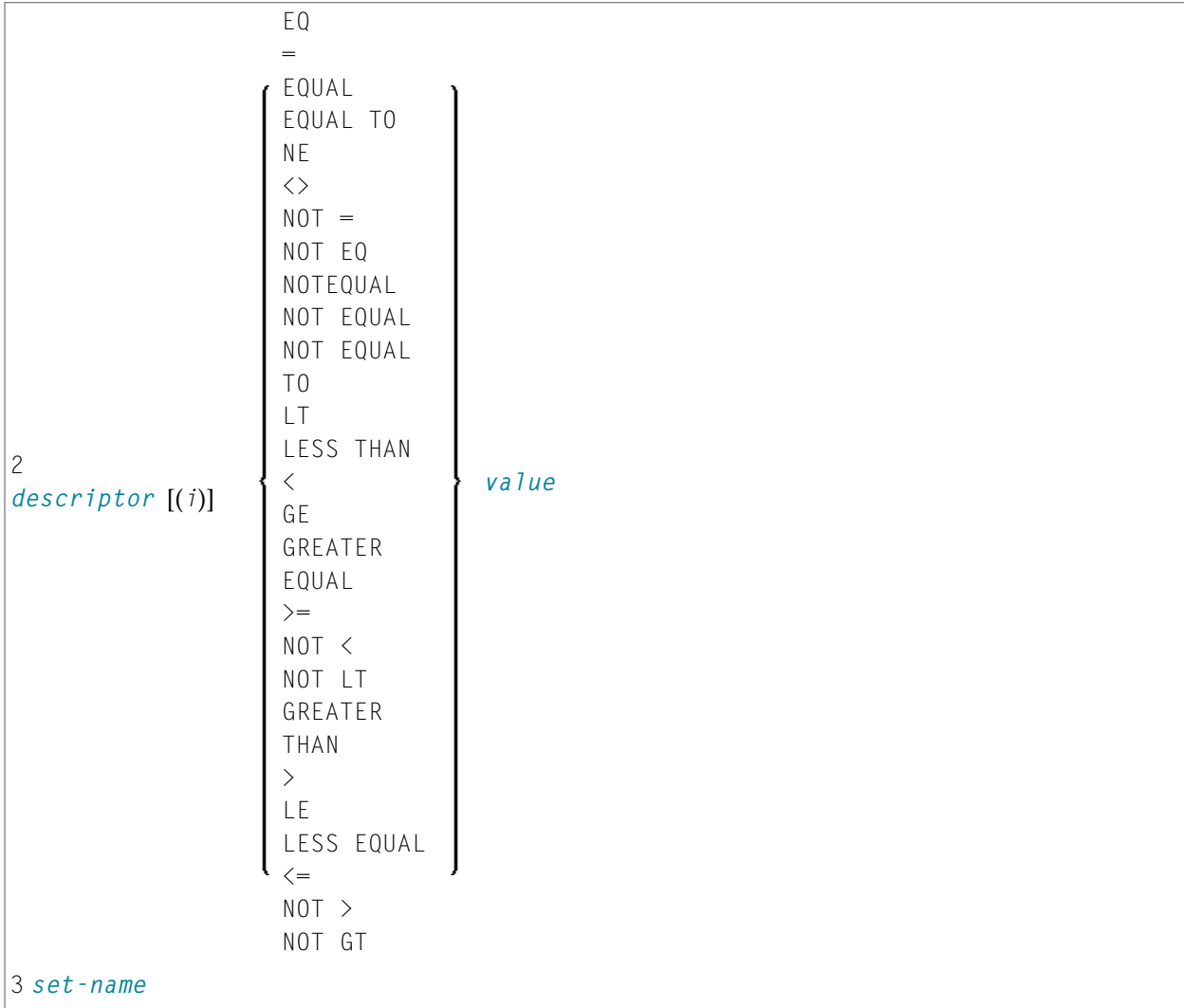


Anmerkung: [MULTI-FETCH OF *multi-fetch-factor*] wird bei den Datenbanktypen ADA und ADA2 nicht ausgewertet. Es erfolgt die Standardverarbeitung (siehe Profilparameter MFSET). Beim Datenbanktyp ADA2 wird die MULTI-FETCH-Klausel komplett ignoriert; siehe *Database Management System Assignments* in der *Configuration Utility*-Dokumentation.

Weitere Informationen siehe *Multi-Fetch-Klausel (Adabas)* im *Leitfaden zur Programmierung*.

Suchkriterium bei Adabas-Dateien (basic-search-criterion)

$1 \textit{ descriptor} [(i)] \left\{ \begin{array}{l} \text{EQ} \\ = \\ \text{EQUAL} \\ \text{EQUAL TO} \end{array} \right\} \textit{value} \left\{ \left[\text{OR} \left\{ \begin{array}{l} \text{EQ} \\ = \\ \text{EQUAL} \\ \text{EQUAL TO} \end{array} \right\} \textit{value} \right] \dots \right\} \left[\text{THRU } \textit{value} [\text{BUT NOT } \textit{value} [\text{THRU } \textit{value}]] \right]$
--



Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>descriptor</i>		S	A		A	U	N	P	I	F	B	D	T	L				nein	nein
<i>value</i>	C	S			A	U	N	P	I	F	B	D	T	L				ja	nein
<i>set-name</i>	C	S			A													nein	nein

Syntax-Element-Beschreibung:

<i>descriptor</i>	<p>Deskriptor-Feld:</p> <p>Es kann ein Adabas-Deskriptor, -Subdeskriptor, -Superdeskriptor, -Hyperdeskriptor oder phonetischer Deskriptor angegeben werden.</p> <p>Es kann auch ein im DDM als Nicht-Deskriptor markiertes Feld angegeben werden.</p>
(i)	<p>Index:</p> <p>Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in einer beliebigen Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muss als Konstante angegeben werden; es darf kein Indexbereich angegeben werden.</p> <p>Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.</p>
<i>value</i>	<p>Suchwert:</p> <p>Der Suchwert, den das Deskriptorfeld haben soll. Die Formate von Suchwert und Deskriptorfeld müssen kompatibel sein.</p>
<i>set-name</i>	<p>Set-Name:</p> <p>Identifiziert einen Set von Datensätzen, die mit einem FIND-Statement ausgewählt wurden, das eine RETAIN-Klausel enthielt. Der referenzierte Set muss von derselben physischen Adabas-Datei ausgewählt worden sein.</p> <p><i>set-name</i> kann entweder als Textkonstante (bis zu 32 Zeichen lang) oder in Form einer alphanumerischen Variablen angegeben werden.</p> <p>Mit Entire System Server kann <i>set-name</i> nicht angegeben werden.</p>

Siehe auch:

- [Beispiel 3 – Basis-Suchkriterium in WITH-Klausel](#)
- [Beispiel 4 – Basis-Suchkriterium mit multiplem Feld](#)

Suchkriterium mit Null-Indikator (*basic-search-criterion*)

$$null-indicator \left\{ \begin{array}{l} = \\ EQ \\ EQUAL \\ [T0] \end{array} \right\} value$$

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition		
<i>null-indicator</i>		S							I										nein	nein
<i>value</i>	C	S				N	P	I	F	B								ja	nein	

Syntax-Element-Beschreibung:

<i>null-indicator</i>	Der Null-Indikator.	
<i>value</i>	Möglicher Wert:	
	-1	Das entsprechende Feld enthält keinen Wert.
	0	Das entsprechende Feld enthält einen Wert.

Verknüpfen von Suchkriterien (für Adabas-Dateien)

Es ist möglich, mehrere Suchkriterien mit den Boole'schen Operatoren AND, OR und NOT miteinander zu verknüpfen. Mit Klammern kann außerdem die Reihenfolge der Kriterienauswertung gesteuert werden. Die Auswertung verknüpfter Suchkriterien geschieht in folgender Reihenfolge:

1. (): Klammern
2. NOT: Negation (nur für *basic-search-criterion* der Form [2]).
3. AND: Und-Verknüpfung
4. OR: Oder-Verknüpfung

Suchkriterien können mit logischen Operatoren verknüpft werden, um einen komplexen Suchausdruck (*search-expression*) zu bilden. Eine solcher komplexer Suchausdruck hat folgende Syntax:

$$[\text{NOT}] \left\{ \begin{array}{l} \textit{basic-search-criterion} \\ \textit{(search-expression)} \end{array} \right\} \left[\left\{ \begin{array}{l} \text{OR} \\ \text{AND} \end{array} \right\} \textit{search-expression} \right] \dots$$

Siehe auch [Beispiel 5 - Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel](#) .

Such-Schlüsselfelder (Deskriptoren) für Adabas-Dateien

Adabas-Benutzer können bei der Suche nach Datensätzen als Suchschlüssel Datenbankfelder verwenden, die als Deskriptoren definiert sind.

Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren

Bei Adabas-Datenbanken können für die Konstruktion von Suchkriterien Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren verwendet werden.

- Ein Subdeskriptor ist ein Schlüsselfeld, das Teil eines Feldes ist.
- Ein Superdeskriptor ist ein Schlüsselfeld, das aus mehreren Feldern oder Teilfeldern besteht.
- Ein Hyperdeskriptor ist ein Schlüsselfeld, das durch einen benutzerdefinierten Algorithmus gebildet wird.
- Ein phonetischer Deskriptor ermöglicht die Suche nach einem Feldwert (z.B. der Name einer Person) anhand des Klanges eines Wertes. Bei der Suche mit einem phonetischen Deskriptor werden alle Werte gefunden, die so ähnlich klingen wie der Suchwert.

Bei welcher Datei welche Felder als Deskriptoren, Sub-, Super-, Hyper- und phonetische Deskriptoren verwendet werden können, ist in dem betreffenden DDM definiert.

Werte für Subdeskriptoren, Superdeskriptoren, phonetische Deskriptoren

Die mit Subdeskriptoren, Superdeskriptoren und phonetischen Deskriptoren angegebenen Suchwerte müssen mit dem jeweiligen internen Format kompatibel sein: ein Subdeskriptor hat dasselbe interne Format wie das Feld, von dem er ein Teil ist; ein phonetischer Deskriptor hat immer alphanumerisches Format; das interne Format eines Superdeskriptors ist binär, falls alle Felder, aus denen er sich zusammensetzt, numerisches Format haben; andernfalls ist sein internes Format alphanumerisch.

Werte für Sub- und Superdeskriptoren können wie folgt angegeben werden:

- als numerische oder hexadezimale Konstanten; hat ein Superdeskriptor binäres Format (vgl. oben), muss eine numerische oder hexadezimale Konstante als Wert angegeben werden;
- als Werte von Benutzervariablen, die mit einem REDEFINE-Statement redefiniert wurden, um die Teile auszuwählen, die den Sub- bzw. Superdeskriptorwert darstellen.

Deskriptoren aus Datenbank-Arrays

Ein Deskriptor, der Teil eines Datenbank-Arrays ist, kann ebenfalls als Suchfeld verwendet werden. Bei Adabas-Dateien kann dies ein multiples Feld sein oder ein Feld, das in einer Periodengruppe enthalten ist.

Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann entweder mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in irgendeiner Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muss als Konstante angegeben werden. Es darf kein Indexbereich angegeben werden.

Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.

Siehe auch [Beispiel 5 - Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel](#).

COUPLED-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-Dateien.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

```

{ AND }
{ OR  } COUPLED  [TO] [FILE] view-name

      [ VIA descriptor1      { EQ
                             =
                             EQUAL
                             EQUAL TO } descriptor2 ]

      [WITH]
      basic-search-criteria
    
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition	
descriptor1	S	A			A	N	P	B						nein	nein
descriptor2	S	A			A	N	P	B						nein	nein

 **Anmerkung:** Ohne VIA-Klausel kann die COUPLED-Klausel bis zu viermal angegeben werden, mit VIA-Klausel bis zu 42-mal.

Adabas bietet die Möglichkeit, Dateien miteinander zu koppeln. Dadurch ist es mit der COUPLED-Klausel möglich, im Suchkriterium eines einzigen FIND-Statements Deskriptoren von verschiedenen Dateien anzugeben.

Zwei verschiedene COUPLED-Klauseln desselben FIND-Statements dürfen nicht dieselbe Adabas-Datei verwenden.

Ein *set-name* (siehe [RETAIN-Klausel](#)) darf nicht im Suchkriterium (*basic-search-criteria*) angegeben werden.

Datenbankfelder der in der COUPLED-Klausel angegebenen Datei können anschließend im Programm nicht referenziert werden, wenn auf diese Datei kein separater FIND- oder READ-Zugriff erfolgt.



Anmerkung: Wenn die COUPLED-Klausel verwendet wird, kann die Haupt-WITH-Klausel gegebenenfalls weggelassen werden. Wenn die Haupt-WITH-Klausel weggelassen wird, dürfen die Schlüsselwörter AND/OR in der COUPLED-Klausel nicht angegeben werden.

Physisches Koppeln ohne VIA-Klausel

Die in der COUPLED-Klausel ohne VIA verwendeten Dateien müssen mit der entsprechenden Adabas-Utility physisch gekoppelte Adabas-Dateien sein (wie in der Adabas-Dokumentation beschrieben).

Siehe auch [Beispiel 7 – Physisch gekoppelte Dateien benutzen](#).

Die Referenzierung von NAME im DISPLAY-Statement ist gültig, da dieses Feld in der Datei EMPLOYEES (Angestellte) enthalten ist; eine Referenzierung von MAKE (Fabrikat) hingegen wäre nicht gültig, da MAKE in der Datei VEHICLES (Fahrzeuge) enthalten ist, welche in der COUPLED-Klausel angegeben wurde.

In diesem Beispiel werden Datensätze nur gefunden, wenn die beiden Dateien EMPLOYEES und VEHICLES physisch gekoppelt sind.

Logisches Koppeln mit VIA-Klausel

Die Option `VIA descriptor1 = descriptor2` erlaubt es Ihnen, in einer Suchabfrage mehrere Adabas-Dateien logisch miteinander zu koppeln, dabei ist:

- *descriptor1* ein Feld aus dem ersten View.
- *descriptor2* ein Feld aus dem zweiten View.

Die beiden Dateien brauchen nicht physisch in Adabas gekoppelt zu sein. Diese COUPLED-Option nutzt die ab Adabas Version 5 gebotene Möglichkeit des Soft Coupling aus, die in der Adabas-Dokumentation beschrieben ist.

Siehe auch [Beispiel 8 – VIA-Klausel](#).

STARTING WITH-Klausel

Diese Klausel gilt nur für Adabas-Datenbanken.

Sie können diese Klausel benutzen, um als *operand5* eine Adabas-ISBN (Internal Sequence Number), die als ein Startwert für die Auswahl von Datensätzen benutzt werden soll.

Diese Klausel kann zum Repositionieren innerhalb einer `FIND`-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden, um auf einfache Weise den nächsten Datensatz zu bestimmen, mit dem die Verarbeitung fortgesetzt werden soll. Dies ist besonders hilfreich, wenn der nächste Datensatz sich nicht eindeutig durch einen seiner Deskriptorwerte ermitteln lässt.



Anmerkung: Als tatsächlicher Startwert wird nicht der Wert von *operand5*, sondern der nächsthöhere Wert genommen.

Beispiel:

Siehe Programm `FNDSISN` in Library `SYSEXSYN`.

SORTED BY-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-, Tamino- und SQL-Datenbanken.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

```
SORTED [BY] descriptor ... 3 [DESCENDING]
```

Die `SORTED BY`-Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren. Diese Deskriptoren brauchen nicht mit den als Suchkriterium verwendeten Deskriptoren identisch zu sein.

Normalerweise wird in *aufsteigender* Reihenfolge der Werte sortiert; wünschen Sie eine *absteigende* Sortierfolge, geben Sie das Schlüsselwort `DESCENDING` an. Der Sortiervorgang verwendet die Adabas-Invertierten-Listen, es werden hierbei keine Datensätze gelesen.



Anmerkung: Diese Klausel kann beträchtliche zusätzliche Verarbeitungszeit beanspruchen, falls das zur Steuerung der Sortierfolge verwendete Deskriptorfeld viele Werte enthält. Das liegt daran, dass die gesamte Wertliste abgesucht werden muss, bis alle ausgewählten Datensätze lokalisiert worden sind. Wollen Sie eine große Zahl von Datensätzen sortieren, sollten Sie daher stattdessen das Statement `SORT` verwenden.

Bei der Verwendung einer `SORTED BY`-Klausel gelten die Adabas-Sortierlimits (siehe `ADARUN`-Parameter `LS` in der Adabas-Dokumentation).

In einer `SORTED BY`-Klausel darf kein Deskriptor, der Teil einer Periodengruppe ist, angegeben werden; ein multiples Feld (ohne Index) darf angegeben werden.

Auf Großrechnern dürfen in einer SORTED BY-Klausel keine Nicht-Deskriptoren angegeben werden.

Eine SORTED BY-Klausel darf nicht zusammen mit einer RETAIN-Klausel verwendet werden.

Siehe auch [Beispiel 9 – SORTED BY-Klausel](#).

Hinweis zur gemeinsamen Verwendung der Klauseln STARTING WITH und SORTED BY

Wenn sowohl die STARTING WITH- als auch die SORTED BY-Klausel im selben FIND-Statement verwendet werden und die darunterliegende Datenbank Adabas ist, ist Folgendes zu beachten:

Bei Adabas für Großrechner

Bei Adabas für Großrechner wird das FIND-Statement in den folgenden Schritten ausgeführt:

1. Alle Datensätze, auf die das Suchkriterium zutrifft, werden gesammelt und in die ISN-Reihenfolge gebracht.
2. Die Datensätze werden nach dem in der SORTED BY-Klausel angegebenen Deskriptor sortiert.
3. Der Datensatz, dessen ISN-Wert in der STARTING WITH-Klausel angegeben ist, wird in die nach Deskriptor sortierte Datensatzliste platziert.
4. Die auf Datensätze, die nach dem unter Schritt 3 gefundenen Datensatz kommen, werden in der FIND-Schleife zurückgegeben.

Bei Adabas für OpenSystems

Bei Adabas für OpenSystems (UNIX, Windows, OpenVMS) wird dasselbe Statement wie folgt ausgeführt:

1. Alle Datensätze, auf die das Suchkriterium zutrifft, werden gesammelt und in die ISN-Reihenfolge gebracht.
2. Der Datensatz, dessen ISN-Wert in der STARTING WITH-Klausel angegeben ist, wird in die nach ISN sortierte Datensatzliste platziert.
3. Die Datensätze, die nach dem unter Schritt 2 gefundenen Datensatz kommen, werden nach dem in der SORTED BY-Klausel angegebenen Deskriptor sortiert und in der FIND-Schleife zurückgegeben.

Beispiel:

FIND

Wenn man das folgende Programm mit Adabas Version 8 für Großrechner und Adabas Version 6.1 für UNIX/OpenVMS/Windows ausführt,

```
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #ISN (I4)
END-DEFINE
FORMAT NL=5 SG=OFF PS=43 AL=15
*
PRINT 'FIND' (I)
FIND V1 WITH NAME = 'B' THRU 'BALBIN'
  RETAIN AS 'SET1'
  IF *COUNTER = 4 THEN
    #ISN := *ISN
  END-IF
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
  ' .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
END
```

erhält man folgendes Ergebnis:

Ergebnis bei Natural für Großrechner (Adabas Version 8)				Ergebnis bei Natural für OpenSystems (Adabas Version 6.1)			
ISN	NAME	FIRST-NAME		ISN	NAME	FIRST-NAME	
CITY				CITY			
-----				-----			
-----				-----			
FIND V1 WITH NAME = 'B' THRU 'BALBIN'				FIND V1 WITH NAME = 'B' THRU 'BALBIN'			
12	BAILLET	PATRICK	LYS	12	BAILLET	PATRICK	
	LEZ LANNOY				LYS LEZ LANNOY		
58	BAGAZJA	MARJAN		58	BAGAZJA	MARJAN	
	MONTHERME				MONTHERME		
351	BAECKER	JOHANNES		351	BAECKER	JOHANNES	
	FRANKFURT				FRANKFURT		
355	BAECKER	KARL		355	BAECKER	KARL	
	SINDELFINGEN				SINDELFINGEN		
370	BACHMANN	HANS		370	BACHMANN	HANS	
	MUENCHEN				MUENCHEN		
490	BALBIN	ENRIQUE		490	BALBIN	ENRIQUE	
	BARCELONA				BARCELONA		
650	BAKER	SYLVIA	OAK	650	BAKER	SYLVIA	
	BROOK				OAK BROOK		
913	BAKER	PAULINE		913	BAKER	PAULINE	
	DERBY				DERBY		
FIND .. SORTED BY NAME				FIND .. SORTED BY NAME			
	370 BACHMANN	HANS			370 BACHMANN	HANS	
	MUENCHEN				MUENCHEN		
	351 BAECKER	JOHANNES			351 BAECKER	JOHANNES	
	FRANKFURT				FRANKFURT		
	355 BAECKER	KARL			355 BAECKER	KARL	
	SINDELFINGEN				SINDELFINGEN		
	58 BAGAZJA	MARJAN			58 BAGAZJA	MARJAN	
	MONTHERME				MONTHERME		
	12 BAILLET	PATRICK	LYS		12 BAILLET	PATRICK	
	LEZ LANNOY				LYS LEZ LANNOY		
	650 BAKER	SYLVIA	OAK		650 BAKER	SYLVIA	
	BROOK				OAK BROOK		
	913 BAKER	PAULINE			913 BAKER	PAULINE	
	DERBY				DERBY		
	490 BALBIN	ENRIQUE			490 BALBIN	ENRIQUE	
	BARCELONA				BARCELONA		
FIND .. STARTING WITH ISN = 355							

Ergebnis bei Natural für Großrechner (Adabas Version 8)	Ergebnis bei Natural für OpenSystems (Adabas Version 6.1)
370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME 58 BAGAZJA MARJAN MONTHERME 12 BAILLET PATRICK LYS LEZ LANNOY 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA	FIND .. STARTING WITH ISN = 355 370 BACHMANN HANS MUENCHEN 490 BALBIN ENRIQUE BARCELONA 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME 370 BACHMANN HANS MUENCHEN 650 BAKER SYLVIA OAK BROOK 913 BAKER PAULINE DERBY 490 BALBIN ENRIQUE BARCELONA

Ein FIND-Statement mit höchstens einer dieser Optionen (SORTED BY oder STARTING WITH ISN) liefert immer dieselben Datensätze in derselben Reihenfolge, egal unter welchem System das Statement ausgeführt wird. Werden jedoch die beiden Klauseln zusammen benutzt, ist das Ergebnis davon abhängig, auf welcher Plattform das Datenbank-Statement von Adabas bedient wird.

Wenn Sie beabsichtigen, ein solches Natural-Programm auf mehreren Plattformen einzusetzen, sollten Sie deshalb die Kombination der Klauseln SORTED BY und STARTING WITH ISN im selben FIND-Statement vermeiden.

RETAIN-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-Datenbanken.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

```
RETAIN AS operand6
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand6</i>	C S	A	ja	nein

Syntax-Element-Beschreibung:

RETAIN AS	<p>Suchergebnis für weitere Verarbeitung bereit stellen:</p> <p>Mit der RETAIN-Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen.</p> <p>Die ausgewählten Datensätze werden als ISN-Set in die Adabas-Arbeitsdatei gestellt. Dieser Set kann in nachfolgenden FIND-Statements als Suchkriterium (<i>basic-search-criterion</i>) angegeben werden, um aus dem Set eine gezieltere Auswahl von Datensätzen für die weitere Verarbeitung zu treffen.</p> <p>Der Set ist dateispezifisch und kann nur von einem anderen FIND-Statement verwendet werden, das dieselbe Datei verarbeitet. Der Set kann von einem beliebigen anderen Natural-Programm referenziert werden.</p>
<i>operand6</i>	<p>Set-Name:</p> <p>Der Set-Name identifiziert den ISN-Set. Der Name kann als alphanumerische Konstante oder in Form einer alphanumerischen Benutzervariablen angegeben werden. Es wird nicht geprüft, ob ein Set-Name bereits vergeben ist; wird ein Set-Name zweimal vergeben, überschreibt der neue Set den alten.</p>

Siehe auch [Beispiel 10 - RETAIN-Klausel](#).

Freigabe von Sets

Die Anzahl der Sets, die zurückgestellt werden können, ist nicht begrenzt; die Anzahl der ISNs pro Set auch nicht. Die zu einer gegebenen Zeit benötigte Mindestanzahl von ISN-Sets sollte definiert werden. Nicht länger benötigte Sets sollten mit einem RELEASE SETS-Statement aus der Arbeitsdatei gelöscht werden.

Wenn sie nicht von einem RELEASE-Statement freigegeben werden, bleiben erstellte ISN-Sets während der gesamten Natural-Session erhalten und werden nicht automatisch gelöscht, außer bei einem Library-Wechsel. Ein mit einem Programm erstellter Set kann von einem anderen Programm referenziert und verarbeitet oder unter Angabe zusätzlicher Suchkriterien weiter selektiert werden.

Zugriffe anderer Benutzer

Die Datensätze, deren ISNs in einem ISN-Set enthalten sind, sind nicht vor Zugriff/Veränderung durch andere Benutzer geschützt. Bevor Sie Datensätze aus dem Set verarbeiten, empfiehlt es sich daher sicherzustellen, dass das ursprüngliche Selektionskriterium, mit dem der Set erstellt wurde, immer noch auf die ausgewählten Datensätze zutrifft.

Dies geschieht mit einem neuen FIND-Statement, bei dem man den Set-Namen in der WITH-Klausel als primäres Suchkriterium angibt und in einer WHERE-Klausel das ursprüngliche Primärsuchkriterium (d.h. das Suchkriterium aus der WITH-Klausel des FIND-Statements, mittels dessen der Set erstellt wurde).

Einschränkung

Eine RETAIN-Klausel darf nicht zusammen mit einer SORTED BY-Klausel verwendet werden.

WHERE-Klausel

```
WHERE logical-condition
```

Die WHERE-Klausel dient dazu, ein zusätzliches Selektionskriterium (*logical-condition*) anzugeben, das ausgewertet wird, *nachdem* ein über die WITH-Klausel ausgewählter Datensatz gelesen wurde und *bevor* ein ausgewählter Datensatz weiter verarbeitet wird (einschließlich AT BREAK-Auswertung).

Die für logische Bedingungen gültige Syntax ist im Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung* erklärt.

Ist für das FIND-Statement die Anzahl der zu verarbeitenden Datensätze durch ein Limit begrenzt, so werden Datensätze, die aufgrund einer WHERE-Klausel *nicht* weiterverarbeitet werden, bei der Ermittlung des Limits nicht mitgezählt. Sie werden allerdings bei der Ermittlung eines auf allgemeinerer Ebene gesetzten Limits (Session-Parameter LT, GLOBALS-Kommando oder LIMIT-Statement) mitgezählt.

Siehe auch [Beispiel 11 - WHERE-Klausel](#).

IF NO RECORDS FOUND-Klausel

Structured Mode-Syntax

```
IF NO [RECORDS] [FOUND]
    { ENTER
      statement ... }
END-NOREC
```

Reporting Mode-Syntax

```
IF NO [RECORDS] [FOUND]
    { ENTER
      statement
      DO statement ... DOEND }
```

In der `IF NO RECORDS FOUND`-Klausel können Sie eine Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die in der `WITH`- und `WHERE`-Klausel des `FIND`-Statements angegebenen Selektionskriterien erfüllt.

Ist dies der Fall, dann löst die `IF NO RECORDS FOUND`-Klausel eine Verarbeitungsschleife aus, die einmal mit einem "leeren" Datensatz durchlaufen wird. Falls Sie dies nicht wünschen, geben Sie in der `IF NO RECORDS FOUND`-Klausel das Statement `ESCAPE BOTTOM` an.

Enthält die `IF NO RECORDS FOUND`-Klausel ein oder mehrere Statements, werden diese ausgeführt, unmittelbar bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine weiteren Statements ausgeführt werden, muss die `IF NO RECORDS FOUND`-Klausel das Schlüsselwort `ENTER` enthalten.

Siehe auch [Beispiel 12 - IF NO RECORDS FOUND-Klausel](#).

Datenbankwerte

Natural setzt alle Datenbankfelder, die die in der aktuellen Verarbeitungsschleife angegebene Datei referenzieren, auf Leerwerte, es sei denn, eines der in der `IF NO RECORDS FOUND`-Klausel angegebenen Statements weist den Feldern andere Werte zu.

Auswertung von Systemfunktionen

Natural-Systemfunktionen werden einmal für den leeren Datensatz ausgewertet, der für die aus der `IF NO RECORDS FOUND`-Klausel resultierende Verarbeitung erstellt wurde.

Einschränkung

Bei `FIND FIRST`, `FIND NUMBER` und `FIND UNIQUE` kann diese Klausel nicht verwendet werden.

Beispiele

- Beispiel 1 — PASSWORD-Klausel
- Beispiel 2 — CIPHER-Klausel
- Beispiel 3 — Basis-Suchkriterium in WITH-Klausel
- Beispiel 4 — Basis-Suchkriterium mit multiplem Feld
- Beispiel 5 — Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel
- Beispiel 6 — Mehrere Beispiele für die Benutzung von Datenbank-Arrays
- Beispiel 7 — Physisch gekoppelte Dateien benutzen
- Beispiel 8 — VIA-Klausel
- Beispiel 9 — SORTED BY-Klausel
- Beispiel 10 — RETAIN-Klausel
- Beispiel 11 — WHERE-Klausel
- Beispiel 12 — IF NO RECORDS FOUND-Klausel
- Beispiel 13 — Systemvariablen mit dem FIND-Statement benutzen
- Beispiel 14 — Mehrere FIND-Statements

Siehe auch das Beispiel für FIND NUMBER: Programm [FNDNUM](#).

Beispiel 1 — PASSWORD-Klausel

```
** Example 'FNDPWD': FIND (with PASSWORD clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASSWORD (AD=N)
LIMIT 2
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
      WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END
```

Ausgabe des Programms FNDPWD:

```
ENTER PASSWORD FOR EMPLOYEE FILE:
```

Beispiel 2 — CIPHER-Klausel

```
** Example 'FNDCIP': FIND (with PASSWORD/CIPHER clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
1 #CIPHER (N8)
END-DEFINE
*
LIMIT 2
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE: ' #PASSWORD (AD=N)
  / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER (AD=N)
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
      CIPHER = #CIPHER
      WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END
```

Ausgabe des Programms FNDCIP:

```
ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:
```

Beispiel 3 — Basis-Suchkriterium in WITH-Klausel

```
FIND STAFF WITH NAME = 'SMITH'
FIND STAFF WITH CITY NE 'BOSTON'
FIND STAFF WITH BIRTH = 610803
FIND STAFF WITH BIRTH = 610803 THRU 610811
FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'
FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
      BUT NOT 100087 THRU 100095
```

Beispiel 4 — Basis-Suchkriterium mit multiplem Feld

Wenn der im Basis-Suchkriterium benutzte Deskriptor ein multiples Feld ist, können grundsätzlich vier verschiedene Arten von Ergebnissen erzielt werden (das Feld MU-FIELD in den folgenden Beispielen wird als multiples Feld angenommen):

```
FIND XYZ-VIEW WITH MU-FIELD = 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *wenigstens* eine Ausprägung von MU-FIELD den Wert A hat.

```
FIND XYZ-VIEW WITH MU-FIELD NOT EQUAL 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *wenigstens* eine Ausprägung von MU-FIELD *nicht* den Wert A hat.

```
FIND XYZ-VIEW WITH NOT MU-FIELD NOT EQUAL 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *jede* Ausprägung von MU-FIELD den Wert A hat.

```
FIND XYZ-VIEW WITH NOT MU-FIELD = 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *keine* der Ausprägungen von MU-FIELD den Wert A hat.

Beispiel 5 — Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'  
                AND (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'  
                AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'  
                AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'  
                AND BIRTH GT 19360101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
      AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

Beispiel 6 — Mehrere Beispiele für die Benutzung von Datenbank-Arrays

In den folgenden Beispielen wird davon ausgegangen, dass das Feld SALARY (Gehalt) ein in einer Periodengruppe enthaltener Deskriptor und das Feld LANG (Sprache) ein multiples Feld ist.

```
FIND EMPLOYEES WITH SALARY LT 20000
```

Führt zu einer Suche aller Ausprägungen von SALARY.

```
FIND EMPLOYEES WITH SALARY (1) LT 20000
```

Führt zu einer Suche nur nach der ersten Ausprägung.

```
FIND EMPLOYEES WITH SALARY (1:4) LT 20000 /* invalid
```

Eine Bereichsangabe muss nicht für ein als Suchkriterium benutztes Feld innerhalb einer Periodengruppe vorgenommen werden.

```
FIND EMPLOYEES WITH LANG = 'FRENCH'
```

Führt zu einer Suche aller Werte von LANG.

```
FIND EMPLOYEES WITH LANG (1) = 'FRENCH' /* invalid
```

Ein Index darf für ein als Suchkriterium benutztes multiples Feld nicht angegeben werden.

Beispiel 7 — Physisch gekoppelte Dateien benutzen

```
** Example 'FNDCPL': FIND (using coupled files)
** NOTE: Adabas files must be physically coupled when using the
**       COUPLED clause without the VIA clause.
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      AND COUPLED TO
      VEHIC-VIEW WITH MAKE = 'VW'
```

FIND

```
  DISPLAY NOTITLE NAME
END-FIND
*
END
```

Beispiel 8 — VIA-Klausel

```
** Example 'FNDVIA': FIND (with VIA clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
      AND COUPLED TO VEHIC-VIEW
      VIA PERSONNEL-ID = PERSONNEL-ID WITH MAKE = 'VOLVO'
      DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
*
END
```

Ausgabe des Programms FNDVIA:

```
Page      1                                05-01-17  13:18:22

PERSONNEL      NAME      FIRST-NAME
  ID
-----
20011000  ADKINSON      BOB
```

Beispiel 9 — SORTED BY-Klausel

```
** Example 'FNDSOR': FIND (with SORTED BY clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
```



```

*
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      SORTED BY NAME PERSONNEL-ID

  DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
*
END

```

Ausgabe des Programms FNDSOR:

NAME	FIRST-NAME	PERSONNEL ID
BAECKER	JOHANNES	11500345
BECKER	HERMANN	11100311
BERGMANN	HANS	11100301
BLAU	SARAH	11100305
BLOEMER	JOHANNES	11200312
DIEDRICHS	HUBERT	11600301
DOLLINGER	MARGA	11500322
FALTER	CLAUDIA	11300311
	HEIDE	11400311
FREI	REINHILD	11500301

Beispiel 10 — RETAIN-Klausel

```

** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
  RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'

```

FIND

```
    DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

Ausgabe des Programms RELEX1:

NAME	CITY	DATE OF BIRTH
RUBIN	NEW YORK	1945-10-27
WALLACE	NEW YORK	1945-08-04

Beispiel 11 — WHERE-Klausel

```
** Example 'FNDWHE': FIND (with WHERE clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
      WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE
      CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
*
END
```

Ausgabe des Programms FNDWHE:

CITY	CURRENT POSITION	PERSONNEL ID	NAME
PARIS	INGENIEUR COMMERCIAL	50007300	CAHN
PARIS	INGENIEUR COMMERCIAL	50006500	MAZUY
PARIS	INGENIEUR COMMERCIAL	50004700	FAURIE
PARIS	INGENIEUR COMMERCIAL	50004400	VALLY

PARIS	INGENIEUR COMMERCIAL	50002800	BRETON
PARIS	INGENIEUR COMMERCIAL	50001000	GIGLEUX
PARIS	INGENIEUR COMMERCIAL	50000400	KORAB-BRZOZOWSKI

Beispiel 12 — IF NO RECORDS FOUND-Klausel

```

** Example 'FNDIFN': FIND (using IF NO RECORDS FOUND)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)

  IF NO RECORDS FOUND
    MOVE '*** NO CAR ***' TO MAKE
  END-NOREC
/*
  DISPLAY NOTITLE
    NAME (EMP.) (IS=ON)
    FIRST-NAME (EMP.) (IS=ON)
    MAKE (VEH.)
  END-FIND
/*
END-READ
END

```

Ausgabe des Programms FNDIFN:

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
	MARSHA	CHRYSLER
		CHRYSLER
	ROBERT	GENERAL MOTORS
	LILLY	FORD
		MG
	EDWARD	GENERAL MOTORS
	MARTHA	GENERAL MOTORS

	LAUREL	GENERAL MOTORS
	KEVIN	DATSUN
	GREGORY	FORD
JOPER	MANFRED	*** NO CAR ***
JOUSSELIN	DANIEL	RENAULT
JUBE	GABRIEL	*** NO CAR ***
JUNG	ERNST	*** NO CAR ***
JUNKIN	JEREMY	*** NO CAR ***
KAISER	REINER	*** NO CAR ***

Beispiel 13 — Systemvariablen mit dem FIND-Statement benutzen

```

** Example 'FNDVAR': FIND (using *ISN, *NUMBER, *COUNTER)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NOTITLE PERSONNEL-ID NAME
                    *ISN *NUMBER *COUNTER
END-FIND
*
END

```

Ausgabe des Programms FNDVAR:

PERSONNEL ID	NAME	ISN	NMBR	CNT
60000114	DE JUAN	400	41	1
60000136	DE LA MADRID	401	41	2
60000209	PINERO	405	41	3

Beispiel 14 — Mehrere FIND-Statements

In dem folgenden Beispiel werden zuerst alle Mitarbeiter mit Namen SMITH in der Datei EMPLOYEES (Angestellte) ausgewählt. Dann wird die PERSONNEL-ID (Personalnummer) aus der Datei EMPLOYEES als Suchschlüssel für einen Zugriff auf die Datei VEHICLES (Fahrzeuge) benutzt.

```

** Example 'FNDMUL': FIND (with multiple files)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
/*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
  IF NO RECORDS FOUND
    MOVE '*** NO CAR ***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    EMP.NAME (IS=ON)
    EMP.FIRST-NAME (IS=ON)
    VEH.MAKE
  END-FIND
END-FIND
END

```

Ausgabe des Programms FNDMUL:

Der sich ergebende Report zeigt NAME und FIRST-NAME (Vorname) aus der Datei EMPLOYEES für alle Mitarbeiter mit Namen SMITH sowie MAKE (Fabrikat) jedes Autos aus der Datei VEHICLES dieser Mitarbeiter an.

NAME	FIRST-NAME	MAKE
SMITH	GERHARD	ROVER
	SEYMOUR	*** NO CAR ***
	MATILDA	FORD
	ANN	*** NO CAR ***
	TONI	TOYOTA
	MARTIN	*** NO CAR ***

THOMAS	FORD
SUNNY	*** NO CAR ***
MARK	FORD
LOUISE	CHRYSLER
MAXWELL	MERCEDES-BENZ
	MERCEDES-BENZ
ELSA	CHRYSLER
CHARLY	CHRYSLER
LEE	*** NO CAR ***
FRANK	FORD

64 FOR

▪ Funktion	412
▪ Syntax-Beschreibung	413
▪ Beispiel — FOR-Statement	414

FOR <i>operand1</i>	{ [:]= EQ FROM	{ <i>operand2</i> (<i>arithmetic-expression</i>)
	{ TO THRU	{ <i>operand3</i> (<i>arithmetic-expression</i>)
	[STEP	{ <i>operand4</i> (<i>arithmetic-expression</i>)
]
	<i>statement ...</i>	
END-FOR	(<i>structured mode only</i>)	
[LOOP]	(<i>reporting mode only</i>)	



Anmerkung: Aus Kompatibilitätsgründen sind die Schlüsselwörter `:=`, `EQ`, `FROM`, `TO`, `THRU` und `STEP` optional, wenn der entsprechende nachfolgende Operand (*operand2*, *operand3* oder *operand4*) anstelle eines arithmetischen Ausdrucks verwendet wird.

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [REPEAT](#) | [ESCAPE](#)

Gehört zur Funktionsgruppe: [Schleifenverarbeitung](#)

Funktion

Mit dem Statement `FOR` wird eine Verarbeitungsschleife ausgelöst und gleichzeitig die Anzahl der Schleifendurchläufe gesteuert.

Konsistenzprüfung

Bevor die `FOR`-Schleife zum erstenmal durchlaufen wird, wird geprüft, ob die Werte der Operanden konsistent sind (d.h. ob es möglich ist, dass durch wiederholtes Addieren von *operand4* zu *operand2* der Wert von *operand3* erreicht werden kann); ist dies nicht der Fall, wird die `FOR`-Schleife nicht durchlaufen (ohne dass eine Fehlermeldung ausgegeben wird; Ausnahme: wenn der `STEP`-Wert Null ist, wird eine Meldung ausgegeben).

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition						
<i>operand1</i>	S					N	P	I	F													ja	ja
<i>operand2</i>	C	S		N	E	N	P	I	F													ja	nein
<i>operand3</i>	C	S		N	E	N	P	I	F													ja	nein
<i>operand4</i>	C	S		N	E	N	P	I	F													ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Schleifenkontrollvariable (<i>operand1</i>) und Ausgangswert (<i>operand2</i>):</p> <p><i>operand1</i> kann ein Datenbankfeld oder eine Benutzervariable sein und steuert die Anzahl der Schleifendurchläufe. Der nach dem Schlüsselwort FROM angegebene Wert (<i>operand2</i>) wird der Kontrollvariablen als Ausgangswert zugeordnet, bevor die Verarbeitungsschleife das erste Mal durchlaufen wird. Dieser Ausgangswert erhöht <i>operand2</i> wird. sich mit jedem Schleifendurchlauf um den Wert des mit STEP angegebenen <i>operand4</i> (bzw. vermindert sich, wenn <i>operand4</i> einen negativen Wert hat).</p> <p>Die Kontrollvariable kann während der Ausführung der Verarbeitungsschleife referenziert werden, um den aktuellen Wert der Kontrollvariablen zu erhalten.</p>
<i>operand2</i>	
<i>operand3</i>	<p>TO-Wert:</p> <p>Die Verarbeitungsschleife wird beendet, sobald <i>operand1</i> einen Wert enthält, der größer ist als der Wert von <i>operand3</i> (oder kleiner, falls der STEP-Wert negativ ist).</p>
<i>operand4</i>	<p>STEP-Wert:</p> <p>Der Wert von <i>operand4</i> kann positiv oder negativ sein. Ist kein Wert angegeben, wird ein Wert von +1 angenommen.</p> <p>Je nach dem Vorzeichen des STEP-Wertes wird die Vergleichsoperation für <i>operand3</i> auf <i>größer als</i> oder <i>kleiner als</i> gesetzt, wenn die Verarbeitungsschleife zum ersten Mal durchlaufen wird.</p> <p><i>operand4</i> darf nicht Null (0) sein.</p>
(<i>arithmetic-expression</i>)	<p>Anstelle von <i>operand2</i>, <i>operand3</i> oder <i>operand4</i> kann ein beliebiger arithmetischer Ausdruck angegeben werden. Weitere Informationen siehe arithmetic-expression in der Beschreibung des COMPUTE-Statements.</p>

	Anmerkung: Der arithmetische Ausdruck muss in Klammern angegeben werden.
END-FOR	Das für Natural reservierte Wort END-FOR muss zum Beenden des FOR-Statements benutzt werden.

Beispiel — FOR-Statement

```

** Example 'FOREX1S': FOR (structured mode)
*****
DEFINE DATA LOCAL
1 #INDEX (I1)
1 #ROOT (N2.7)
END-DEFINE
*
FOR #INDEX 1 TO 5
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE NOTITLE '=' #INDEX 3X '=' #ROOT
END-FOR
*
SKIP 1
FOR #INDEX 1 TO 5 STEP 2
  COMPUTE #ROOT = SQRT (#INDEX)
  WRITE '=' #INDEX 3X '=' #ROOT
END-FOR
*
END

```

Ausgabe des Programms FOREX1S:

```

#INDEX:    1  #ROOT:    1.0000000
#INDEX:    2  #ROOT:    1.4142135
#INDEX:    3  #ROOT:    1.7320508
#INDEX:    4  #ROOT:    2.0000000
#INDEX:    5  #ROOT:    2.2360679

#INDEX:    1  #ROOT:    1.0000000
#INDEX:    3  #ROOT:    1.7320508
#INDEX:    5  #ROOT:    2.2360679

```

Äquivalentes Reporting-Mode-Beispiel: [FOREX1R](#).

65

FORMAT

- Funktion 416
- Syntax-Beschreibung 417
- Parameter 417
- Beispiel 418

```
FORMAT [(rep)] parameter ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET SAME | GET TRANSACTION | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `FORMAT` dient dazu, Werte für Eingabe- und Ausgabeparameter festzusetzen.

Die Gültigkeit der mit einem `FORMAT`-Statement festgesetzten Werte hat zur Kompilierungszeit Vorrang vor den auf Session-Ebene mit einem `GLOBALS`-Kommando, `SET GLOBALS`-Statement oder vom Natural-Administrator gesetzten Parameterwerten.

Die mit dem `FORMAT`-Statement festgesetzten Werte können ihrerseits auf Statement- oder Elementebene (Feldebene) von den mit den Statements `DISPLAY`, `INPUT`, `PRINT`, `WRITE`, `WRITE TITLE` oder `WRITE TRAILER` gesetzten Parameterwerten überschrieben werden.

Die Einstellungen gelten Sie bis zum Ende des betreffenden Programms, oder bis sie mit einem weiteren `FORMAT`-Statement geändert werden.

Das `FORMAT`-Statement generiert keinen ausführbaren Code im Natural-Programm. Seine Ausführung hängt nicht vom logischen Ablauf des Programms ab. Es wird während der Kompilierung ausgewertet, um die Parameter für die Kompilierung der betroffenen `DISPLAY`-, `WRITE`-, `PRINT`- und `INPUT`-Statements zu setzen. Das `FORMAT`-Statement wirkt sich auf alle nachfolgenden `DISPLAY`-, `WRITE`-, `PRINT`- und `INPUT`-Statements aus.

Syntax-Beschreibung

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das FORMAT-Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls nichts anderes angegeben wird, bezieht sich das FORMAT-Statement auf den ersten Report (Report 0).</p> <p>Informationen zum Steuern des Formats eines mit Natural erzeugten Ausgabe-Reports siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
<i>parameter</i>	<p>Parameter:</p> <p>Die Parameter können in beliebiger Reihenfolge angegeben werden und müssen jeweils durch ein oder mehr Leerzeichen voneinander getrennt werden. Der Eintrag für einen Parameter darf nicht über das Ende einer Sourcecode-Zeile hinausgehen.</p> <p>Die hier gültigen feldsensitiven Parameter-Einstellungen kommen nur für Variablenfelder in Betracht, die in einem INPUT-, WRITE-, DISPLAY- oder PRINT-Statement des ausgewählten Reports verwendet werden. Sie haben aber keine Auswirkung auf in einem der erwähnten Statements verwendete Text-Konstanten.</p> <p>Siehe auch <i>Parameter</i> weiter unten.</p>

Parameter

Die Beschreibungen der Parameter, die Sie beim FORMAT-Statement verwenden können, finden Sie in der *Parameter-Referenz*.

Parameter	Beschreibung
AD	Attribute Definition
AL	Alphanumeric Length for Output
CD	Color Definition
DF	Date Format
DL	Display Length for Output
EM	Edit Mask
ES	Empty Line Suppression
FC	Filler Character

Parameter	Beschreibung
FL	Floating Point Mantissa Length
GC	Filler Character for Group Heading
HC	Header Centering
HW	Heading Width
IC	Insertion Character
IP	Input Prompting Text
IS	Identical Suppress
KD	Key Definition
LC	Leading Characters
LS	Line Size
MC	Multiple-Value Field Count (Kann nur im Reporting Mode verwendet werden.)
MP	Maximum Number of Pages of a Report
MS	Manual Skip
NL	Numeric Length for Output
PC	Periodic Group Count (Kann nur im Reporting Mode verwendet werden.)
PM	Print Mode
PS	Page Size
SF	Spacing Factor
SG	Sign Position
TC	Trailing Characters
UC	Underlining Character
ZP	Zero Printing

Siehe auch *Unterstreichungszeichen für Überschriften – der UC-Parameter im Leitfaden zur Programmierung.*

Beispiel

```

** Example 'FMTEX1': FORMAT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 POST-CODE
  2 COUNTRY
END-DEFINE
*
    
```

```

FORMAT AL=7      /* Alpha-numeric field output length
      FC=+       /* Filler character for field header
      GC=*       /* Filler character for group header
      HC=L       /* Header left justified
      IC=<<      /* Insert characters
      IS=ON      /* Identical suppress on
      TC=>>     /* Trailing character
      UC==      /* Underline character
      ZP=OFF    /* Zero print off

```

```

*

```

```

LIMIT 5
READ EMPLOY-VIEW BY NAME
  DISPLAY NOTITLE
    NAME 3X CITY 3X POST-CODE 3X COUNTRY

```

```

END-READ

```

```

*

```

```

END

```

Ausgabe des Programms FMTEX1:

```

NAME+++++++  CITY+++++++  POSTAL+++++  COUNTRY+++++
            ADDRESS+++++
=====
<<ABELLAN>>  <<MADRID >>  <<28014 >>  <<E >>
<<ACHIESO>>  <<DERBY >>  <<DE3 4TR>> <<UK >>
<<ADAM >>    <<JOIGNY >>  <<89300 >>  <<F >>
<<ADKINSO>>  <<BROOKLY>>  <<11201 >>  <<USA>>
                <<BEVERLE>> <<90211 >>

```


66 GET

▪ Funktion	422
▪ Einschränkungen	423
▪ Syntax-Beschreibung	423
▪ Beispiel	424

Im Structured Mode sowie im Reporting Mode mit einem `DEFINE DATA LOCAL`-Statement gilt die folgende Syntax:

```
GET  [IN] [FILE] view-name
     [PASSWORD=operand1]
     [CIPHER=operand2]
     [ { [RECORD] } ] [ { operand3 } ]
     [ { [RECORDS] } ] [ { *ISN [(r)] } ]
```

Im Reporting Mode ohne `DEFINE DATA LOCAL`-Statement gilt die folgende Syntax:

```
GET  [IN] [FILE] dsm-name
     [PASSWORD=operand1]
     [CIPHER=operand2]
     [ { [RECORD] } ] [ { operand3 } ] operand4
     [ { [RECORDS] } ] [ { *ISN [(r)] } ] ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: `ACCEPT/REJECT` | `AT BREAK` | `AT START OF DATA` | `AT END OF DATA` | `BACKOUT TRANSACTION` | `BEFORE BREAK PROCESSING` | `DELETE` | `END TRANSACTION` | `FIND` | `GET SAME` | `GET TRANSACTION` | `HISTOGRAM` | `LIMIT` | `PASSW` | `PERFORM BREAK PROCESSING` | `READ` | `RETRY` | `STORE` | `UPDATE`

Gehört zur Funktionsgruppe: [Datenbankzugriffe und Datenbankänderungen](#)

Funktion

Das Statement `GET` dient dazu, einen Datensatz mit einer bestimmten Adabas-ISN (Interne Satz-Nummer) zu lesen.

Bei XML-Datenbanken wird das `GET`-Statement verwendet, um ein XML-Objekt mit bekannter Objekt-ID zu lesen.

Das `GET`-Statement löst keine Verarbeitungsschleife aus.

Einschränkungen

- Das GET-Statement kann nicht für SQL-Datenbanken verwendet werden.
- Mit Entire System Server kann das GET-Statement nicht verwendet werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	C	S				A												ja	nein
<i>operand2</i>	C	S				N												nein	nein
<i>operand3</i>	C	S			N	N	P	I	B*									ja	nein
<i>operand4</i>		S	A			A	N	P	I	F	B	D	T	L				ja	ja

* Format B von *operand3* kann nur mit einer Länge von kleiner als oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

<i>view-name</i>	<p>View-Name:</p> <p>Im Structured Mode sowie im Reporting Mode mit einem DEFINE DATA LOCAL-Statement ist <i>view-name</i> der Name eines Views, der entweder direkt in einem DEFINE DATA-Statement oder in einer separaten Global oder Local Data Area definiert ist.</p>
<i>ddm-name</i>	<p>DDM-Name:</p> <p>Im Reporting Mode ohne DEFINE DATA LOCAL-Statement ist <i>ddm-name</i> der Name eines DDMs.</p>
PASSWORD= <i>operand1</i>	<p>PASSWORD-Klausel/CYPHER-Klausel:</p> <p>Diese beiden Klauseln sind nur auf Adabas-Datenbanken anwendbar.</p> <p>Die PASSWORD-Klausel dient dazu, ein Passwort anzugeben, um auf Daten einer passwort-geschützten Adabas-Datei zugreifen zu können.</p> <p>Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Adabas-Daten in entschlüsselter Form zu erhalten.</p> <p>Näheres hierzu siehe FIND- und PASSW-Statement.</p>
CIPHER= <i>operand2</i>	

*ISN / operand3	<p>Interne Sequenz-Nummer:</p> <p>Die ISN kann als numerische Konstante oder Benutzervariable (<i>operand3</i>) oder über die Natural-Systemvariable *ISN angegeben werden.</p>
(r)	<p>Statement-Referenz:</p> <p>Die Notation (<i>r</i>) wird benutzt, um das Statement anzugeben, welches das zum ersten Lesen des Datensatzes benutzte FIND- oder READ-Statement enthält.</p> <p>Wenn (<i>r</i>) nicht angegeben wird, bezieht sich das GET-Statement auf die innerste aktive Verarbeitungsschleife.</p> <p>(<i>r</i>) kann als Statement-Label oder Sourcecode- Zeilennummer angegeben werden.</p>
operand4	<p>Referenz auf Datenbank-Felder:</p> <p>Eine spätere Referenzierung von Datenbankfeldern, die mit einem GET-Statement gelesen wurden, kann entweder unter Angabe eines Statement-Labels oder über die Sourcecode-Zeilenummer des GET-Statements erfolgen.</p>

Beispiel

```

** Example 'GETEX1': GET
*****
DEFINE DATA LOCAL
1 PERSONS VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 SALARY-INFO VIEW OF EMPLOYEES
  2 NAME
  2 CURR-CODE (1:1)
  2 SALARY    (1:1)
*
1 #ISN-ARRAY (B4/1:10)
1 #LINE-NR   (N2)
END-DEFINE
*
FORMAT PS=16
LIMIT 10
READ PERSONS BY NAME
  MOVE *COUNTER TO #LINE-NR
  MOVE *ISN     TO #ISN-ARRAY (#LINE-NR)
  DISPLAY #LINE-NR PERSONNEL-ID NAME FIRST-NAME
  /*
  AT END OF PAGE
  INPUT / 'PLEASE SELECT LINE-NR FOR SALARY INFORMATION:' #LINE-NR
  IF #LINE-NR = 1 THRU 10

```

```

GET SALARY-INFO #ISN-ARRAY (#LINE-NR)
WRITE / SALARY-INFO.NAME
      SALARY-INFO.SALARY (1)
      SALARY-INFO.CURR-CODE (1)
END-IF
END-ENDPAGE
/*
END-READ
END

```

Ausgabe des Programms GETEX1:

Page 1 05-01-13 13:17:42

#LINE-NR	PERSONNEL ID	NAME	FIRST-NAME
1	60008339	ABELLAN	KEPA
2	30000231	ACHIESON	ROBERT
3	50005800	ADAM	SIMONE
4	20008800	ADKINSON	JEFF
5	20009800	ADKINSON	PHYLLIS
6	20012700	ADKINSON	HAZEL
7	20013800	ADKINSON	DAVID
8	20019600	ADKINSON	CHARLIE
9	20008600	ADKINSON	MARTHA
10	20005700	ADKINSON	TIMMIE

PLEASE SELECT LINE-NR FOR SALARY INFORMATION: 1

ABELLAN 1450000 PTA

67 GET SAME

▪ Funktion	428
▪ Einschränkungen	428
▪ Syntax-Beschreibung	429
▪ Beispiel	429

Structured Mode-Syntax

```
GET SAME [(r)]
```

Reporting Mode-Syntax

```
GET SAME [(r)] [operand1 ...]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `ACCEPT/REJECT` | `AT BREAK` | `AT START OF DATA` | `AT END OF DATA` | `BACKOUT TRANSACTION` | `BEFORE BREAK PROCESSING` | `DELETE` | `END TRANSACTION` | `FIND` | `GET` | `GET TRANSACTION DATA` | `HISTOGRAM` | `LIMIT` | `PASSW` | `PERFORM BREAK PROCESSING` | `READ` | `RETRY` | `STORE` | `UPDATE`

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement `GET SAME` dient dazu, einen Datensatz, der gerade verarbeitet wird, erneut zu lesen. Das Statement wird in der Regel dazu verwendet, Werte von Datenbank-Arrays (Periodengruppen oder multiplen Feldern) zu erhalten, falls die Nummer(n) und der Bereich der vorhandenen bzw. gewünschten Ausprägung(en) nicht bekannt war, als der Datensatz zum erstenmal gelesen wurde.

Einschränkungen

- Das Statement `GET SAME` ist nur beim Zugriff auf Adabas-Datenbanken gültig.
- Mit Entire System Server ist dieses Statement nicht verfügbar.
- Bei einem `UPDATE`- oder `DELETE`-Statement darf keine Referenzierung auf ein `GET SAME`-Statement erfolgen; vielmehr sollten diese Statements das `FIND`-, `READ`- oder `GET`-Statement referenzieren, mit dem der betreffende Datensatz ursprünglich gelesen wurde.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A	A U N P B	nein	ja

Syntax-Element-Beschreibung:

(<i>r</i>)	<p>Statement-Referenz:</p> <p>Die Notation (<i>r</i>) wird benutzt, um das Statement anzugeben, das das FIND- oder READ-Statement enthält, mit dem der Datensatz zum erstenmal gelesen wurde.</p> <p>Falls keine Referenzierung erfolgt, bezieht sich das GET SAME-Statement auf die innerste aktive Verarbeitungsschleife.</p> <p>(<i>r</i>) kann als ein Statement-Label oder eine Sourcecode- Zeilennummer angegeben werden.</p>
<i>operand1</i>	<p>Angabe der Felder:</p> <p>Als <i>operand1</i> geben Sie das Feld bzw. die Felder an, deren Werte Sie mit dem GET SAME-Statement erhalten wollen.</p> <p>Anmerkung: <i>operand1</i> kann nicht angegeben werden, wenn das Feld in einem DEFINE DATA-Statement definiert ist.</p>

Beispiel

```

** Example 'GSAEX1': GET SAME
*****
DEFINE DATA LOCAL
1 I          (P3)
1 POST-ADDRESS VIEW OF EMPLOYEES
  2 FIRST-NAME
  2 NAME
  2 ADDRESS-LINE (I:I)
  2 C*ADDRESS-LINE
  2 POST-CODE
  2 CITY
*
1 #NAME          (A30)

```

GET SAME

```
END-DEFINE
*
FORMAT PS=20
MOVE 1 TO I
*
READ (10) POST-ADDRESS BY NAME
  COMPRESS NAME FIRST-NAME INTO #NAME WITH DELIMITER ','
  WRITE // 12T #NAME
  WRITE / 12T ADDRESS-LINE (I.1)
  /*
  IF C*ADDRESS-LINE > 1
    FOR I = 2 TO C*ADDRESS-LINE
      GET SAME /* READ NEXT OCCURRENCE
      WRITE 12T ADDRESS-LINE (I.1)
    END-FOR
  END-IF
  WRITE / POST-CODE CITY
  SKIP 3
END-READ
END
```

Ausgabe des Programms GSAEX1:

```
Page      1                                05-01-13  13:23:36

          ABELLAN,KEPA
          CASTELAN 23-C
28014     MADRID

          ACHIESON,ROBERT
          144 ALLESTREE LANE
          DERBY
          DERBYSHIRE
DE3 4TR   DERBY
```

68

GET TRANSACTION DATA

▪ Funktion	432
▪ Einschränkung	433
▪ Syntax-Beschreibung	433
▪ Beispiel	433

```
GET TRANSACTION [DATA] operand1 ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement `GET TRANSACTION DATA` dient dazu, die Transaktionsdaten, die mit einem vorherigen `END TRANSACTION`-Statement gespeichert wurden, zu lesen.

`GET TRANSACTION DATA` erzeugt keine Verarbeitungsschleife.

Systemvariable *ETID

Um die von der Datenbank zu lesenden Transaktionsdaten zu identifizieren, kann die Natural-Systemvariable *ETID eingesetzt werden.

Keine Transaktionsdaten gespeichert

Werden bei der Ausführung des `GET TRANSACTION DATA`-Statements keine Transaktionsdaten gefunden, werden alle mit dem Statement angegebenen Felder mit Leerzeichen gefüllt, gleichgültig welches Format die Felder haben.



Vorsicht: Achten Sie darauf, dass keine arithmetischen Operationen auf der Grundlage „leerer“ Transaktionsdaten ausgeführt werden, da dies einen Programmabbruch zur Folge hätte.

Einschränkung

Das GET TRANSACTION DATA-Statement gilt nur für Transaktionen auf Adabas-Datenbanken.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S	A U N P I F B D T	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Angabe der Felder:</p> <p>Reihenfolge, Länge und Format der mit GET TRANSACTION DATA zu lesenden Felder muss mit Reihenfolge, Länge und Format der mit dem jeweiligen END TRANSACTION-Statement angegebenen Felder übereinstimmen.</p>
-----------------	--

Beispiel

```

** Example 'GTREX1': GET TRANSACTION
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 MIDDLE-I
  2 CITY
*
1 #PERS-NR (A8) INIT <' '>
END-DEFINE
*
GET TRANSACTION DATA #PERS-NR
IF #PERS-NR NE ' '
  WRITE 'LAST TRANSACTION PROCESSED FROM PREVIOUS SESSION' #PERS-NR

```

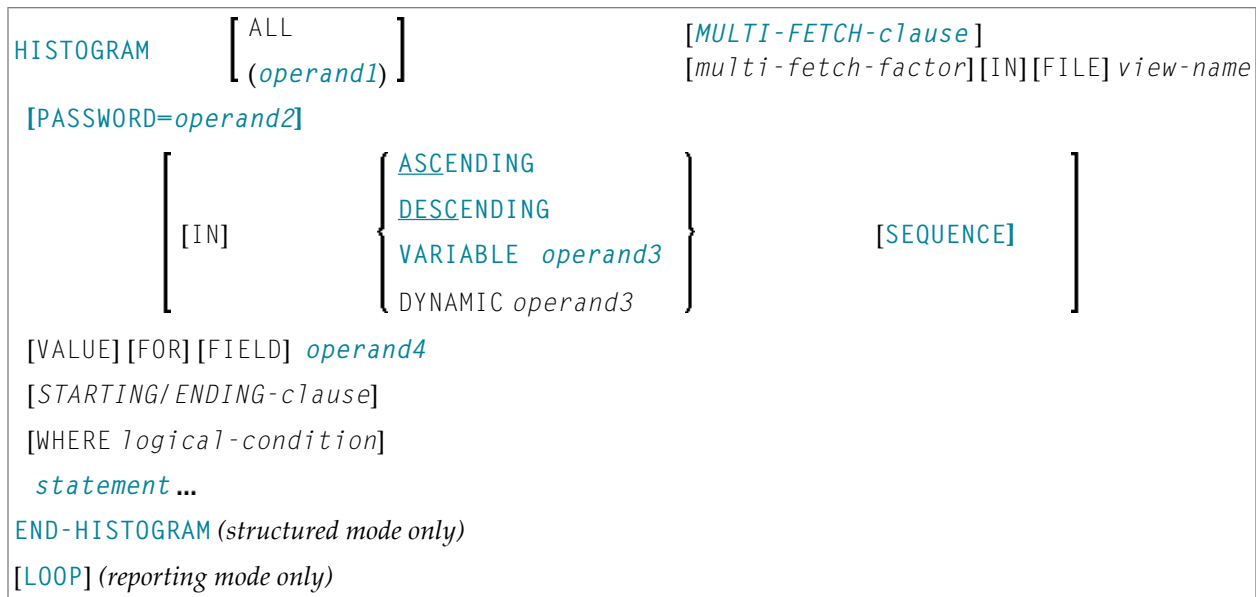
GET TRANSACTION DATA

```
END-IF
*
REPEAT
  /*
  INPUT 10X 'ENTER PERSONNEL NUMBER TO BE UPDATED:' #PERS-NR
  IF #PERS-NR = ' '
    STOP
  END-IF
  /*
  FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  IF NO RECORDS FOUND
    REINPUT 'NO RECORD FOUND'
  END-NOREC
  INPUT (AD=M) PERSONNEL-ID (AD=0)
    / NAME
    / FIRST-NAME
    / CITY

  UPDATE
  END TRANSACTION #PERS-NR
END-FIND
  /*
END-REPEAT
END
```

69 HISTOGRAM

▪ Funktion	436
▪ Einschränkungen	437
▪ Syntax-Beschreibung	437
▪ Beispiele	444



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement HISTOGRAM dient dazu, Werte eines Datenbankfeldes, das als Deskriptor, Subdeskriptor oder Superdeskriptor definiert ist, zu lesen. Die Werte werden direkt von der Adabas Invertierten Liste (bzw. dem VSAM-Index) gelesen.

Das HISTOGRAM-Statement löst zwar eine Verarbeitungsschleife aus, es kann aber auf keine anderen Datenbankfelder als auf das mit dem HISTOGRAM-Statement angegebene Feld zugegriffen werden.

Siehe auch *HISTOGRAM-Statement* im *Leitfaden zur Programmierung*.



Anmerkung: Bei SQL-Datenbanken: Mit HISTOGRAM erhalten Sie die Anzahl der Reihen, die in einer bestimmten Spalte den gleichen Wert haben.

Einschränkungen

Dieses Statement kann nicht bei XML-Datenbanken oder mit Entire System Server verwendet werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate										Referenzierung erlaubt	Dynam. Definition					
<i>operand1</i>	C	S						N	P	I	B *							ja	nein
<i>operand2</i>	C	S					A											ja	nein
<i>operand3</i>		S					A											ja	nein
<i>operand4</i>		S					A	N	P	I	F	B	D	T	L			nein	nein

* Format B von *operand1* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

<p><i>operand1</i> ALL</p>	<p>Begrenzen der Schleifendurchläufe</p> <p>Sie können die Anzahl der Deskriptorwerte, die mit dem HISTOGRAM-Statement gelesen werden sollen, auf eine bestimmte Zahl (<i>operand1</i>) begrenzen, die Sie entweder als numerische Konstante (0 bis 99999999) oder über eine Benutzervariable (die einen Ganzzahlwert enthält) angeben.</p> <p>Andernfalls werden alle Deskriptorwerte gelesen, was Sie zusätzlich durch das Schlüsselwort ALL hervorheben können.</p> <p>Das mit <i>operand1</i> angegebene Limit hat bei diesem Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.</p> <p>Ist mit dem Profilparameter LT ein kleineres Limit gesetzt, so gilt das LT-Limit.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn Sie eine vierstellige Anzahl von Deskriptorwerten lesen möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement. 2. <i>operand1</i> wird zu Beginn des ersten HISTOGRAM-Schleifendurchlaufs ausgewertet. Wird der Wert von <i>operand1</i> innerhalb der HISTOGRAM-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Werte.
<p>MULTI-FETCH-clause</p>	<p>Siehe MULTI-FETCH-Klausel weiter unten.</p>
<p><i>view-name</i></p>	<p>Als <i>view-name</i> geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Block oder in einer programmexternen Global oder Local Data Area definiert ist.</p> <p>Der View darf außer dem im HISTOGRAM-Statement verwendeten Feld (<i>operand4</i>) keine anderen Felder enthalten.</p> <p>Ist das im View definierte Feld ein in einer Periodengruppe enthaltenes Feld oder ein multiples Feld, das mit einem Indexbereich definiert ist, dann wird jeweils nur die erste Ausprägung dieses Bereiches vom HISTOGRAM-Statement gefüllt; alle anderen Ausprägungen bleiben von der Ausführung des HISTOGRAM-Statements unberührt.</p> <p>Im Reporting Mode ist <i>view-name</i> der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.</p>
<p>PASSWORD=<i>operand2</i></p>	<p>PASSWORD-Klausel:</p> <p>Die PASSWORD-Klausel dient dazu, ein Passwort (<i>operand2</i>) anzugeben, um auf Daten einer passwort-geschützten Adabas-Datei zugreifen zu können. Weitere Informationen hierzu siehe FIND-Statement und PASSW-Statement.</p>

SEQUENCE	<p>SEQUENCE-Klausel</p> <p>Die Klausel gilt nur für Adabas- und SQL-Datenbanken.</p> <p>Mit dieser Klausel können Sie bestimmen, ob die Werte in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen.</p> <ul style="list-style-type: none"> ■ Standardmäßig werden die Werte in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort <code>ASCENDING</code> auch ausdrücklich angeben können, aber nicht müssen). ■ Wenn die Werte in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort <code>DESCENDING</code> an. ■ Wenn erst zur Laufzeit bestimmt werden soll, ob die Werte in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort <code>VARIABLE</code> oder <code>DYNAMIC</code> gefolgt von einer Variablen (<i>operand3</i>) an. Der Wert von <i>operand3</i> zu Beginn der HISTOGRAM-Verarbeitungsschleife bestimmt dann die Reihenfolge. <i>operand3</i> muss Format/Länge <code>A1</code> haben und kann den Wert <code>A</code> (für <code>ASCENDING</code> = aufsteigend) oder <code>D</code> (für <code>DESCENDING</code> = absteigend) enthalten. ■ Wenn das Schlüsselwort <code>VARIABLE</code> benutzt wird, wird die Leserichtung (Wert von <i>operand3</i>) beim Start der HISTOGRAM-Verarbeitungsschleife ausgewertet, und bleibt dieselbe, bis die Schleife beendet wird, ungeachtet der Tatsache, ob das Feld für <i>operand3</i> in der HISTOGRAM-Schleife geändert wird oder nicht. ■ Wenn das Schlüsselwort <code>DYNAMIC</code> benutzt wird, wird die Leserichtung (Wert von <i>operand3</i>) vor jedem Einlesen eines Datensatzes in der HISTOGRAM-Verarbeitungsschleife ausgewertet und kann von Datensatz zu Datensatz geändert werden. Dies ermöglicht das Ändern der Blätter-Reihenfolge von aufsteigend in absteigend (und umgekehrt) an einer beliebigen Stelle in der HISTOGRAM-Schleife. <p>Beispiele für SEQUENCE-Klausel:</p> <ul style="list-style-type: none"> ■ <i>Beispiel 2 – HISTOGRAM-Statement mit in absteigender Reihenfolge gelesenen Sätzen</i> ■ <i>Beispiel 3 – HISTOGRAM-Statement mit variabler Reihenfolge</i>
----------	--

<p><i>operand4</i></p>	<p>Deskriptor:</p> <p>Als <i>operand4</i> kann ein Deskriptor, ein Subdeskriptor, ein Superdeskriptor oder ein Hyperdeskriptor angegeben werden.</p> <p>Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann entweder mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in irgendeiner Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Es muss ein konstanter Index angegeben werden; es darf kein Indexbereich angegeben werden.</p> <p>Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.</p>
<p>STARTING-ENDING-clause</p>	<p>STARTING/ENDING-Klausel:</p> <p>Mit den Schlüsselwörtern STARTING und ENDING (bzw. THRU) können Sie einen Startwert und einen Endwert angeben, und zwar in Form einer Konstanten oder einer Benutzervariablen. Damit legen Sie fest, ab welchem Wert und bis zu welchem Wert gelesen werden soll.</p> <p>Weitere Informationen siehe <i>Start-/Endwerte angeben</i> weiter unten.</p>
<p>WHERE</p> <p><i>logical-condition</i></p>	<p>WHERE-Klausel:</p> <p>Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (<i>logical-condition</i>) angeben. Dies wird ausgewertet, <i>nachdem</i> ein Wert gelesen wurde, aber bevor eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich AT BREAK-Verarbeitung) erfolgt.</p> <p>Der in der WHERE-Klausel angegebene Deskriptor muss mit dem im HISTOGRAM-Statement referenzierten Deskriptor identisch sein. Keine anderen Felder von der ausgewählten Datei stehen zur Verarbeitung bei einem HISTOGRAM-Statement zur Verfügung.</p> <p>Die Syntax für eine <i>logical-condition</i> ist im Abschnitt <i>Logische Bedingungen im Leitfaden zur Programmierung</i> beschrieben.</p> <p>Systemvariablen</p> <p>Die Natural-Systemvariablen *ISN, *NUMBER und *COUNTER können bei einem HISTOGRAM-Statement verwendet werden.</p> <p>*NUMBER und *ISN stehen erst nach Auswertung der WHERE-Klausel zur Verfügung. Sie dürfen nicht innerhalb der logischen Bedingung einer WHERE-Klausel eingesetzt werden.</p>

	*NUMBER	*NUMBER enthält die Anzahl der Datensätze, die den zuletzt gelesenen Wert enthalten. Bezüglich SQL-Datenbanken siehe *NUMBER bei SQL Datenbanken in der <i>Systemvariablen</i> -Dokumentation.
	*ISN	*ISN enthält die Anzahl der Ausprägung einer Periodengruppe des aktuellen Datensatzes, die den zuletzt gelesenen Wert enthält. Ist der Deskriptor nicht in einer Periodengruppe enthalten, enthält *ISN den Wert 0. Bei SQL-Datenbanken kann *ISN nicht verwendet werden.
	*COUNTER	*COUNTER enthält die Gesamtzahl der bisher gelesenen Werte (nach Auswertung der WHERE-Klausel).
END-HISTOGRAM	Das für Natural reservierte Schlüsselwort END-HISTOGRAM muss zum Beenden des HISTOGRAM-Statements benutzt werden.	

MULTI-FETCH-Klausel



Anmerkung: Diese Klausel kann nur bei Adabas-Datenbanken benutzt werden.

```
[ MULTI-FETCH { ON
                OFF
                OF multi-fetch-factor } ]
```



Anmerkung: [MULTI-FETCH OF *multi-fetch-factor*] wird bei den Datenbanktypen ADA und ADA2 nicht ausgewertet. Es erfolgt die Standardverarbeitung (siehe Profilparameter MFSET). Beim Datenbanktyp ADA2 wird die MULTI-FETCH-Klausel komplett ignoriert; siehe *Database Management System Assignments* in der *Configuration Utility*-Dokumentation.

Weitere Informationen siehe *Multi-Fetch-Klausel (Adabas)* im *Leitfaden zur Programmierung*.

Start-/Endwerte angeben

Mit den Schlüsselwörtern `STARTING` und `ENDING` (bzw. `THRU`) können Sie einen Startwert und einen Endwert angeben, und zwar in Form einer Konstanten oder einer Benutzervariablen. Damit legen Sie fest, ab welchem Wert und bis zu welchem Wert gelesen werden soll.

Wenn der angegebene Startwert nicht vorhanden ist, wird der nächsthöhere vorhandene Wert als Startwert genommen. Ist kein höherer Wert vorhanden, wird die `HISTOGRAM`-Schleife nicht ausgeführt.

Wenn Sie einen Endwert angeben, wird bis einschließlich des Endwertes gelesen.

Für Deskriptoren des Formats A oder B können hexadezimale Konstanten als Start- und Endwert angegeben werden.

Syntax-Option 1:

```
1 [ [STARTING] [ WITH FROM ] [VALUES] operand5 ] [ [ THRU ENDING AT ] operand6 ]
```

Syntax-Option 2:

```
2 [STARTING] [ WITH FROM ] [VALUES] operand5 TO operand6
```

Syntax-Option 3:

```
3 {
    <
    LT
    LESS THAN
    >
    GT
    GREATER THAN
    <=
    LE
    LESS EQUAL
    >=
    GE
    GREATER EQUAL
} operand5
```



Anmerkung: Wenn die Vergleichsoperatoren von Diagramm 3 benutzt werden, dürfen die Optionen `ENDING AT`, `THRU` und `TO` nicht benutzt werden. Diese Vergleichsoperatoren sind auch für das `READ`-Statement gültig.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand5</i>	C S	A U N P I F B D T L	ja	nein
<i>operand6</i>	C S	A U N P I F B D T L	ja	nein

Syntax-Element-Beschreibung:

STARTING FROM ... ENDING AT/TO	<p>Die Klauseln <code>STARTING FROM</code> und <code>ENDING AT</code> werden benutzt, um das Lesen auf einen vom Benutzer angegebenen Bereich von Werten einzuschränken.</p> <p>Die <code>STARTING FROM</code>-Klausel (= oder <code>EQ</code> oder <code>EQUAL TO</code> oder <code>[STARTING] FROM</code>) legt den Startwert für die Lese-Operation fest. Wenn ein Startwert angegeben wird, beginnt der Lesevorgang bei dem angegebenen Wert. Wenn der Startwert nicht vorhanden ist, wird der nächsthöhere Wert (oder nächstniedrigere Wert für einen absteigenden Lesevorgang) zurückgegeben. Ist kein höherer Wert (oder niedrigerer Wert für <code>DESCENDING</code>) vorhanden, wird die <code>HISTOGRAM</code>-Schleife nicht ausgeführt.</p> <p>Um die Werte auf einen Endwert zu beschränken, können Sie eine <code>ENDING AT</code>-Klausel bei den Schlüsselwörtern <code>THRU</code>, <code>ENDING AT</code> oder <code>TO</code> angeben, die einen einschließenden Bereich implizieren. Immer wenn das Deskriptorfeld den angegebenen Endwert überschreitet, erfolgt eine automatische Beendigung der Schleife. Obwohl die Basis-Funktionalität der Schlüsselwörter <code>TO</code>, <code>THRU</code> und <code>ENDING AT</code> ähnlich ist, so unterscheiden sie sich doch intern durch ihre Funktionsweise.</p>
THRU/ENDING AT	<p>Wenn <code>THRU</code> oder <code>ENDING AT</code> benutzt wird, wird nur der Startwert der Datenbank bekannt gegeben; es wird aber vom Natural-Laufzeitsystem eine Prüfung auf den Endwert vorgenommen, nachdem der Wert von der Datenbank zurückgegeben worden ist.</p> <p>Die Klauseln <code>THRU</code> und <code>ENDING AT</code> können für alle Datenbanken benutzt werden, die die <code>HISTOGRAM</code>-Statements unterstützen.</p>
TO	<p>Wird das Schlüsselwort <code>TO</code> benutzt, werden sowohl der Startwert als auch der Endwert der Datenbank übergeben, und Natural prüft nicht auf Wertebereiche ab. Wenn der Endwert überschritten wird, reagiert die Datenbank genauso, als sei das Dateiende (End-of-File) erreicht worden, und die Datenbank-Schleife wird verlassen. Da die ganzen Bereichsprüfungen von der Datenbank durchgeführt werden, wird der niedrigere Wert (des Bereichs) immer beim Startwert und der höhere Wert beim Endwert angegeben, ganz gleich ob Sie einen Lesevorgang in aufsteigender (<code>ASCENDING</code>) oder absteigender (<code>DESCENDING</code>) Reihenfolge durchführen.</p>

Beispiele

- Beispiel 1 — HISTOGRAM-Statement
- Beispiel 2 — HISTOGRAM-Statement mit in absteigender Reihenfolge gelesenen Sätzen
- Beispiel 3 — HISTOGRAM-Statement mit variabler Reihenfolge

Beispiel 1 — HISTOGRAM-Statement

```

** Example 'HSTEX1S': HISTOGRAM (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
END-DEFINE
*
LIMIT 8
HISTOGRAM EMPLOY-VIEW CITY STARTING FROM 'M'
  DISPLAY NOTITLE
    CITY 'NUMBER OF/PERSONS' *NUMBER *COUNTER
END-HISTOGRAM
*
END
    
```

Ausgabe des Programms HSTEX1S:

CITY	NUMBER OF PERSONS	CNT
MADISON	3	1
MADRID	41	2
MAILLY LE CAMP	1	3
MAMERS	1	4
MANSFIELD	4	5
MARSEILLE	2	6
MATLOCK	1	7
MELBOURNE	2	8

Äquivalentes Reporting Mode-Beispiel: [HSTEX1R](#).

Beispiel 2 — HISTOGRAM-Statement mit in absteigender Reihenfolge gelesenen Sätzen

```

** Example 'HSTDSCND': HISTOGRAM (with DESCENDING)
*****
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
HISTOGRAM (10) EMPL IN DESCENDING SEQUENCE FOR NAME FROM 'ZZZ'
  DISPLAY NAME *NUMBER
END-HISTOGRAM
END

```

Ausgabe des Programms HSTDSCND:

```

Page          1                               05-01-13  13:41:03
          NAME                NMBR
-----
ZINN                      1
YOT                        1
YNCLAN                     1
YATES                      1
YALCIN                     1
YACKX-COLTEAU             1
XOLIN                      1
WYLLIS                    2
WULFRING                  1
WRIGHT                    1

```

Beispiel 3 — HISTOGRAM-Statement mit variabler Reihenfolge

```

** Example 'HSTVSEQ': HISTOGRAM (with VARIABLE SEQUENCE)
*****
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
*
1 #DIR      (A1)
1 #STARTVAL (A20)
END-DEFINE
*
SET KEY PF3 PF7 PF8
*
MOVE 'ADKINSON' TO #STARTVAL

```

HISTOGRAM

```
*
HISTOGRAM (9) EEMPL FOR NAME FROM #STARTVAL
  WRITE NAME *NUMBER
  IF *COUNTER = 5
    MOVE NAME TO #STARTVAL
  END-IF
END-HISTOGRAM
*
#DIR := 'A'
*
REPEAT
  HISTOGRAM EEMPL IN VARIABLE #DIR SEQUENCE
    FOR NAME FROM #STARTVAL
      MOVE NAME TO #STARTVAL
      INPUT NO ERASE (IP=OFF AD=0)
      15/01 NAME *NUMBER
      // 'Direction:' #DIR
      // 'Press PF3 to stop'
      / ' PF7 to go step back'
      / ' PF8 to go step forward'
      / ' ENTER to continue in that direction'
/*
  IF *PF-KEY = 'PF7' AND #DIR = 'A'
    MOVE 'D' TO #DIR
    ESCAPE BOTTOM
  END-IF
  IF *PF-KEY = 'PF8' AND #DIR = 'D'
    MOVE 'A' TO #DIR
    ESCAPE BOTTOM
  END-IF
  IF *PF-KEY = 'PF3'
    STOP
  END-IF
END-HISTOGRAM
/*
  IF *COUNTER(0250) = 0
    STOP
  END-IF
END-REPEAT
END
```

Ausgabe des Programms HSTVSEQ:

```
Page      1                                     05-01-13  13:50:31

ADKINSON           8
AECKERLE           1
AFANASSIEV         2
AHL                 1
AKROYD              1
ALEMAN              1
```

ALESTIA	1
ALEXANDER	5
ALLEGRE	1

MORE

Nach Drücken von EINGABE:

Page 1 05-01-13 13:50:31

ADKINSON	8
AECKERLE	1
AFANASSIEV	2
AHL	1
AKROYD	1
ALEMAN	1
ALESTIA	1
ALEXANDER	5
ALLEGRE	1

AKROYD	1
--------	---

Direction: A

Press PF3 to stop
PF7 to go step back
PF8 to go step forward
ENTER to continue in that direction

70 IF

▪ Funktion	450
▪ Syntax-Beschreibung	451
▪ Beispiel — IF-Statement	451

Structured Mode-Syntax

```
IF logical-condition
  [THEN] statement ...
  [ELSE statement ...]
END-IF
```

Reporting Mode-Syntax

```
IF logical-condition
  [THEN] { statement
           DO statement ... DOEND }
  [ ELSE { statement
           DO statement ... DOEND } ]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [DECIDE FOR](#) | [DECIDE ON](#) | [IF SELECTION](#) | [ON ERROR](#)

Gehört zur Funktionsgruppe: [Logische Bedingungen](#)

Funktion

Mit dem Statement IF wird die Verarbeitung eines Statements oder einer Gruppe von Statements in Abhängigkeit von einer logischen Bedingung (*logical-condition*) gesteuert.



Anmerkung: Falls keine Handlung ausgeführt werden soll, wenn die Bedingung erfüllt ist, geben Sie das Statement `IGNORE` in der THEN-Klausel an.

Syntax-Beschreibung

IF <i>logical-condition</i>	<p>Die logische Bedingung, die Sie definieren, bestimmt, wann das Statement bzw. die Statements, die Sie im IF-Statement angeben, ausgeführt werden soll/en und wann nicht.</p> <p>Beispiele:</p> <pre>IF #A = #B IF LEAVE-TAKEN GT 30 IF #SALARY(1) * 1.15 GT 5000 IF SALARY (4) = 5000 THRU 6000 IF DEPT = 'A10' OR = 'A20' OR = 'A30'</pre> <p>Näheres hierzu siehe Abschnitt <i>Logische Bedingungen</i> im <i>Leitfaden zur Programmierung</i>.</p>
THEN <i>statement</i>	<p>In der THEN-Klausel geben Sie eines oder mehrere <i>Statements</i> an, die ausgeführt werden sollen, wenn die logische Bedingung erfüllt ist.</p>
ELSE <i>statement</i>	<p>In der ELSE-Klausel geben Sie eines oder mehrere <i>Statements</i> an, die ausgeführt werden sollen, wenn die logische Bedingung <i>nicht</i> erfüllt ist.</p>
END-IF	<p>Das für Natural reservierte Wort END-IF muss zum Beenden des IF-Statements benutzt werden.</p>

Beispiel — IF-Statement

```
** Example 'IFEX1S': IF (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY (1)
  2 BIRTH
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
SUSPEND IDENTICAL SUPPRESS
LIMIT 20
```

```

*
FND. FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      SORTED BY NAME BIRTH
IF SALARY (1) LT 40000
  WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
ELSE
  IF BIRTH GT #BIRTH
    FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FND.)
    DISPLAY (IS=ON)
      NAME BIRTH (EM=YYYY-MM-DD)
      SALARY (1) MAKE (AL=8)
  END-FIND
END-IF
END-IF
END-FIND
END

```

Ausgabe des Programms IFEX1S:

NAME	DATE OF BIRTH	ANNUAL SALARY	MAKE	
BAECKER	1956-01-05	74400	BMW	
***** BECKER				SALARY LT 40000
BLOEMER	1979-11-07	45200	FIAT	
FALTER	1954-05-23	70800	FORD	
***** FALTER				SALARY LT 40000
***** GROTHE				SALARY LT 40000
***** HEILBROCK				SALARY LT 40000
***** HESCHMANN				SALARY LT 40000
HUCH	1952-09-12	67200	MERCEDES	
***** KICKSTEIN				SALARY LT 40000
***** KLEENE				SALARY LT 40000
***** KRAMER				SALARY LT 40000

Äquivalentes Reporting-Mode-Beispiel: [IFEX1R](#).

71 IF SELECTION

▪ Funktion	454
▪ Syntax-Beschreibung	455
▪ Beispiel — IF SELECTION-Statement	456

Structured Mode-Syntax

```
IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1 ...  
  [THEN] statement...  
  [ELSE statement...]  
END-IF
```

Reporting Mode-Syntax

```
IF SELECTION [NOT UNIQUE [IN [FIELDS]]] operand1...  
  [THEN] { statement }  
          DO statement... DOEND  
  [ ELSE { statement } ]  
          DO statement... DOEND ]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [DECIDE FOR](#) | [DECIDE ON](#) | [IF](#)

Gehört zur Funktionsgruppe: [Logische Bedingungen](#)

Funktion

Das Statement `IF SELECTION` dient dazu, zu verifizieren, dass in einer Reihe von alphanumerischen Feldern genau ein Feld einen Wert enthält.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A	A U L C	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Auswahlfeld:</p> <p>Als <i>operand1</i> geben Sie die Felder an, die verifiziert werden sollen.</p> <p>Wenn Sie eine Kontrollvariable (Format C) angeben, so wird angenommen, dass sie einen Wert enthält, wenn ihr Status sich auf MODIFIED geändert hat.</p> <p>Anmerkung: Um zu überprüfen, ob einer bestimmten Kontrollvariable der Status MODIFIED zugewiesen wurde, benutzen Sie die MODIFIED-Option z.B. eines IF-Statements. Damit können Sie abprüfen, dass genau ein Feld <i>geändert</i> wurde.</p>
THEN <i>statement</i>	<p>Ausführende Statements:</p> <p>Das (oder die) in der THEN-Klausel angegebenen Statement(s) werden ausgeführt, wenn eine der folgenden Bedingungen erfüllt ist:</p> <ul style="list-style-type: none"> ■ Keines der angegebenen Felder (<i>operand1</i>) enthält einen Wert. ■ Mehr als eines der angegebenen Felder (<i>operand1</i>) enthält einen Wert. <p>Dieses Statement wird in der Regel dazu eingesetzt, zu verifizieren, dass auf einer über ein INPUT-Statement erzeugten Map vom Terminal-Benutzer nicht gleichzeitig mehr als eine Funktion eingegeben wurde.</p> <p>Anmerkung: Falls keine Maßnahme ausgeführt werden soll, wenn eine der beiden Bedingungen erfüllt ist, geben Sie das Statement IGNORE in der THEN-Klausel an.</p>
ELSE <i>statement</i>	In der ELSE-Klausel geben Sie das (oder die) Statement(s) an, die ausgeführt werden sollen, wenn genau ein Feld einen Wert enthält.
END-IF	Das für Natural reservierte Wort END-IF muss zum Beenden des IF SELECTION-Statements benutzt werden.

Beispiel — IF SELECTION-Statement

```
** Example 'IFSEL': IF SELECTION
*****
DEFINE DATA LOCAL
1 #A (A1)
1 #B (A1)
END-DEFINE

*
INPUT 'Select one function:' //
    9X 'Funktion A:' #A
    9X 'Funktion B:' #B
*
IF SELECTION NOT UNIQUE #A #B
    REINPUT 'Please enter one function only.'
END-IF
*
IF #A NE ' '
    WRITE 'Funktion A selected.'
END-IF
IF #B NE ' '
    WRITE 'Funktion B selected.'
END-IF
*
END
```

Ausgabe des Programms IFSEL:

```
Select one function:
      Funktion A:      Funktion B:
```

Nach Auswahl und Bestätigung der Funktion A:

```
Page      1                                05-01-17  11:04:07
Funktion A selected.
```

72 IGNORE

▪ Funktion	458
▪ Beispiel — IGNORE-Statement	458

IGNORE

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `IGNORE` ist ein *leeres* Statement, das selbst keine Funktion ausführt.

Während der Entwicklungsphase einer Anwendung können Sie `IGNORE` vorübergehend innerhalb von Statement-Blöcken einsetzen, in denen ein oder mehrere Statements angegeben werden müssen, welche Sie aber erst später codieren möchten (z.B. in `AT BREAK` oder `AT START OF DATA/AT END OF DATA`). Sie können dann die Programmierung in einem anderen Teil Ihrer Anwendung fortsetzen, ohne dass der noch unvollständige Statement-Block zu einem Fehler führt.

Das `IGNORE`-Statement muss auch in Bedingungs-Statements wie `IF` oder `DECIDE FOR` verwendet werden, wenn keine Funktion ausgeführt werden soll, falls eine bestimmte Bedingung erfüllt ist.

Beispiel — IGNORE-Statement

```
...
...
AT TOP OF PAGE
  IGNORE      /* top-of-page processing still to be coded
END-TOPPAGE
...
...
```

73 INCLUDE

▪ Funktion	460
▪ Syntax-Beschreibung	460
▪ Beispiele	462

```
INCLUDE copycode-name [operand1 ... 99]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das Statement INCLUDE dient dazu, den Sourcecode eines externen Objekts vom Typ Copycode bei der Kompilierung in ein anderes Objekt einzufügen.

Das INCLUDE-Statement wird bei der *Kompilierung* ausgewertet. Die Sourcecode-Zeilen des Copycodes werden nicht physisch in den Sourcecode des Programms eingefügt, das das INCLUDE-Statement enthält, und der eingefügte Copycode ist als Teil des Objektmoduls im kompilierten Programm enthalten.



Anmerkung: Eine Sourcecode-Zeile, die ein INCLUDE-Statement enthält, darf kein anderes Statement enthalten.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C	A U	nein	nein

Syntax-Element-Beschreibung:

<i>copycode-name</i>	<p>Copycode-Name:</p> <p>Als <i>copycode-name</i> geben Sie den Namen des Copycodes an, dessen Source eingefügt werden soll.</p> <p>Der <i>copycode-name</i> kann ein Kaufmännisches Und (&) enthalten; bei der Kompilierung wird dieses Zeichen durch den aus einem Zeichen bestehenden Code ersetzt, der dem aktuellen Wert der Systemvariablen *LANGUAGE entspricht. Diese Funktion ermöglicht die Verwendung mehrsprachiger Copycode-Namen.</p> <p>Das Objekt, dessen Namen Sie angeben, muss vom Objekttyp Copycode sein. Der Copycode muss entweder in derselben Library gespeichert sein wie das Programm, das</p>
----------------------	--

	<p>das INCLUDE-Statement enthält, oder in der betreffenden Steplib (Standard-Steplib ist SYSTEM).</p> <p>Wenn der Sourcecode des Copycodes verändert wird, müssen alle Programme, in denen der Copycode eingefügt ist, neu kompiliert werden, damit die Änderungen in den Objektmodulen zum Tragen kommen.</p> <p>Der Sourcecode des Copycodes muss aus syntaktisch vollständigen Statements bestehen.</p>
<i>operand1</i>	<p>Dynamisch einzufügende Werte:</p> <p>In den einzufügenden Copycode können Sie dynamisch Werte einsetzen. Diese Werte werden mit <i>operand1</i> angegeben.</p> <p>Im Copycode werden die Werte mit der folgenden Notation referenziert:</p> <p><code>&n&</code></p> <p>d.h. Sie markieren die Stelle, an der ein Wert eingesetzt werden soll, mit <code>&n&</code>. <i>n</i> ist die laufende Nummer des mit dem INCLUDE-Statement übergebenen Wertes. Zum Beispiel würde sich <code>&3&</code> auf den dritten übergebenen Wert beziehen.</p> <p>Für jede <code>&n&</code>-Notation im Copycode müssen Sie im INCLUDE-Statement einen Wert angeben. Wenn der Copycode beispielsweise <code>&5&</code> enthält, muss <i>operand1</i> mindestens fünfmal angegeben werden.</p> <p>Sie können einen Copycode-Parameter (<code>&n&</code>) hinter dem anderen ohne Leerzeichen (d.h. <code>&1&&2&&3&</code>) schreiben. Diese Methode wird zur Verkettung mehrerer Copycode-Parameter mit einer Source benutzt.</p> <p>Eine Zeichenkette kann einem oder mehreren Copycode-Parametern ohne ein Leerzeichen folgen (d.h. <code>&1&abc</code> oder <code>&1&&2&abc</code>). Diese Methode wird zur Verkettung einer Zeichenkette mit mehreren Copycode-Parametern benutzt.</p> <p>Anmerkung: Da <code>&n&</code> ein gültiger Teil eines Namens ist, darf diese Notation nicht als Ersatzzeichen für einen Copycode-Parameter in anderen, oben beschriebenen Positionen (d.h. <code>abc&1&</code> or <code>&1&abc&2&</code>) benutzt werden. Mit anderen Worten kann eine Zeichenkette nur hinter Copycode-Parametern auftreten, nicht davor oder dazwischen.</p> <p>Mit dem INCLUDE-Statement angegebene Werte, die im Copycode nicht referenziert werden, werden ignoriert.</p>

Beispiele

- Beispiel 1 — INCLUDE-Statement mit einzufügendem Copycode
- Beispiel 2 — INCLUDE-Statement mit einzufügendem Copycode mit Parametern
- Beispiel 3 — INCLUDE-Statement mit geschachtelten Copycodes
- Beispiel 4 — INCLUDE-Statement mit verketteten Parametern in Copycode

Beispiel 1 — INCLUDE-Statement mit einzufügendem Copycode

INCEX1 ist das Programm, das das INCLUDE-Statement enthält:

```
** Example 'INCEX1': INCLUDE (include copycode)
*****
*
WRITE 'Before copycode'
*
INCLUDE INCEX1C
*
WRITE 'After copycode'
*
END
```

Einzufügender Copycode INCEX1C:

```
** Example 'INCEX1C': INCLUDE (copycode used by INCEX1)
*****
*
WRITE 'Inside copycode'
```

Ausgabe des Programms INCEX1:

```
Page      1                                05-01-25  16:26:36
Before copycode
Inside copycode
After copycode
```

Beispiel 2 — INCLUDE-Statement mit einzufügendem Copycode mit Parametern

INCEX2 ist das Programm, das das INCLUDE-Statement enthält:

```
** Example 'INCEX2': INCLUDE (include copycode with parameters)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
*
INCLUDE INCEX2C 'EMPL-VIEW' 'NAME' '' 'ARCHER'' '20' '' 'BAILLET''
*
END
```

Einzufügender Copycode INCEX2C:

```
** Example 'INCEX2C': INCLUDE (copycode used by INCEX2)
*****
* Transferred parameters from INCEX2:
*
* &1& : EMPL-VIEW
* &2& : NAME
* &3& : 'ARCHER'
* &4& : 20
* &5& : 'BAILLET'
*
*
READ (&4&) &1& BY &2& = &3&
  DISPLAY &2&
  IF &2& = &5&
    WRITE 5X 'LAST RECORD FOUND' &2&
    STOP
  END-IF
END-READ
*
* Statements above will be completed to:
*
* READ (20) EMPL-VIEW BY NAME = 'ARCHER'
*   DISPLAY NAME
*   IF NAME = 'BAILLET'
*     WRITE 5X 'LAST RECORD FOUND' NAME
*     STOP
*   END-IF
* END-READ
```

Ausgabe des Programms INCEX2:

```

Page      1                               05-01-25  16:30:43
      NAME
-----
ARCHER
ARCONADA
ARCONADA
ARNOLD
ASTIER
ATHERTON
ATHERTON
ATHERTON
AUBERT
BACHMANN
BAECKER
BAECKER
BAGAZJA
BAILLET
      LAST RECORD FOUND BAILLET

```

Beispiel 3 — INCLUDE-Statement mit geschachtelten Copycodes

INCEX3 ist das Programm, das das INCLUDE-Statement enthält:

```

** Example 'INCEX3': INCLUDE (using nested copycodes)
*****
DEFINE DATA LOCAL
1 #A (I4)
END-DEFINE
*
MOVE 123 TO #A
WRITE 'Program INCEX3 ' '=' #A
*
INCLUDE INCEX31C '#A' '5'           /* source line is #A := 5
*
*
MOVE 300 TO #A
WRITE 'Program INCEX3 ' '=' #A
*
INCLUDE INCEX32C '''#A''' '''20''' /* source line is #A := 20
*
WRITE 'Program INCEX3 ' '=' #A
END

```

Einzufügender Copycode INCEX31C:

```

** Example 'INCEX31C': INCLUDE (copycode used by INCEX3)
*****
* Transferred parameters from INCEX3:
*
* &1& : #A
* &2& : 5
*
*
&1& := &2&
*
WRITE 'Copycode INCEX31C' '=' &1&

```

Einzufügender Copycode INCEX32C:

```

** Example 'INCEX32C': INCLUDE (copycode used by INCEX3)
*****
* Transferred parameters from INCEX3:
*
* &1& : '#A'
* &2& : '20'
*
*
WRITE 'Copycode INCEX32C' &1& &2&
*
INCLUDE INCEX31C &1& &2&

```

Ausgabe des Programms INCEX3:

```

Page          1                                     05-01-25  16:35:36

Program INCEX3   #A:          123
Copycode INCEX31C #A:           5
Program INCEX3   #A:          300
Copycode INCEX32C #A 20
Copycode INCEX31C #A:           20
Program INCEX3   #A:           20

```

Beispiel 4 — INCLUDE-Statement mit verketteten Parametern in Copycode

INCEX4 ist das Programm, das das INCLUDE-Statement enthält:

```
** Example 'INCEX4': INCLUDE (with concatenated parameters in copycode)
*****
DEFINE DATA LOCAL
1 #GROUP
  2 ABC(A10) INIT <'1234567890'>
END-DEFINE
*
INCLUDE INCEX4C '#GROUP.' 'ABC' 'AB'
*
END
```

Einzufügender Copycode INCEX4C:

```
** Example 'INCEX4C': INCLUDE (copycode used by INCEX4)
*****
* Transferred parameters from INCEX4:
*
* &1& : #GROUP.
* &2& : ABC
* &3& : AB
*
*
WRITE '=' &2& /* 'ABC' results into ABC
WRITE '=' &1&ABC /* '#GROUP.' ABC results into #GROUP.ABC
WRITE '=' &1&&2& /* '#GROUP.' 'ABC' results into #GROUP.ABC
WRITE '=' &1&&3&C /* '#GROUP.' 'AB' C results into #GROUP.ABC
```

Ausgabe des Programms INCEX4:

```
Page      1                                     05-01-25 16:37:59

ABC: 1234567890
ABC: 1234567890
ABC: 1234567890
ABC: 1234567890
```

74 INPUT

▪ Funktion	468
▪ Eingabe-Modi	468
▪ Eingabe von Daten als Reaktion auf ein INPUT-Statement	470
▪ SB – Auswahlfenster (Selection Box)	472
▪ Eingabefehler	473
▪ Geteilter Schirm (Split Screen)	473
▪ Systemvariablen beim INPUT-Statement	473

Dieses Kapitel behandelt folgende Themen:

Die Syntax des `INPUT`-Statements wird in den folgenden Abschnitten beschrieben:

- **INPUT-Syntax 1 – Dynamisch generierter Eingabeschirm**
- **INPUT-Syntax 2 – Verwendung einer vordefinierten Eingabemaske**

Verwandte Statements: `DEFINE WINDOW` | `REINPUT` | `SET WINDOW`

Gehört zur Funktionsgruppe: *Bildschirmgenerierung für interaktive Verarbeitung*

Funktion

Das `INPUT`-Statement dient bei der interaktiven Verarbeitung dazu, formatierte Schirme oder Maps auszugeben oder zu generieren, die zur Eingabe von Daten verwendet werden.

Das Statement kann auch in Verbindung mit dem Natural-Stack (siehe `STACK`-Statement) verwendet werden, sowie zur Eingabe von Benutzerdaten bei Programmen, die im Batch-Betrieb ausgeführt werden.

Für den Natural Remote Procedure Call (RPC): Siehe *Notes on Natural Statements on the Server* in der *Natural Remote Procedure Call (RPC)*-Dokumentation.

Eingabe-Modi

Das `INPUT`-Statement kann unter drei verschiedenen Eingabe-Modi verwendet werden: Screen-Modus, Forms-Modus oder Keyword/Delimiter-Modus. Im Falle von Videoterminals/ Videobildschirmen wird in der Regel der Screen-Modus verwendet. Forms-Modus kann bei TTY-Terminals verwendet werden. Keyword/Delimiter-Modus kann bei TTY-Terminals oder im Batch-Betrieb benutzt werden (auf Großrechnern). Standardmäßig gilt Screen-Modus.

Sie können den Eingabemodus mit dem Session-Parameter `IM` ändern.

Screen-Modus

Im Screen-Modus bewirkt die Ausführung eines `INPUT`-Statements die Anzeige eines Schirms (Screen) entsprechend der angegebenen Felder und ihrer Positionen. Die Meldungszeile des Schirms wird von Natural zur Ausgabe von Fehlermeldungen benutzt. Die Position dieser Zeile (Kopf- oder Fußzeile) kann mit dem Terminalkommando `%M` beeinflusst werden. Der Terminal-Benutzer kann über die verschiedenen Tabulator-Tasten bestimmte Felder ansteuern.

Da Natural die sogenannte Bildschirmfenster- oder „Window“-Technik unterstützt, ist es erlaubt, dass die Größe einer logischen vom Programm ausgegebenen Map (Bildschirmmaske, theoretisch maximal 250 Stellen breit und 250 Zeilen lang, aber begrenzt durch den internen Bildschirm-Puffer) über die Größe des physischen Bildschirms hinausgeht.

Um ein Bildschirmfenster, d.h. den auf dem physischen Schirm sichtbaren Ausschnitt einer logischen Programmseite, zu beeinflussen und auf der logischen Seite zu verschieben, kann das Terminalkommando `%W` verwendet werden (Näheres zur Fenster-Verarbeitung siehe Terminalkommando `%W`).

Für Eingabefelder (definiert mit Session-Parameter `AD=A` oder `AD=M`), die auf dem physischen Bildschirm nicht vollständig angezeigt werden, gilt folgendes:

- Ein Eingabefeld, dessen Anfang außerhalb des Fensters liegt, wird immer zu einem geschützten Feld gemacht.
- Ein Eingabefeld, das im Fenster beginnt aber außerhalb des Fensters endet, wird nur dann geschützt, wenn der Wert, den es enthält, nicht vollständig innerhalb des Fensters sichtbar ist. Bitte beachten Sie, dass es hierbei darauf ankommt, ob die *Wertlänge*, nicht die *Feldlänge*, über das Fenster hinausgeht. Füllzeichen (wie mit dem Profilparameter `FC` oder dem Session-Parameter `AD` angegeben) zählen nicht als Teil des Wertes.
- Falls Sie in ein derart geschütztes Eingabefeld Eingaben machen möchten, müssen Sie zunächst die Fenstergröße so ändern, dass sich der Anfang des Feldes bzw. das Ende des Feldwertes innerhalb des Fensters befindet (siehe Terminalkommando `%W`).

Andere Eingabe-Modi

Das `INPUT`-Statement kann sowohl für Operationen auf zeilenorientierten Geräten wie zur Verarbeitung von Batch-Eingaben aus sequentiellen Dateien verwendet werden.

Dieselben Maps, die im interaktiven Screen-Modus verwendet werden, können auch in einem der anderen Eingabe-Modi verarbeitet werden.

Im Forms- oder Keyword/Delimiter-Modus werden die Eingaben entweder ohne Maps verarbeitet oder durch Map-Simulation im Line-Modus.

Siehe auch:

- [INPUT-Statement unter Nicht-Screen-Modi](#)

■ *Eingabedaten aus dem Natural-Stack*

Eingabe von Daten als Reaktion auf ein INPUT-Statement

Bei alphanumerischen Feldern müssen die Daten linksbündig eingegeben werden; jedes eingegebene Zeichen (auch Leerzeichen) hat eine Bedeutung. Die Daten werden ein Zeichen pro Byte dem internen Feld zugeordnet. In ein alphanumerisches Feld eingegebene Daten werden nicht auf Gültigkeit überprüft.

Die Umsetzung von Klein- in Großbuchstaben kann über die Terminalkommandos %L und %U sowie die Feldattribute AD=T und AD=W gesteuert werden.

In numerische Felder können Daten an beliebiger Stelle eingegeben werden, wobei Leerzeichen und Nullen vor und Leerzeichen nach dem eingegebenen Wert erlaubt sind; darüber hinaus dürfen ein Vorzeichen und ein Komma (Dezimalpunkt) eingegeben werden. Natural richtet den Feldwert entsprechend der internen Definition des Feldes aus.

Gilt SG=OFF, so reserviert oder vergibt Natural keine Stelle für das Vorzeichen. Bei Feldern mit Format P müssen Daten in Dezimalform eingegeben werden; falls nötig, setzt Natural dezimale Daten automatisch in gepackte um. Ein Feld, das nur Leerzeichen enthält, wird als Nullwert interpretiert.

Bei in ein numerisches Feld eingegebenen Daten überprüft Natural, ob es sich um keine anderen Zeichen als Zahlenzeichen, Komma (Dezimalpunkt, optional), Vorzeichen (optional) und vor- oder nachgestellte Leerzeichen handelt. Wird kein Komma eingegeben, so wird angenommen, dass es sich rechts neben dem eingegebenen Wert befindet.

Daten für binäre Felder müssen für alle Byte-Positionen eingegeben werden (zwei Zeichen pro Byte); es dürfen nur Hexadezimalzeichen (0 - 9, A - F) eingegeben werden. Ein Leerzeichen (H'20' in ASCII bzw. H'40' in EBCDIC) ist erlaubt und wird in binäre Nullen umgesetzt. Natural überprüft, ob keine anderen außer den gültigen Hexadezimalzeichen eingegeben wurden.

Bei logischen Feldern (Format L) kann entweder ein Leerzeichen (für *falsch*) oder ein anderes Zeichen (für *wahr*) eingegeben werden.

Bei Feldern der Formate F, D und T müssen die Daten entsprechend den für Gleitkomma-, Datums- bzw. Zeitkonstanten gültigen Regeln eingegeben werden.

Numerischer Editiermasken-Freimodus

Innerhalb eines Feldelements können Sie die Darstellung des Feldinhalts mit einer Editiermaske formatieren. Die Editiermaske dient zwei verschiedenen Zwecken:

- zum Erstellen des Layouts zur Anzeige des Feldes auf dem Bildschirm;
- zum Extrahieren der Felddaten aus der eingegebenen Zeichenkette nach dem Ändern einer Zeichenkette und dem Drücken von FREIG.

Der Vorteil der Verbesserung des Formats der mit zusätzlichen Einfügungszeichen angezeigten Felddaten kann sich als ein Nachteil herausstellen, weil ein neu eingegebener Datenwert genau dem Format der Editiermaske entsprechen muss.

Beispiel:

```
SET GLOBALS ID=; DC=,
RESET N (N7,3)
INPUT N (AD=M EM=Z'.'ZZZ'.'ZZZ,999EUR)
END
```

Ausgabewert	angezeigt als:	Eingabewert	einzugeben als:	-->Eingabefehler, wenn eingegeben als:
0	,000EUR	1	1,000EUR	1 1EUR 01,000EUR
1234	1.234,000EUR	1234567	1.234.567,000EUR	1234567 1.234.567 1.234.567EUR
0,123	,123EUR	1,234	1,234EUR	1,234

Eine andere Möglichkeit zur Eingabe von numerischen Feldern in die Editiermaske besteht in der Benutzung eines anderen INPUT-Modus, der Editiermasken-Freimodus genannt wird. Wenn sie aktiviert sind (entweder beim Session-Start über den Profilparameter EMFM oder in einer laufenden Natural-Session über das Terminalkommando %FM+), können alle oder einige der Einfügungszeichen der Editiermaske beim INPUT-Statement weggelassen werden.

Erscheint aber eine benachbarte Zeichenkette mit Einfügungszeichen in der Editiermaske (wie EUR im folgenden Beispiel), dürfen Sie nur die Zeichenkette angeben oder sie vollständig weglassen. Die Anzahl der optionalen oder zwingenden Ziffern (Editiermasken-Zeichen Z und 9), die angegeben werden müssen, wird nicht beeinflusst.

Beispiel mit aktiviertem Editiermasken-Freimodus:

```

SET GLOBALS ID=; DC=,
SET CONTROL 'FM+'          /* activate numeric Edit Mask Free Mode
RESET N (N7,3)
INPUT N (AD=M EM=Z'.'ZZZ'.'ZZZ,999EUR)
END

```

Eingabewert	kann eingegeben werden als:	führt zu einem Fehler, wenn eingegeben als:
1	1 1,0 001 1,00EUR 0.001 1,EUR	1EUR
1234567	1234567 1.234.567 1234.567 1234567,0 1.234.567,0 1.234.567,EUR 1.234.567,0EUR 1.234.567,000EUR	1.234.567EUR
1,234	1,234 1,234EUR 001,234 0.001,234EUR 00001,234EUR	1,234EU



Anmerkung: Der Editiermasken-Freimodus gilt nur für das INPUT-Statement, wird aber bei einem MOVE EDITED-Statement ignoriert.

SB – Auswahlfenster (Selection Box)

Auswahlfenster (Selection Boxes) in einem INPUT-Statement stehen nur auf Großrechnern zur Verfügung. Bei anderen Plattformen können Auswahlfenster nur im Map Editor definiert werden.



Anmerkung: Unter UNIX und OpenVMS können keine Auswahlfenster definiert werden. Auswahlfenster, die aus einer Großrechner- oder Windows-Umgebung importiert wurden, werden unter UNIX und OpenVMS ignoriert.

Auswahlfenster können an Eingabefelder angehängt werden. Sie sind eine komfortable Alternative zu an Feldern angehängte Helprountinen, da Sie ja ein Auswahlfenster direkt in Ihrem Programm kodieren können. Sie brauchen kein zusätzliches Programm wie bei Helprountinen.

Weitere Informationen entnehmen Sie der Beschreibung des Session-Parameters `SB` in der *Parameter-Referenz*.

Eingabefehler

Entsprechen die in ein Eingabefeld eingegebenen Daten nicht dem Format bzw. der Editiermaske des Feldes, gibt Natural eine entsprechende Fehlermeldung aus (ohne die Programmausführung abzubrechen) und platziert den Cursor in das betreffende Feld; der Benutzer kann dann den Fehler berichtigen und gültige Daten eingeben, woraufhin die Verarbeitung fortgesetzt wird.

Geteilter Schirm (Split Screen)

In der Regel erzeugt jedes `INPUT`-Statement eine neue Ausgabeseite (bzw. einen neuen Schirm).

Ein an ein `AT END OF PAGE`-Statement geknüpftes `INPUT`-Statement erzeugt keinen neuen Schirm. Dadurch besteht die Möglichkeit, einen geteilten Schirm (Split Screen) zu erhalten, dessen obere Hälfte mehrere Zeilen anzeigt, während in der unteren Hälfte eine Eingabe-Map erstellt werden kann.

Damit die Eingabe-Map auf denselben physischen Schirm passt, muss die logische Seitenlänge mit dem Profilparameter `PS` in einem `SET GLOBALS`- oder `FORMAT`-Statement entsprechend gesetzt werden.

Die erste `INPUT`-Zeile wird unter die letzte angezeigte Zeile platziert. Falls die `NO ERASE`-Option verwendet wird, wird die erste `INPUT`-Zeile an den Anfang der Seite platziert.

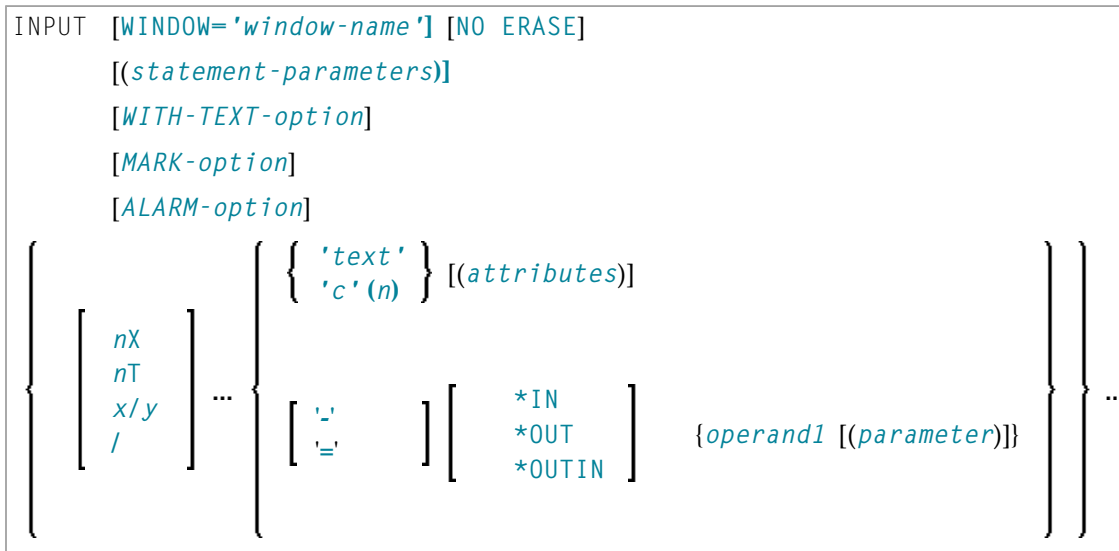
Systemvariablen beim INPUT-Statement

Zu Informationen über die relevanten Systemvariablen siehe Abschnitt *Eingabe/Ausgabebezogene Systemvariablen* in der *Systemvariablen*-Dokumentation.

75 INPUT-Syntax 1 — Dynamisch generierter Eingabeschirm

- INPUT Syntax 1 — Beschreibung 476
- Beispiele — Verwendung von Syntax 1 487

Diese Form des INPUT-Statements wird dazu verwendet, entweder einen Eingabe-Schirm zu generieren oder ein Eingabedaten-Layout zu erstellen, das (auf Großrechnern) im Batch-Betrieb von einer sequentiellen Eingabedatei gelesen werden kann.



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

INPUT Syntax 1 — Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A U N P I F B D T L G	ja	ja

Syntax-Element-Beschreibung:

INPUT WINDOW='window-name'	<p>Eingabe-Fenster:</p> <p>Mit der Option WINDOW='window-name' bewirken Sie, dass das INPUT-Statement für das angegebene Fenster (Window) ausgeführt werden soll. Das angegebene Fenster muss in einem DEFINE WINDOW-Statement definiert sein; siehe Beispiel 2 — INPUT-Statement mit DEFINE WINDOW-Statement weiter unten.</p>
---	--

	<p>Das angegebene Fenster ist nur für die Dauer des betreffenden INPUT-Statements aktiv und wird nach Ausführung des INPUT-Statements automatisch deaktiviert.</p> <p>Siehe auch die Statements DEFINE WINDOW und SET WINDOW.</p>
<p>NO ERASE</p>	<p>Überlagerte Anzeige:</p> <p>NO ERASE bewirkt, dass die vom INPUT-Statement ausgegebene Schirmanzeige eine bereits vorhandene Anzeige überlagern kann, ohne letztere zu löschen.</p> <p>Schirm bezieht sich in diesem Zusammenhang auf die logische Ausgabe und nicht auf den physischen Bildschirm.</p> <p>Alle ungeschützten Felder auf dem alten Schirm werden geschützt, so dass keine Eingaben mehr in sie gemacht werden können. Die alten Daten bleiben auf dem Schirm, bis der neue Schirm angezeigt wird. Überlagert ein Feld des neuen Schirms teilweise ein altes, so werden das Zeichen vor dem neuen Feld und das nächste Zeichen im alten Feld durch ein Leerzeichen ersetzt.</p>
<p><i>statement-parameters</i></p>	<p>Parameter auf Statement-/Feldebene:</p> <p>Unmittelbar nach dem Schlüsselwort INPUT oder nach einem der auszugebenden Felder können Sie in Klammern einen oder mehrere Session-Parameter setzen.</p> <p>Eine Liste der mit dem INPUT-Statement anzugebenden Parameter finden Sie im Abschnitt <i>Statement-Parameter</i> weiter unten.</p> <p>Diese Parameter haben dann für das jeweilige Statement oder Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht, haben aber keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen Sie explizit für dieses Element gesetzt werden.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 1 VARI (A4) INIT <'1234'> /* Output END-DEFINE /* Produced * /* ----- INPUT 'Text' VARI /* Text 1234 INPUT (AD=U) 'Text' VARI /* Text 1234 INPUT 'Text' (AD=U) VARI (AD=U) /* Text 1234 INPUT 'Text' (AD=U) VARI /* Text 1234 END </pre>

	Beispiele für den Einsatz von Parametern auf Statement- und Element-Ebene finden Sie auf den folgenden Seiten.
<i>WITH TEXT-option</i>	Textausgabe in Meldungszeile: Diese Option dient dazu, Text anzugeben, der in der Meldungszeile ausgegeben werden soll (siehe den entsprechenden Abschnitt weiter unten).
<i>MARK-option</i>	Cursorposition im Feld: Siehe Abschnitt MARK-Option weiter unten.
<i>ALARM-option</i>	Warntonausgabe: Siehe Abschnitt Alarm-Option weiter unten.
Other syntax elements (<i>nX</i> , <i>nT</i> , <i>x/y</i> , <i>operand1</i> usw.)	Sonstige Syntax-Elemente: Siehe Abschnitt Feldpositionierung , Text , Attributzuweisung weiter unten.

Statement-Parameter

Parameter, die mit dem INPUT-Statement angegeben werden können:		Spezifikation
		S=auf Statement-Ebene
		E=auf Element-Ebene
AD	Attribute Definition	SE
AL	Alphanumeric Length for Output	SE
CD	Color Definition	SE
CV	Control Variable	SE
DF	Date Format	SE
DL	Display Length for Output	SE
DY	Dynamic Attributes	SE
EM	Edit Mask	SE
EMU	Unicode Edit Mask	E
FL	Floating Point Mantissa Length	SE
HE	Helproutine	SE
IP	Input Prompting Text	SE
LS	Line Size	S
MC	Multiple-Value Field Count	S
MS	Manual Skip	S
NL	Numeric Length for Output	SE
PC	Periodic Group Count	S
PM	Print Mode *	SE
PS	Page Size **	S
SB	Selection Box	E
SG	Sign Position	SE

Parameter, die mit dem INPUT-Statement angegeben werden können:		Spezifikation
		S=auf Statement-Ebene
		E=auf Element-Ebene
ZP	Zero Printing	SE

* Der Session-Parameter PM darf nicht bei Textkonstanten angegeben werden.

** Die Einstellung des Session-Parameters PS bleibt unberücksichtigt, wenn die Anzahl der Ausprägungen eines Arrays den im Parameter PS angegebenen Wert überschreitet.

Beschreibungen der einzelnen Session-Parameter sind in der *Parameter-Referenz* enthalten.

WITH TEXT-Option

```
[WITH] TEXT { * operand1
              operand2 } [(attributes)][,operand3] ... 7
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate										Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S			N	P	I	B*								ja	ja
<i>operand2</i>	C	S			A											ja	ja
<i>operand3</i>	C	S			A	N	P	I	F	B	D	T	L			ja	ja

* Format B von *operand1* kann nur mit einer Länge kleiner gleich 4 benutzt werden.

Diese Option dient dazu, Text anzugeben, der in der Meldungszeile ausgegeben werden soll. In der Regel handelt es sich dabei um eine Meldung, was auf dem jeweiligen Schirm getan werden soll bzw. wie eine falsche Eingabe korrigiert werden soll.

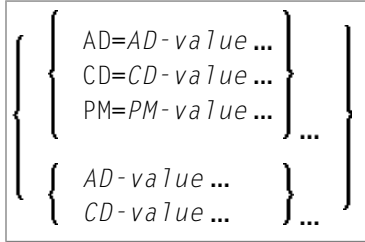
Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Meldungstext aus der Natural-Fehlermeldungsdatei:</p> <p>Als <i>operand1</i> geben Sie eine Natural-Fehlernummer an. Natural liest dann die entsprechende Fehlermeldung von der Natural- Fehlermeldungsdatei.</p> <p>Es können entweder benutzerdefinierte Meldungen oder Natural-Systemmeldungen gelesen werden.</p> <ul style="list-style-type: none"> ■ Wenn Sie einen positiven Wert von bis zu vier Ziffern (zum Beispiel: 0954) angeben, werden benutzerdefinierte Meldungen gelesen.
-----------------	--

	<p>■ Wenn Sie einen negativen Wert von bis zu vier Ziffern (zum Beispiel -0954) angeben, werden Natural-Systemmeldungen gelesen.</p> <p>Siehe auch REINPUT-Statement, <i>Beispiel 4 – WITH TEXT-Optionen</i>.</p> <p>Natural-Fehlermeldungsdateien werden mit der SYSERR-Utility erstellt und gepflegt.</p> <p>Die Natural-Fehlermeldungen sind in der Messages and Codes-Dokumentation enthalten.</p>
<i>operand2</i>	<p>Eigener Meldungstext:</p> <p>Als <i>operand2</i> geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll.</p> <p>Siehe REINPUT-Statement, <i>Beispiel 4 – WITH TEXT-Optionen</i>.</p>
<i>attributes</i>	<p>Ausgabeattribute:</p> <p>Als <i>attributes</i> können Sie <i>operand1</i> oder <i>operand2</i> bestimmte Anzeige- und Farbattribute zuordnen.</p> <p>Diese Attribute und die benutzbare Syntax sind im Abschnitt <i>Ausgabeattribute</i> weiter unten beschrieben.</p>
<i>operand3</i>	<p>Dynamische Meldungstext-Komponente:</p> <p><i>operand3</i> kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden.</p> <p>Der entweder mit <i>operand1</i> oder <i>operand2</i> angegebene Wert dient dazu, einen Teil der Meldung zu ersetzen und dynamisch zu generieren.</p> <p>Innerhalb der Fehlermeldung dient die Notation <i>:n</i>: zur Referenzierung von <i>operand3</i>, wobei <i>n</i> die Ausprägung (1 – 7) von <i>operand3</i> darstellt.</p> <p>Siehe REINPUT-Statement, <i>Beispiel 4 - WITH TEXT-Optionen</i>.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wird <i>operand3</i> mehrmals angegeben, müssen diese Operanden mit einem Komma voneinander getrennt werden. Falls das Komma als Dezimalzeichen verwendet wird (wie mit dem Session-Parameter DC definiert) und es sich bei <i>operand3</i> um numerische Konstanten handelt, setzen Sie Leerzeichen vor und nach dem Komma, damit es nicht als Dezimalkomma missinterpretiert wird. 2. Alternativ können mehrere Ausprägungen von <i>operand3</i> auch mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) voneinander getrennt werden; dies geht jedoch nicht bei ID=/ (Schrägstrich), da der Schrägstrich in der Syntax des INPUT-Statements eine andere Bedeutung hat. 3. Nicht signifikante Nullen oder Leerzeichen werden aus dem Feldwert entfernt, bevor er in einer Meldung angezeigt wird.

Ausgabeattribute

attributes gibt die für die Text-Anzeige zu benutzenden Ausgabe-Attribute an. Es gibt die folgenden Attribute:



Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD - Attribute Definition, Abschnitt Feldanzeige*
- *CD - Color Definition*
- *PM - Print Mode*



Anmerkung: Der Compiler akzeptiert tatsächlich mehr als einen Attributwert für ein Ausgabefeld. Zum Beispiel können Sie Folgendes angeben: AD=BDI. In solch einem Fall gilt allerdings nur der letzte Wert. Im hier gezeigten Beispiel erhält nur der Wert I Gültigkeit, und das Ausgabefeld wird intensiviert (hell hervorgehoben) angezeigt.

MARK-Option

Mit `MARK POSITION` können Sie bewirken, dass der Cursor in einem beliebigen, nicht geschützten Feld auf dem Bildschirm platziert wird. Zusätzlich können Sie die Position des Cursors innerhalb dieses Feldes bestimmen.

Standardmäßig, das heisst, wenn Sie die `MARK POSITION` weglassen, wird der Cursor an den Anfang des ersten, nicht geschützten Feldes positioniert.

```
MARK [POSITION operand4 [IN]] [FIELD] { operand1
                                         *fieldname }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand4</i>	C S	N P I	ja	ja
<i>operand1</i>	C S A	N P I	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Feldnummer für Referenzierung:</p> <p><i>operand1</i> gibt die Nummer des Feldes an, in dem der Cursor platziert werden soll:</p> <p>Jedem in einem INPUT-Statement angegebenen Feldattribut AD=A oder AD=M (d.h. ein ungeschütztes Feld) wird eine Feldreferenz- Nummer zugewiesen. Der Startwert ist 1.</p>
<i>*fieldname</i>	<p>Feldname für Referenzierung:</p> <p>Anstelle der Feldnummer können Sie den Feldnamen angeben, um den Cursor in ein bestimmtes Feld zu platzieren. Dazu verwenden Sie die Notation <i>*fieldname</i>.</p>
<i>operand4</i>	<p>Cursorposition im referenzierten Feld:</p> <p>Mit MARK POSITION können Sie den Cursor an eine bestimmte Stelle — die Sie mit <i>operand4</i> angeben — innerhalb des mit <i>operand1</i> oder <i>*fieldname</i> angegebenen Feldes platzieren.</p> <p><i>operand4</i> darf keine Dezimalstellen enthalten.</p>

Beispiele:

```
MARK #NUMBER /* Field number
MARK 3 /* Third map field
MARK *#FIELD1 /* Map field
MARK POSITION 3 IN #NUMBER /* Third character in field number
```

Siehe auch [Beispiel 3 — INPUT-Statement mit MARK POSITION-Option](#) weiter unten.

ALARM-Option

Diese Option bewirkt, dass der Warnton des Terminals ausgelöst wird, wenn das INPUT-Statement ausgeführt wird. Voraussetzung ist, dass die verwendete Terminal-Hardware dies ermöglicht.

```
[[[AND] [SOUND] ALARM]
```

Text vor einem Feld

Ist der Session-Parameter IP nicht auf IP=OFF gesetzt, so wird der jeweilige Feldname vor dem Feldwert (Forms-Modus) oder als Aufforderung, ein Feld auszuwählen, (Keyword/Delimiter-Modus) angezeigt. Wenn Sie vor dem Feld ' - ' angeben, wird der Feldname nicht angezeigt; wenn Sie einen 'text' angeben, wird dieser statt des Feldnamens angezeigt.

Feldpositionierung, Text, Attributzuweisung

nX	'text' [(attributes)]	*IN	{operand1 [(parameter(s))]}
nT	'c'(n) [(attributes)]	*OUT	
x/y	'-'	*OUTIN	
	'='		
	'/ ...		

Verschiedene Notationen stehen zur Feldpositionierung, Texterstellung und Attributzuweisung zur Verfügung.

nX	Leerstellen zwischen Spalten: Fügt zwischen den Feldern n Leerstellen ein.						
nT	Tabulator für Feldausgabe: Setzt einen Tabulator, d.h. die Ausgabe eines Feldes beginnt ab Spalte n .						
x/y	Ausgabeposition für nächstes Feld: Gibt das nachfolgende Element in Zeile x ab Spalte y aus. y darf nicht 0 sein. Rückwärtspositionierung in derselben Zeile ist nicht möglich.						
'text'	Textausgabe: In Apostrophen angegebener Text wird schreibgeschützt ausgegeben. Siehe auch den Abschnitt <i>Text-Notation</i> , Unterabschnitt <i>Mit einem Statement zu benutzenden Text definieren</i> .						

'c' (n)	<p>Wiederholungszeichen:</p> <p>Wie 'text'. Aber das Zeichen c (character) wird n-mal ausgegeben. n darf 1 – 132 sein. Siehe auch den Abschnitt <i>Text-Notation</i>, Unterabschnitt <i>Vor einem Feldwert n mal anzuzeigendes Zeichen definieren..</i></p>						
attributes	<p>Ausgabeattribute:</p> <p>Dient zum Setzen der Anzeigeattribute. Siehe <i>Attribute</i> weiter unten.</p>						
'	<p>Unterdrückung des Feldnamens:</p> <p>Die Notation ' - ' unmittelbar vor einem Feld bewirkt, dass die Anzeige des Feldnamens vor dem Feld unterdrückt wird.</p> <p>Anmerkung: Eine Textkette vor einem Feld ersetzt den Feldnamen als Eingabeaufforderungstext.</p>						
'='	<p>Feldüberschrift-Ausgabe:</p> <p>Die Notation '= ' unmittelbar vor einem Feld bewirkt, dass unmittelbar vor dem Feldwert die Feldüberschrift ausgegeben wird.</p>						
'/	<p>Zeilenvorschub:</p> <p>Ein Schrägstrich '/ ' zwischen zwei Feldern oder Textelementen bewirkt einen Zeilenvorschub, d.h. das nachfolgende Element wird in der nächsten Zeile ausgegeben.</p> <p>Felder können als reine Eingabefelder (Session-Parameter AD=A), reine Ausgabefelder (AD=0) oder als modifizierbare Ausgabefelder (AD=M) definiert werden. Standardmäßig gilt AD=A. Felder, die mit AD=A oder AD=M definiert sind, werden als ungeschützte Felder ausgegeben, d.h. der Benutzer hat die Möglichkeit, Daten in diese Felder einzugeben.</p> <p>Bei TTY-Geräten beansprucht die Ausgabe modifizierbarer Felder die doppelte Feldgröße (einmal für Ausgabe und einmal für Eingabe), damit ein neuer Wert eingegeben werden kann. Bei TTY-Bildschirmen beginnt ein Eingabefeld (AD=A oder AD=M), dessen Wert bei der Eingabe nicht angezeigt wird (Feldattribut N), immer in einer neuen Zeile.</p> <p>Beispiel:</p> <pre data-bbox="342 1541 1299 1608">INPUT #A (AD=A) #B (AD=0) #C (AD=M)</pre> <p>#A ist ein Eingabefeld (ungeschützt), in das ein Wert eingegeben werden kann.</p> <p>#B ist ein Ausgabefeld (schreibgeschützt), dessen angezeigter Wert nicht überschrieben werden kann.</p> <p>#C ist ein Feld, dessen ausgegebener Wert verändert werden kann, indem er durch einen neuen Wert überschrieben wird.</p>						

<p>*IN, *OUT und *OUTIN</p>	<p>Feldattribute:</p> <p>Entspricht den Feldattributen, die mit dem Session-Parameter AD gesetzt werden können: AD=A, AD=0 bzw. AD=M.</p> <p>Anmerkung: Wenn eine nicht änderbare Systemvariable in einem INPUT-Statement benutzt wird, wird der Wert als ein reines Ausgabefeld AD=0 oder *OUT-Attribut angezeigt.</p>	
<p><i>operand1</i></p>	<p>Zu verwendende Felder:</p> <p>Als <i>operand1</i> geben Sie das zu verwendende Feld an. Sie können Datenbankfelder oder Benutzervariablen angeben.</p> <p>Natural überträgt den Inhalt eines Feldes direkt vom Datenbereich an das INPUT-Statement; eine MOVE-Operation ist hierzu nicht erforderlich.</p> <p>Ändert sich der Wert eines Datenbankfeldes aufgrund einer INPUT-Verarbeitung, so betrifft dies nur den Feldwert im Datenbereich. Um den Wert eines Feldes auf der Datenbank zu ändern, sind entsprechende UPDATE- bzw. STORE-Statements erforderlich.</p> <p>Wird in einem INPUT-Statement der Name einer Gruppe von Datenbankfeldern referenziert, so werden alle in der Gruppe enthaltenen Felder einzeln als Eingabefelder verwendet.</p> <p>Wird ein Bereich von Ausprägungen eines Arrays referenziert, so wird jede Ausprägung einzeln als Eingabefeld verarbeitet; allerdings wird nur der ersten Ausprägung ein Text oder der Feldname vorangestellt.</p> <p>Auf Großrechnern können keine Arrays mit Bereichen angegeben werden, die es ermöglichen, zur Ausführungszeit die Anzahl der Ausprägungen zu variieren.</p>	

<i>parameter(s)</i>	<p>Statement-Parameter:</p> <p>Unmittelbar nach <i>operand1</i> können Sie in Klammern einen oder mehrere Parameter angeben (siehe Tabelle Statement-Parameter und folgendes Beispiel).</p> <p>Diese Parameter haben dann für das jeweilige Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- (im Reporting Mode) oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Die hier gültigen Parameter-Einstellungen kommen nur für Variablenfelder in Betracht, haben aber keine Auswirkung auf Textkonstanten. Wenn Sie Feldattribute für eine Textkonstante setzen möchten, müssen Sie explizit für dieses Element gesetzt werden.</p> <p>Informationen zu den einzelnen Parametern entnehmen Sie der Tabelle im Abschnitt Statement-Parameter.</p> <p>Anmerkung: Ist für ein Datenbankfeld eine Editiermaske definiert, so wird der Editiermasken-Parameter EM dynamisch im entsprechenden DDM referenziert. Editiermasken können für Eingabe- wie für Ausgabefelder angegeben werden. Wird für ein Eingabefeld eine Editiermaske definiert, müssen die Daten in Einklang mit der Editiermasken-Definition eingegeben werden.</p>
---------------------	---

Ausgabeattribute

Die folgenden Attribute können verwendet werden:

<table style="border: none; margin: auto;"> <tr> <td style="border: none; padding: 5px;">[AD=]</td> <td style="border: none; padding: 5px;">{</td> <td style="border: none; padding: 5px;">B</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">[CD=]</td> <td style="border: none; padding: 5px;">{</td> <td style="border: none; padding: 5px;">L</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">[PM=]</td> <td style="border: none; padding: 5px;">{</td> <td style="border: none; padding: 5px;">C</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">D</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">I</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">N</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">V</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">TU</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">YE</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">N</td> <td style="border: none; padding: 5px;">}</td> <td style="border: none; padding: 5px;"> </td> <td style="border: none; padding: 5px;">1</td> <td style="border: none; padding: 5px;">2</td> <td style="border: none; padding: 5px;">3</td> </tr> </table>	[AD=]	{	B	}		[CD=]	{	L	}		[PM=]	{	C	}		D	}		I	}		N	}		V	}		TU	}		YE	}		N	}		1	2	3
[AD=]	{	B	}		[CD=]	{	L	}		[PM=]	{	C	}		D	}		I	}		N	}		V	}		TU	}		YE	}		N	}		1	2	3	

1. Anzeigeattribute - siehe Session-Parameter AD in der *Parameter-Referenz*.
2. Farbattribute - siehe Session-Parameter CD in der *Parameter-Referenz*.
3. Attribute für Druck-Modus - siehe Session-Parameter PM in der *Parameter-Referenz*.

Beispiele — Verwendung von Syntax 1

- Beispiel 1 — INPUT-Statement
- Beispiel 2 — INPUT-Statement mit DEFINE WINDOW-Statement
- Beispiel 3 — INPUT-Statement mit MARK POSITION-Option

Beispiel 1 — INPUT-Statement

```

** Example 'IPTEX1': INPUT
*****
DEFINE DATA LOCAL
1 #FNC (A1)
END-DEFINE
*
INPUT 10X 'SELECTION MENU FOR EMPLOYEES SYSTEM' /
      10X '-' (35) //
      10X 'ADD      (A)' /
      10X 'UPDATE   (U)' /
      10X 'DELETE   (D)' /
      10X 'STOP     (.)' //
      10X 'PLEASE ENTER FUNCTION: ' #FNC
*
DECIDE ON EVERY VALUE OF #FNC
  VALUE 'A' /* invoke the object containing the add function here
    WRITE 'Add function selected.'
  VALUE 'U' /* invoke the object containing the update function here
    WRITE 'Update function selected.'
  VALUE 'D' /* invoke the object containing the delete function here
    WRITE 'Delete function selected.'
  VALUE '.'
    STOP
  NONE
    REINPUT 'Please enter a valid function.' MARK *#FNC
END-DECIDE
*
END

```

Ausgabe des Programms IPTEX1:

```
SELECTION MENU FOR EMPLOYEES SYSTEM
-----
ADD      (A)
UPDATE  (U)
DELETE  (D)
STOP    (.)

PLEASE ENTER FUNCTION:
```

Beispiel 2 — INPUT-Statement mit DEFINE WINDOW-Statement

```
** Example 'INPEX1': INPUT (with DEFINE WINDOW statement)
*****
DEFINE DATA LOCAL
1 #STRING (A15)
END-DEFINE
*
DEFINE WINDOW WIND1
  SIZE 10 * 40
  BASE 5 / 10
  FRAMED ON POSITION TEXT
*
INPUT WINDOW='WIND1'
  'PLEASE ENTER HERE:' / #STRING
*
END
```

Ausgabe des Programms INPEX1:

```
+-----Top+
! PLEASE ENTER HERE:      !
! #STRING                  !
!                          !
!                          !
!                          !
!                          !
!                          !
!                          !
!                          !
+-----Bottom+
```

Beispiel 3 — INPUT-Statement mit MARK POSITION-Option

```
** Example 'INPEX2': INPUT (with POSITION)
*****
DEFINE DATA LOCAL
1 #START (A30)
END-DEFINE
*
ASSIGN #START = 'EXAM_'
*
INPUT (AD=M) MARK POSITION 5 IN *#START
      / 'PLEASE COMPLETE START VALUE FOR SEARCH'
      / 5X #START
END
```

Ausgabe des Programms INPEX2:

```
PLEASE COMPLETE START VALUE FOR SEARCH
#START EXAM[]
```


76 INPUT-Syntax 2 — Verwendung einer vordefinierten

Eingabemaske

▪ INPUT USING MAP ohne Parameterliste	492
▪ Im Programm definierte Eingabefelder	493
▪ INPUT Syntax 2 — Beschreibung	493
▪ INPUT-Statement unter Nicht-Screen-Modi	495
▪ Eingabedaten aus dem Natural-Stack	496

Diese Form des INPUT-Statements wird benutzt, wenn bei der Eingabeverarbeitung eine mit dem Natural-Map-Editor erstellte Eingabemaske (Map) verwendet werden soll.

Hierbei gibt es zwei Möglichkeiten:

- das Programm enthält keine Parameterliste
- das Programm enthält eine Parameterliste (*operand1*).

```
INPUT [WINDOW='window-name'] [WITH-TEXT-option]
[MARK-option]
[ALARM-option]
[USING] MAP map-name [NO ERASE]
[
    operand1 ...
    NO PARAMETER
]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

INPUT USING MAP ohne Parameterliste

Die folgenden Anforderungen müssen bei der Benutzung von INPUT USING MAP ohne Parameterliste erfüllt werden:

- Der *map-name* muss als alphanumerische Konstante (maximal 8 Zeichen lang) angegeben werden.
- Die verwendete Map muss bereits erstellt sein, bevor das Programm, das sie referenziert, kompiliert werden kann.
- Die Feldnamen werden bei der Kompilierung dynamisch von der Map-Source übernommen. Die Feldnamen müssen in Map und Programm identisch sein.
- Zu diesem Zeitpunkt muss auf alle im INPUT-Statement referenzierten Felder zugegriffen werden können.
- Im Structured Mode müssen die Felder vorher definiert werden, und Datenbankfelder müssen durch Referenzierung der betreffenden Verarbeitungsschleife bzw. des betreffenden Views korrekt referenziert werden.
- Im Reporting Mode müssen Benutzervariablen in der Map neu definiert werden.
- Wird das Layout der Map verändert, müssen die die Map verwendenden Programme nicht neu katalogisiert werden. Wenn aber Array-Strukturen oder -Namen, Format/Länge von Feldern geändert oder Felder zur Map hinzugefügt bzw. aus ihr gelöscht werden, müssen die die Map verwendenden Programme neu katalogisiert werden.

- Die Map-Source muss bei der Programm-Kompilierung zur Verfügung stehen; sonst kann das `INPUT USING MAP`-Statement nicht kompiliert werden.



Anmerkung: Wollen Sie das Programm kompilieren, obwohl noch keine Map zur Verfügung steht, geben Sie `NO PARAMETER` an: das `INPUT USING MAP`-Statement kann dann kompiliert werden, auch wenn die Map noch nicht vorhanden ist.

Im Programm definierte Eingabefelder

Wenn Sie Namen der Eingabefelder (*operand1*) im Programm definieren, müssen diese nicht mit den für die Map verwendeten Feldnamen übereinstimmen.

Die Reihenfolge der Felder im Programm muss allerdings zur Reihenfolge der Felder in der Map passen. Hierbei ist zu beachten, dass der Map-Editor die in der Map definierten Felder in alphabetischer Reihenfolge der Feldnamen sortiert. Näheres hierzu finden Sie in der Map-Editor-Beschreibung in der *Editors*-Dokumentation.

Das Programm-Editor-Zeilenkommando `.I(mapname)` kann dazu verwendet werden, ein vollständiges `INPUT USING MAP`-Statement mit einer Parameterliste, die anhand der in der angegebenen Map definierten Felder generiert wird, zu erstellen.

Wird das Layout der Map verändert, muss das Programm nicht neu katalogisiert werden, es sei denn, in der Map werden Felder gelöscht, hinzugefügt oder umbenannt, Feldformate/-längen geändert oder Array-Strukturen modifiziert.

Bei der Ausführung prüft Natural, ob Format und Länge der Map-Felder in Einklang mit denen der Programm-Felder stehen. Ist dies nicht der Fall, wird eine entsprechende Fehlermeldung ausgegeben.

INPUT Syntax 2 — Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition		
<i>map-name</i>	C	S				A	U											ja	nein
<i>operand1</i>		S	A			A	U	N	P	I	F	B	D	T	L	C		ja	ja

Syntax-Element-Beschreibung:

<p>INPUT WINDOW='window-name'</p>	<p>Eingabe-Fenster: Siehe <i>Syntax 1</i> des INPUT-Statements.</p>
<p><i>WITH</i> <i>TEXT/MARK/ALARM-options</i></p>	<p>Optionen: Diese Optionen sind unter <i>Syntax 1</i> des INPUT-Statements beschrieben; siehe <i>WITH TEXT-Option, MARK-Option, ALARM-Option</i>.</p>
<p>USING MAP <i>map-name</i></p>	<p>Name der Eingabemaske: Mit der USING MAP-Klausel wird eine Map-Definition aufgerufen, die vorher mit dem Map-Editor in einer Natural-Systemdatei gespeichert worden ist. Der <i>map-name</i> kann als 1 bis 8 Zeichen lange alphanumerische Konstante oder in Form einer Benutzervariablen angegeben werden. Wird eine Variable verwendet, muss diese vorher definiert worden sein. Der Map-Name darf ein Und-Zeichen (&) enthalten; dies wird dann zur Ausführungszeit den aus einem Zeichen bestehenden Code ersetzt, der dem aktuellen Wert der Systemvariablen *LANGUAGE entspricht. Dadurch ist es möglich, verschiedensprachige Maps aufzurufen. Die Ausführung des INPUT-Statements löscht den bisherigen Bildschirminhalt, es sei denn, Sie verwenden eine NO ERASE-Klausel (siehe unten), wobei dann die Map den aktuellen Inhalt des Bildschirms überlagert.</p>
<p>NO ERASE</p>	<p>Überlagerte Anzeige: Siehe <i>Syntax 1</i> des INPUT-Statements.</p>
<p><i>operand1</i></p>	<p>Zu verwendende Felder: Es können Datenbankfelder oder Benutzervariablen angegeben werden, die jedoch alle vorher definiert worden sein müssen. Die Felder müssen in Anzahl, Reihenfolge Format und Länge und für (Arrays) in der Anzahl der Ausprägungen mit den referenzierten Map-Feldern übereinstimmen, da andernfalls ein Fehler generiert wird. Ändert sich der Wert eines Datenbankfeldes aufgrund einer INPUT-Verarbeitung, so betrifft dies nur den Feldwert im Datenbereich. Um den Wert eines Feldes auf der Datenbank zu ändern, sind entsprechende UPDATE- bzw. STORE-Statements erforderlich.</p>

INPUT-Statement unter Nicht-Screen-Modi

Sie können den Eingabemodus mit dem Session-Parameter `IM` ändern.

Forms-Modus

Im Forms-Modus (Profil/Session-Parameter `IM=F`) zeigt Natural den gesamten Ausgabertext des Map-Layouts Feld für Feld an, und zwar entsprechend der Positionierungsparameter. Dadurch kann der Benutzer Daten Feld für Feld eingeben. Wenn alle Daten eingegeben sind, wird eine Hardcopy-Ausgabe erzeugt, die genau dem Abbild auf dem Bildschirm entspricht.

Im Falle eines Fehlers kann der Benutzer das Terminalkommando `%R` eingeben und sodann das gesamte Formular erneut eingeben. Die Eingaben werden dann wie bei der ersten Ausführung des `INPUT`-Statements verarbeitet.

Keyword/Delimiter-Modus

In diesem Modus (Profil/Session-Parameter `IM=D`) können Daten 1.) unter *Verwendung von Schlüsselwörtern* oder 2.) *in Abhängigkeit von der Position/Reihenfolge der Felder* eingegeben werden, wie auf der folgenden Seite beschrieben.

1. Dateneingabe unter Verwendung von Schlüsselwörtern

Der Text, der im Forms-Modus vor einem Feld ausgegeben würde, wird im Keyword/Delimiter-Modus als Schlüsselwort (Keyword) verwendet, um das betreffende Feld zu identifizieren. Nach dem Schlüsselwort muss das Input-Assign-Zeichen stehen (siehe Session-Parameter `IA`) und unmittelbar danach die Eingabedaten.

Die Eingabedaten werden durch das mit dem Session-Parameter `ID` definierte Input-Delimiterzeichen voneinander getrennt. Auf ein Input-Assign-Zeichen folgende Leerzeichen werden als Eingabedaten interpretiert. Nach dem letzten Datenelement ist kein Input-Delimiterzeichen erforderlich.

Die Reihenfolge der Eingabedaten ist beliebig. Wird ein nicht im `INPUT`-Statement definiertes Schlüsselwort eingegeben, gibt Natural eine entsprechende Fehlermeldung aus. Es brauchen nicht alle Felder mit Eingabedaten gefüllt zu werden. Felder, in die nichts eingegeben wird, werden auf Leerzeichen (alphanumerische Felder) bzw. Null (numerische und hexadezimale Felder) gesetzt.

2. Dateneingabe in Abhängigkeit von der Position/Reihenfolge der Felder

Es können auch lediglich Eingabedaten, jeweils durch das gültige Input-Delimiter-Zeichen (`ID`-Parameter) voneinander getrennt, eingegeben werden. In diesem Fall muss die Reihenfolge der Eingabedaten der Reihenfolge der Eingabefelder im `INPUT`-Statement entsprechen.

Beide Eingabearten (1 und 2) können auch miteinander kombiniert werden. Solange keine Schlüsselwörter angegeben werden, ist die im `INPUT`-Statement angegebene Reihenfolge der Felder maßgeblich; wird ein Schlüsselwort angegeben, so werden die auf das Schlüsselwort folgenden Eingabedaten dem durch das Schlüsselwort identifizierten Feld zugeordnet. Folgen hierauf wieder Eingabedaten ohne Schlüsselwörter, so werden diese Daten den Feldern zugeordnet, die im `INPUT`-Statement auf das mit Schlüsselwort identifizierte Feld folgen.



Anmerkung: Ein Schlüsselwort und das dazugehörige Eingabefeld müssen sich auf derselben logischen Zeile befinden. Wenn die Länge beider zusammen die Zeilenlänge überschreitet, vergrößern Sie die Zeilenlänge (Session-Parameter `LS`) entsprechend, so dass Schlüsselwort und Feld in eine Zeile passen.

Im `Keyword/Delimiter`-Modus eingegebene Eingabedaten werden entsprechend der auch im `Screen`-Modus gültigen Regeln auf ihre Gültigkeit überprüft. Wird versucht, einen Wert einzugeben, der länger als das betreffende Feld ist, so gibt Natural eine entsprechende Fehlermeldung aus.

Werden Daten für ein `INPUT`-Statement im `Keyword/Delimiter`-Modus von einem gepufferten Terminal (Typ 3270) oder einer Workstation aus eingegeben, so müssen alle von einem bestimmten `INPUT`-Statement zu verarbeitenden Eingabedaten auf einem einzigen Schirm eingegeben werden; erst wenn alle Daten eingegeben sind, darf `FREIG` gedrückt werden.

Eingabedaten aus dem Natural-Stack

Daten, die mittels eines `FETCH`-, `RUN`- oder `STACK`-Statements im Natural-Stack abgelegt wurden, werden bei der Ausführung des nächsten `INPUT`-Statements als Eingabedaten verarbeitet.

Das `INPUT`-Statement verarbeitet die Daten in dem oben beschriebenen **Keyword/Delimiter-Modus**.

Felder, für die keine Eingabedaten zur Verfügung stehen, werden je nach Format mit Leerzeichen bzw. Nullen gefüllt. Sind mehr Daten als Eingabefelder vorhanden, so werden überschüssige Eingabedaten ignoriert.

Wenn ein Feld mit Daten aus dem Stack gefüllt wird, gelten die Feldattribute nicht für die Daten.

Über die Natural-Systemvariable `*DATA` können Sie erfahren, wieviele Datenelemente gegenwärtig im Natural-Stack zur Verfügung stehen.

77 INTERFACE

▪ Funktion	498
▪ Syntax-Beschreibung	499

```
INTERFACE interface-name
[ EXTERNAL]
[ID interface-GUID]
[property-definition]
[method-definition]
END-INTERFACE
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: CREATE OBJECT | DEFINE CLASS | INTERFACE | METHOD | PROPERTY | SEND METHOD

Gehört zur Funktionsgruppe: *Komponentenbasierte Programmierung*

Funktion

Bei der komponentenbasierten Programmierung ist ein Interface (eine Schnittstelle) eine Sammlung von Methoden und Eigenschaften, die semantisch zusammengehören und eine bestimmte Funktion einer Klasse darstellen.

Sie können ein oder mehrere Interfaces für eine Klasse definieren. Durch die Definition mehrerer Interfaces wird es Ihnen ermöglicht, Methoden nach ihrer Funktionsweise zu strukturieren/gruppieren, z.B. stellen Sie alle Methoden, die mit Dauerhaftigkeit (Laden, Speichern, Aktualisieren) zu tun haben, in ein Interface und die anderen Methoden in andere Interfaces.

Das INTERFACE-Statement dient zur Definition eines Interface. Es darf nur innerhalb eines Natural-Klassenmoduls verwendet werden und kann wie folgt definiert werden:

- innerhalb eines DEFINE CLASS-Statements. Diese Form wird verwendet, wenn das Interface nur in einer Klasse implementiert werden soll.
- in innerhalb der INTERFACE USING-Klausel des DEFINE CLASS-Statements enthaltenem Copycode. Diese Form wird verwendet, wenn das Interface in mehr als einer Klasse implementiert werden soll.

Die mit dem Interface verbundenen Eigenschaften und Methoden werden in den *Property-Definitionen* bzw. *Method-Definitionen* festgelegt.

Syntax-Beschreibung

<i>interface-name</i>	<p>Interface-Name:</p> <p>Dies ist der dem Interface zuzuweisende Name. Der Interface-Name kann maximal 32 Zeichen lang sein und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie im Abschnitt <i>Namenskonventionen für Benutzervariablen</i>). Er muss pro Klasse eindeutig und nicht mit dem Klassen-Namen identisch sein.</p> <p>Wenn das Interface von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Interface-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.</p>
EXTERNAL	<p>EXTERNAL-Klausel:</p> <p>Mit der EXTERNAL-Klausel wird angegeben, dass dieses Interface von der Klasse implementiert wird, es ursprünglich aber in einer anderen Klasse definiert ist. Diese Klausel ist nur relevant, wenn die Klasse mit DCOM registriert werden soll. Interfaces mit der EXTERNAL-Klausel werden ignoriert, wenn die Klasse mit DCOM registriert wird. Es wird angenommen, dass das Interface von der Klasse registriert ist, die es ursprünglich definiert hat.</p>
ID <i>interface-GUID</i>	<p>ID-Klausel:</p> <p>Mit der ID-Klausel wird dem Interface eine global eindeutige ID (globally unique ID) zugewiesen. Die Interface-GUID ist der Name einer GUID, die in einer Data Area definiert ist, die von der LOCAL-Klausel einbezogen wird. Die Interface-GUID ist eine (benannte) Alpha-Konstante. Eine GUID muss einem Interface zugewiesen werden, wenn die Klasse mit DCOM registriert werden soll.</p>
<i>property-definition</i>	<p>Property-Definition für das Interface:</p> <p>Die <i>property-definition</i> dient zur Definition von Eigenschaften für das Interface. Siehe <i>Property-Definition</i> weiter unten.</p>
<i>method-definition</i>	<p>Method-Definition für das Interface:</p> <p>Die <i>method-definition</i> dient zur Definition einer Methode für das Interface. Siehe <i>Method-Definition</i> weiter unten.</p>
END-INTERFACE	<p>Das für Natural reservierte Wort END-INTERFACE muss zum Beenden des INTERFACE-Statements benutzt werden.</p>

Property-Definition

Die Property-Definition dient zur Definition von Eigenschaften für das Interface.

```
PROPERTY property-name
  [(format-length/array-definition)]
  [ID dispatch-ID]
  [READONLY]
  [IS operand]
END-PROPERTY
```

Properties sind Attribute eines Objekts, das von Clients aufgerufen werden kann. Ein Objekt, das einen Angestellten darstellt, kann zum Beispiel eine Property mit Namen `Name` und eine andere mit Namen `Department` haben. Das Einlesen oder Ändern des Namens oder der Abteilung des Angestellten durch Aufruf der Property für dessen Namen oder dessen Abteilung ist viel einfacher für einen Client als eine Methode aufzurufen, die den Wert zurückgibt, und eine andere Methode aufzurufen, die den Wert ändert.

Jede Property benötigt eine Variable in der Object Data Area der Klasse, um dessen Wert zu speichern – dies wird als Objektdaten-Variable bezeichnet. Die Property-Definition dient dazu, diese Variable für den Zugriff von Clients freizugeben. Die Property-Definition legt den Namen und das Format der Property fest und verbindet sie mit der Objektdaten-Variable. Im einfachsten Fall übernimmt die Property den Namen und das Format der Objektdaten-Variable selbst. Es ist auch möglich, den Namen und das Format innerhalb bestimmter Grenzen zu überschreiben.

<i>property-name</i>	<p>Property-Name:</p> <p>Dies ist der der Property zuzuweisende Name. Der Property-Name kann maximal bis zu 32 Zeichen enthalten und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie unter <i>Namenskonventionen für Benutzervariablen</i>).</p> <p>Wenn die Property von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Property-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.</p>
<i>format-length/array-definition</i>	<p>Property-Format:</p> <p>Damit wird das Format der Property definiert, wie es von den Clients erkannt wird.</p> <p>Wenn <i>format-length/array-definition</i> weggelassen wird, wird <i>format-length</i> und <i>array-definition</i> aus der in der IS-Klausel zugewiesenen Objektdaten-Variable genommen.</p>

	<p>Wenn <i>format-length/array-definition</i> angegeben wird, muss diese Angabe unbedingt datenübertragungskompatibel sein, und zwar zu dem und von dem in <i>operand</i> in der <i>IS</i>-Klausel angegebenen Format der Objektdaten-Variable.</p> <p>Im Falle einer READONLY-Property braucht die Datenübertragungs-Kompatibilität ausschließlich in einer Richtung zu gelten: mit der Objektdaten-Variable als Ausgangsoperand und der Property als Zieloperand.</p> <p>Wenn eine Array-Definition angegeben wird, muss sie in den Dimensionen, Ausprägungen pro Dimension, Untergrenzen und Obergrenzen mit der Array-Definition der entsprechenden Objektdaten-Variable übereinstimmen. Dies kommt durch Spezifikation eines Sterns (*) für jede Dimension zum Ausdruck.</p>
ID <i>dispatch-ID</i>	<p>ID-Klausel:</p> <p>Mit der <i>ID</i>-Klausel wird einem Property ein bestimmter numerischer Bezeichner (Identifier) zugewiesen. Dieser Bezeichner (die sogenannte Dispatch-ID) ist nur relevant, wenn die Klasse mit DCOM registriert werden soll.</p> <p>Normalerweise weist Natural einem Property automatisch eine Dispatch-ID zu. Es ist nur dann erforderlich, für ein Property eine bestimmte Dispatch-ID explizit zu definieren, wenn das Property zu einem Interface mit einer EXTERNAL-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, das jedoch ursprünglich in einer anderen Klasse definiert wurde.) In diesem Fall ist die zu benutzende Dispatch-ID in der Regel durch die ursprüngliche Implementierung des Interfaces vorgegeben.</p> <p>Die Dispatch-ID ist eine positive Konstante, die nicht Null ist, im Format I4.</p>
READONLY	<p>Schreibschutz für Property-Wert:</p> <p>Wenn dieses Schlüsselwort angegeben wird, kann der Wert der Property nur gelesen und nicht gesetzt werden. Das in <i>operand</i> in der <i>IS</i>-Klausel angegebene Format der Objektdaten-Variable muss datenübertragungskompatibel mit dem in <i>format-length/array-definition</i> angegebenen Format sein. Es muss nicht datenübertragungskompatibel in der umgekehrten Richtung sein.</p> <p>Wenn das Schlüsselwort READONLY weggelassen wird, kann der Property-Wert sowohl gelesen als auch gesetzt werden.</p>
IS <i>operand</i>	<p>IS-Klausel:</p> <p>Der <i>operand</i> in der <i>IS</i>-Klausel weist eine Objektdaten-Variable als Adresse zu, um den Property-Wert zu speichern. Die zugewiesene Objektdaten-Variable muss nicht unbedingt eine</p>

	<p>Gruppe sein. Die Variable wird in normaler Operanden-Syntax referenziert. Dies bedeutet, dass wenn die Objektdaten-Variablen ein Array ist, sie mit Index-Notation referenziert werden muss. Nur die vollständige Indexbereichs-Notation und Sternchen-Notation ist zulässig.</p> <p>Die IS-Klausel sollte nicht verwendet werden, wenn das INTERFACE-Statement aus einem Copycode-Member übernommen und in mehreren Klassen wiederverwendet wird. Wenn Sie das INTERFACE-Statement nochmals verwenden möchten, müssen Sie die Objektdaten-Variablen in einem PROPERTY-Statement außerhalb des INTERFACE-Statements zuweisen.</p> <p>Wenn die IS-Klausel weggelassen wird, wird die Property mit der Objektdaten-Variablen mit demselben Namen wie die Property verknüpft. Wenn eine Variable mit diesem Namen nicht definiert ist, oder wenn es sich um eine Gruppe handelt, führt dies zu einem Syntax-Fehler.</p>
END-PROPERTY	Das für Natural reservierte Wort END-PROPERTY muss zum Beenden des Interfaces PROPERTY-Definition benutzt werden.

Beispiele

Angenommen die Object Data Area enthält die folgenden Daten-Definitionen:

```
1 Salary(p7.2)
1 SalaryHistory(p7.2/1:10)
```

Dann sind die folgenden Property-Definitionen erlaubt:

```
property Salary
end-property
property Pay is Salary
end-property
property Pay(P7.2) is Salary
end-property
property Pay(N7.2) is Salary
end-property
property SalaryHistory
end-property
property OldPay is SalaryHistory(*)
end-property
property OldPay is SalaryHistory(1:10)
end-property
property OldPay(P7.2/*) is SalaryHistory(1:10)
end-property
```

```
property OldPay(N7.2/*) is SalaryHistory(*)
end-property
```

Die folgenden Property-Definitionen sind nicht zulässig:

```
/* Not data transfer-compatible. */
property Pay(L) is Salary
end-property
/* Not data transfer-compatible. */
property OldPay(L/*) is SalaryHistory(*)
end-property
/* Not data transfer-compatible. */
property OldPay(L/1:10) is SalaryHistory(1:10)
end-property
/* Assigns an array to a scalar. */
property OldPay(P7.2) is SalaryHistory(1:10)
end-property
/* Takes only a sub-array. */
property OldPay(P7.2/3:5) is SalaryHistory(*)
end-property
/* Index specification omitted in ODA variable SalaryHistory. */
property OldPay is SalaryHistory
end-property
/* Only asterisk notation allowed in property format specification. */
property OldPay(P7.2/1:10) is SalaryHistory(*)
end-property
```

Method-Definition

Die Method-Definition dient zur Definition einer Methode für das Interface.

```
METHOD method-name
  [ID dispatch-ID]
  [IS subprogram-name]
  [ PARAMETER { USING parameter-data-area } ]...
  [data-definition...]
END-METHOD
```

Um das Interface in verschiedenen Klassen wiederverwenden zu können, übernehmen Sie die Interface-Definition aus einem Copycode und definieren Sie das Subprogramm hinter der Interface-Definition mit einem METHOD-Statement. Dann können Sie die Methode in verschiedenen Klassen anders implementieren.

<i>method-name</i>	<p>Method-Name:</p> <p>Dies ist der der Methode zuzuweisende Name. Der Methoden-Name kann maximal bis zu 32 Zeichen enthalten und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen entnehmen Sie dem Abschnitt <i>Namenskonventionen für Benutzervariablen</i>). Er muss pro Interface eindeutig sein.</p> <p>Wenn die Methode von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Methoden-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.</p>
ID <i>dispatch-ID</i>	<p>ID-Klausel:</p> <p>Mit der ID-Klausel wird einer Methode ein bestimmter numerischer Bezeichner (Identifizier) zugewiesen. Dieser Bezeichner (die sogenannte Dispatch-ID) ist nur relevant, wenn die Klasse mit DCOM registriert werden soll.</p> <p>Normalerweise weist Natural einer Methode automatisch eine Dispatch-ID zu. Es ist nur dann erforderlich, für eine Methode eine bestimmte Dispatch-ID explizit zu definieren, wenn die Methode zu einem Interface mit einer EXTERNAL-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, das jedoch ursprünglich in einer anderen Klasse definiert wurde.) In diesem Fall ist die zu benutzende Dispatch-ID in der Regel durch die ursprüngliche Implementierung des Interfaces vorgegeben.</p> <p>Die Dispatch-ID ist eine positive Konstante, die nicht Null ist, im Format I4.</p>
IS <i>subprogram-name</i>	<p>Name des Subprogramms:</p> <p>Dies ist der Name des die Methode implementierenden Subprogramms. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist <i>method-name</i> (wenn die IS-Klausel nicht angegeben wird).</p>
PARAMETER	<p>Parameter-Definition:</p> <p>Mit dieser Klausel werden die Parameter der Methode angegeben; sie hat dieselbe Syntax wie die PARAMETER-Klausel vom DEFINE DATA-Statement.</p> <p>Die Parameter müssen mit den Parametern übereinstimmen, die später bei der Implementierung des Subprogramms verwendet werden. Dies wird am besten durch Verwendung einer Parameter Data Area (PDA) gewährleistet.</p> <p>In der Parameter Data Area als BY VALUE markierte Parameter sind Eingabe-Parameter der Methode.</p> <p>Nicht als BY VALUE markierte Parameter werden referenziert (By Reference) übergeben und sind Eingabe-/Ausgabe-Parameter. Dies ist die Voreinstellung.</p> <p>Der als BY VALUE RESULT markierte erste Parameter wird als Rückgabewert für die Methode zurückgegeben. Wenn mehr als ein Parameter so markiert ist, werden die anderen als Eingabe-/Ausgabe-Parameter behandelt.</p>
END-METHOD	<p>Das für Natural reservierte Wort END-METHOD muss zum Beenden der METHOD-Definition für das Interface benutzt werden.</p>

78

LIMIT

▪ Funktion	506
▪ Syntax-Beschreibung	507
▪ Beispiele	507

LIMIT *n*

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: `ACCEPT/REJECT` | `AT BREAK` | `AT START OF DATA` | `AT END OF DATA` | `BACKOUT TRANSACTION` | `BEFORE BREAK PROCESSING` | `DELETE` | `END TRANSACTION` | `FIND` | `GET` | `GET SAME` | `GET TRANSACTION` | `HISTOGRAM` | `PASSW` | `PERFORM BREAK PROCESSING` | `READ` | `RETRY` | `STORE` | `UPDATE`

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement `LIMIT` dient dazu, die Anzahl der Durchläufe einer Verarbeitungsschleife, die mit einem `FIND`-, `READ`- oder `HISTOGRAM`-Statement initiiert wurde, zu begrenzen.

Das festgesetzte Limit gilt für alle nachfolgenden Verarbeitungsschleifen des Programms, bis es durch ein weiteres `LIMIT`-Statement außer Kraft gesetzt wird.

Das `LIMIT`-Statement gilt nicht für einzelne Statements, in denen ausdrücklich ein anderes Limit angegeben ist, zum Beispiel `FIND (n)`

Wenn das Limit erreicht ist, wird die Verarbeitung der betreffenden Schleife abgebrochen und eine entsprechende Meldung ausgegeben. Siehe auch Session-Parameter `LE`, der die Reaktion darauf festlegt, wann das Limit für die Verarbeitungsschleife überschritten wird.

Wird kein `LIMIT`-Statement verwendet, so gilt standardmäßig das mit dem Natural- Profilparameter `LT` bei der Natural-Installation festgesetzte globale Limit.

Zählweise

Um zu ermitteln, ob eine Verarbeitungsschleife das Limit erreicht hat, wird jeder mit der Schleife gelesene Datensatz gezählt; hierbei gilt:

- Ein Datensatz, der aufgrund der `WHERE`-Bedingung eines `FIND`- oder `READ`-Statements zurückgewiesen wird, wird nicht mitgezählt.
- Ein Datensatz, der aufgrund eines `ACCEPT/REJECT`-Statements zurückgewiesen wird, wird mitgezählt.

Syntax-Beschreibung

LIMIT <i>n</i>	<p>Limit-Angabe:</p> <p>Das Limit <i>n</i> muss als numerische Konstante angegeben werden und kann einen Wert von 0 bis 4294967295 (führende Nullen sind optional) haben.</p> <p>Ist das Limit auf Null (0) gesetzt, wird die Schleife nicht durchlaufen.</p>
-----------------------	--

Beispiele

- [Beispiel 1 — LIMIT-Statement](#)
- [Beispiel 2 — LIMIT-Statement \(gültig für zwei Datenbankschleifen\)](#)

Beispiel 1 — LIMIT-Statement

```

** Example 'LMTEX1': LIMIT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 4
*
READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
  DISPLAY NOTITLE
    NAME PERSONNEL-ID CITY *COUNTER
END-READ
*
END

```

Ausgabe des Programms LMTEX1:

NAME	PERSONNEL ID	CITY	CNT
BAKER	20016700	OAK BROOK	1
BAKER	30008042	DERBY	2
BALBIN	60000110	BARCELONA	3
BALL	30021845	DERBY	4

Beispiel 2 — LIMIT-Statement (gültig für zwei Datenbankschleifen)

```

** Example 'LMTEX2': LIMIT (valid for two database loops)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
LIMIT 3
*
FIND EMPLOY-VIEW WITH NAME > 'A'
  READ EMPLOY-VIEW BY NAME STARTING FROM 'BAKER'
    DISPLAY NOTITLE 'CNT(0100)' *COUNTER(0100)
                  'CNT(0110)' *COUNTER(0110)
  END-READ
END-FIND
*
END
    
```

Ausgabe des Programms LMTEX2:

CNT(0100)	CNT(0110)
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

79 LOOP

▪ Funktion	510
▪ Einschränkung	510
▪ Syntax-Beschreibung	511
▪ Beispiele	511

[CLOSE] LOOP [(r)]

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `LOOP` dient dazu, eine Verarbeitungsschleife zu schließen. Es bewirkt, dass der aktuelle Schleifendurchlauf beendet wird und die Kontrolle wieder an den Anfang der Schleife übergeben wird.

Sobald die Verarbeitungsschleife, auf die sich das `LOOP`-Statement bezieht, beendet ist (d.h. sobald alle Datensätze verarbeitet und alle Schleifendurchläufe ausgeführt sind), wird die Verarbeitung mit dem auf das `LOOP`-Statement folgenden Statement fortgesetzt.

Referenzierung von Datenbankvariablen

Neben dem Schließen der Schleife(n) bewirkt das `LOOP`-Statement, dass alle Referenzierungen von Feldern, die in `FIND`-, `FIND FIRST`-, `FIND UNIQUE`-, `READ`- und `GET`-Statements innerhalb der geschlossenen Schleife(n) verwendet werden, eliminiert werden.

Ein Feld, das in einem View enthalten ist, kann auch außerhalb einer mit `LOOP` geschlossenen Schleife referenziert werden, und zwar indem bei der Referenzierung der View-Name angegeben wird.

Einschränkung

- Dieses Statement gilt nur für Reporting Mode.
- Ein `LOOP`-Statement darf nicht an eine logische Bedingung wie etwa ein `IF`- oder `AT BREAK`-Statement geknüpft werden.

Syntax-Beschreibung

LOOP (r)	Statement-Referenzierung: Sollen mehrere Schleifen geschlossen werden, so kann ein bestimmtes Statement per Statement-Label oder Sourcecode- Zeilennummer referenziert werden (Notation (r)); in diesem Falle werden durch das LOOP-Statement dann die referenzierte Verarbeitungsschleife sowie alle innerhalb der referenzierten Schleife befindlichen Schleifen geschlossen.
-----------------	---



Anmerkung: Im Reporting Mode werden durch ein **END**-Statement alle noch aktiven Verarbeitungsschleifen, die noch nicht explizit durch ein **LOOP**-Statement beendet wurden, automatisch geschlossen.

Beispiele

Beispiel 1

```
0010 FIND ...
0020   READ ...
0030     READ ...
0040 LOOP (0010)   /* closes all loops
```

Beispiel 2

```
0010 FIND ...
0020   READ ...
0030     READ ...
0040       LOOP   /* closes loop initiated on line 0030
0050     LOOP   /* closes loop initiated on line 0020
0060   LOOP   /* closes loop initiated on line 0010
```


80

METHOD

▪ Funktion	514
▪ Syntax-Beschreibung	514
▪ Beispiel	515

```

METHOD method-name
  OF [INTERFACE] interface-name
  IS subprogram-name
END-METHOD

```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [CREATE OBJECT](#) | [DEFINE CLASS](#) | [INTERFACE](#) | [PROPERTY](#) | [SEND METHOD](#)

Gehört zur Funktionsgruppe: [Komponentenbasierte Programmierung](#)

Funktion

Das METHOD-Statement weist ein Subprogramm als Implementierung zu einer Methode zu, und zwar außerhalb einer Interface-Definition. Es wird verwendet, wenn die betreffende Interface-Definition aus einem Copycode übernommen wird und auf eine klassenspezifische Weise implementiert werden soll.

Das METHOD-Statement kann nur innerhalb eines DEFINE CLASS-Statements und im Anschluss an die Interface-Definition verwendet werden. Die angegebenen Interface- und Methoden-Namen müssen innerhalb der Interface-Definitionen des DEFINE CLASS-Statements festgelegt werden.

Syntax-Beschreibung

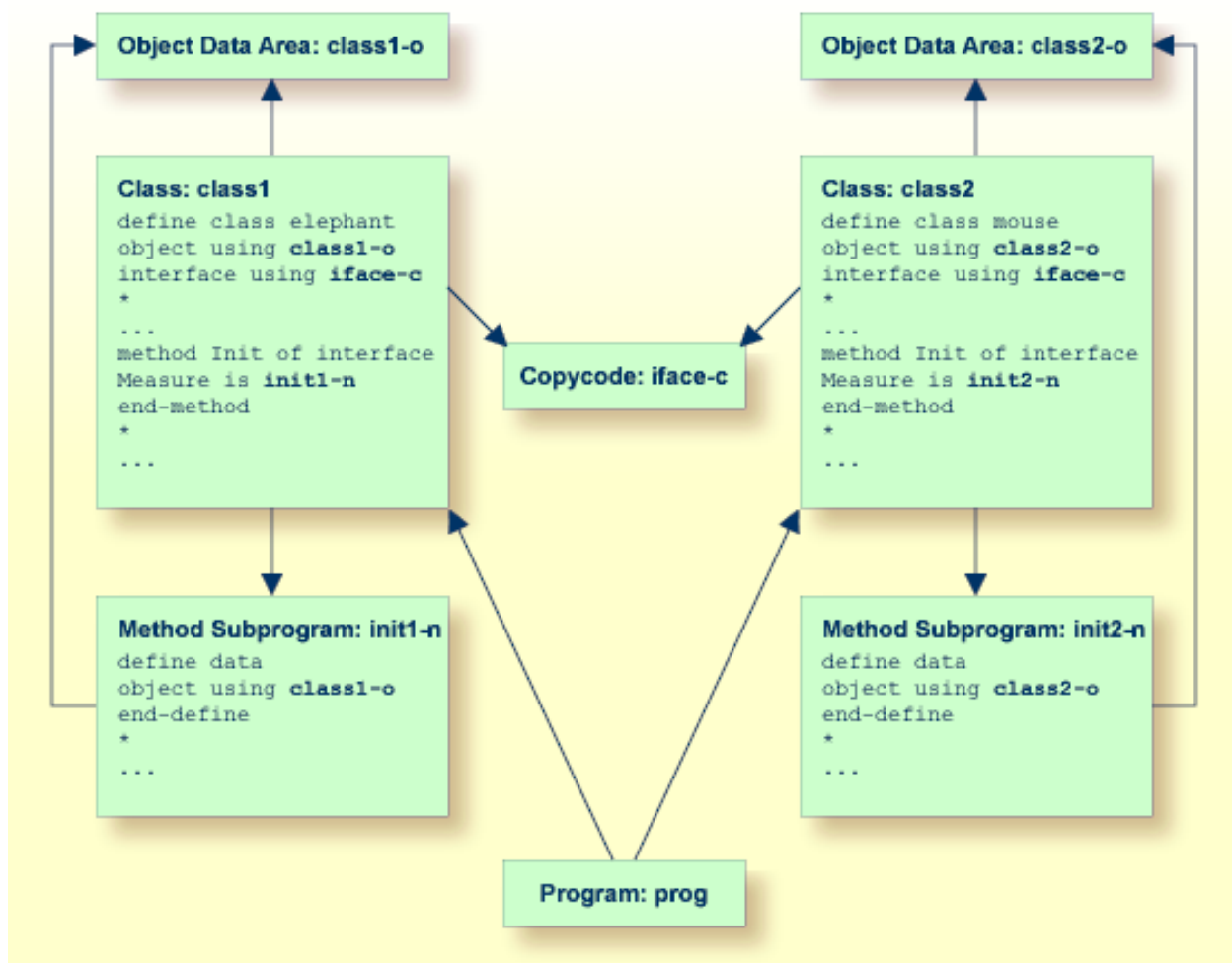
<i>method-name</i>	Method-Name: Dies ist der der Methode zugewiesene Name.
OF <i>interface-name</i>	Interface-Name: Dies ist der dem Interface zugewiesene Name.
IS <i>subprogram-name</i>	Name des Subprogramms: Dies ist der Name des Subprogramms, das die Methode implementiert. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist <i>method-name</i> (wenn die IS-Klausel nicht angegeben wird).
END-METHOD	Das für Natural reservierte Wort END-METHOD muss zum Beenden des METHOD-Statements benutzt werden.

Beispiel

Das folgende Beispiel zeigt, wie dasselbe Interface in zwei Klassen unterschiedlich implementiert wird und wie das `PROPERTY`-Statement und das `METHOD`-Statement zu diesem Zweck verwendet werden.

Das Interface `Measure` wird im Copycode `iface-c` definiert. Die Klassen `Elephant` und `Mouse` implementieren beide das Interface `Measure`. Deshalb beinhalten sie beide den Copycode `iface-c`. Die Klassen implementieren aber die Property `Height` mittels verschiedener Variablen von ihren betreffenden Object Data Areas, und sie implementieren die Methode `Init` mit unterschiedlichen Subprogrammen. Sie verwenden das Statement `PROPERTY`, um die ausgewählte Data Area-Variable der Property zuzuweisen, und das Statement `METHOD`, um das ausgewählte Subprogramm der Methode zuzuweisen.

Jetzt kann das Programm `prog` Objekte beider Klassen erstellen und sie mittels derselben Methode `Init` initialisieren, wobei die Schritte der Initialisierung der betreffenden Klassen-Implementierung überlassen werden.



Im folgenden finden Sie den vollständigen Inhalt der im vorstehenden Beispiel verwendeten Natural-Module:

Copycode: iface-c

```

interface Measure
*
property Height(p5.2)
end-property
*
property Weight(i4)
end-property
*
method Init
end-method
*
end-interface
  
```


Class: class1

```

define class elephant
object using class1-o
interface using iface-c
*
property Height of interface Measure is height
end-property
*
property Weight of interface Measure is weight
end-property
*
method Init of interface Measure is init1-n
end-method
*
end-class
end

```

LDA Object Data: class1-o

```

*   *** Top of Data Area ***
1  HEIGHT                P 5.2
1  WEIGHT                 I 2
*   *** End of Data Area ***

```

Method Subprogram: init1-n

```

define data
object using class1-o
end-define
*
height := 17.3
weight := 120
*
end

```

Class: class2

```

define class mouse
object using class2-o
interface using iface-c
*
property Height of interface Measure is size
end-property
*
property Weight of interface Measure is weight

```

METHOD

```
end-property
*
method Init of interface Measure is init2-n
end-method
*
end-class
end
```

LDA Object Data: class2-o

```
*   *** Top of Data Area ***
1  SIZE                P 3.2
1  WEIGHT              I 1
*   *** End of Data Area ***
```

Method Subprogram: init2-n

```
define data
object using class2-o
end-define
*
size := 1.24
weight := 2
*
end
```

Program: prog

```
define data local
1 #o handle of object
1 #height(p5.2)
1 #weight(i4)
end-define
*
create object #o of class 'Elephant'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
create object #o of class 'Mouse'
send "Init" to #o
#height := #o.Height
#weight := #o.Weight
write #height #weight
*
end
```

81 MOVE

▪ Funktion	520
▪ Syntax-Beschreibung	521
▪ Beispiele	534

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das Statement `MOVE` dient dazu, den Wert eines Operanden in einen oder mehrere andere Operanden (Felder oder Arrays) zu übertragen.

Ein `MOVE`-Statement mit mehreren Zieloperanden ist mit den betreffenden einzelnen `MOVE`-Statements identisch:

```
MOVE #SOURCE TO #TARGET1 #TARGET2
```

ist identisch mit:

```
MOVE #SOURCE TO #TARGET1
MOVE #SOURCE TO #TARGET2
```

Beispiel:

```
DEFINE DATA LOCAL
1 #ARRAY(I4/1:3) INIT <3,0,9>
1 #INDEX(I4)
1 #RESULT(I4)
END-DEFINE
*
#INDEX := 1
MOVE #ARRAY(#INDEX) TO #INDEX      /* #INDEX is 3
                        #RESULT     /* #RESULT is 9
*
#INDEX := 2
MOVE #ARRAY(#INDEX) TO #INDEX      /* #INDEX is 0
                        #ARRAY(3)  /* returns run time error NAT1316
```

Ist *operand2* eine dynamische Variable, kann ihre Länge mit der `MOVE`-Operation geändert werden. Die aktuelle Länge einer dynamischen Variable kann mittels der Systemvariable `*LENGTH` bestimmt werden. Allgemeine Informationen zu dynamischen Variablen siehe den Abschnitt *Dynamische und große Variablen benutzen im Leitfaden zur Programmierung*.

Hat *operand2* das Format C, kann *operand1* auch als (*parameter*) angegeben werden. Es gibt die folgenden gültigen Parameter:

Parameter, die mit dem MOVE-Statement angegeben werden können:		Spezifikation:
		S = auf Statement-Ebene
		E = auf Element-Ebene
AD	Attribute Definition	SE
CD	Color Definition	S

Weitere Informationen zur Datenübertragungs-Kompatibilität und den Regeln für die Datenübertragung finden Sie im Abschnitt *Datenübertragung* im *Leitfaden zur Programmierung*.

Weitere Hinweise

Wird ein Datenbankfeld als Ergebnisfeld (*operand2*) verwendet, so ändert sich der Wert des Feldes durch die MOVE-Operation nur programmintern. Der in der Datenbank gespeicherte Feldwert wird davon nicht beeinflusst.

Eine Natural-Systemfunktion darf nur eingesetzt werden, wenn das MOVE-Statement in Verbindung mit einem AT BREAK-, AT END OF DATA- oder AT END OF PAGE-Statement verwendet wird.

Siehe auch Abschnitt *Arithmetische Operationen* im *Leitfaden zur Programmierung*.



Anmerkung: Wenn *operand1* eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übertragen, aber nicht die Datumskomponente (außer bei MOVE EDITED). Siehe *Syntax 4* und *Syntax 5*.

Syntax-Beschreibung

Das MOVE-Statement bietet mehrere Syntax-Varianten:

- [Syntax 1 — MOVE ROUNDED](#)
- [Syntax 2 - MOVE SUBSTRING](#)
- [Syntax 3 - MOVE BY NAME / POSITION](#)
- [Syntax 4 – MOVE EDITED \(Editiermaske mit operand2 angegeben\)](#)
- [Syntax 5 – MOVE EDITED \(Editiermaske mit operand1 angegeben\)](#)
- [Syntax 6 - MOVE LEFT / RIGHT JUSTIFIED](#)
- [Syntax 7 - MOVE NORMALIZED](#)
- [Syntax 8 - MOVE ENCODED](#)

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Syntax 1 — MOVE ROUNDED

```
MOVE [ROUNDED] operand1 [(parameter)] TO operand2 ...
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate											Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>	C	S	A		N	A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
<i>operand2</i>		S	A		M	A	U	N	P	I	F	B	D	T	L	C	G	O	ja	ja

Syntax-Element-Beschreibung:

MOVE ROUNDED	<p>Rundung:</p> <p>Das Schlüsselwort ROUNDED bewirkt, dass der Wert von <i>operand2</i> auf- bzw. abgerundet wird.</p> <p>ROUNDED wird ignoriert, wenn <i>operand2</i> nicht numerisch ist.</p> <p>Wenn <i>operand2</i> das Format N oder P hat und mehr als einmal angegeben wird, wird ROUNDED bei Zieloperanden mit 7 Stellen hinter dem Dezimalpunkt (Komma) ignoriert.</p> <p>Siehe auch Beispiel 1 – Verschiedene Beispiele für die Benutzung des MOVE-Statements.</p>	
<i>(parameter)</i>	<p>Parameter:</p> <p>Als <i>parameter</i> können Sie die Option PM=I oder den Session-Parameter DF angeben:</p>	<p>Schreibrichtung:</p> <p>Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft, können Sie die Option PM=I angeben, um den Wert von <i>operand1</i> invers (d.h. von rechts nach links) in <i>operand2</i> zu übertragen.</p> <p>Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld #B den Inhalt ZYX:</p> <pre>MOVE 'XYZ' TO #A MOVE #A (PM=I) TO #B</pre> <p>PM=I kann nur angegeben werden, wenn operand2 alphanumerisches Format hat.</p> <p>Nachfolgende Leerzeichen in <i>operand1</i> werden entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), dann wird der Wert umgedreht und anschließend in <i>operand2</i></p>

- alphanumerisch (A) oder numerisch (N) ist, gelten die mit *operand3* oder *operand4* angegebenen Werte als Byte-Zahlen.
- Unicode (U) ist, gelten die mit *operand3* oder *operand4* angegebenen Werte als Zahl der Unicode-Codeeinheiten, d.h. als Doppelbytes.

Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A in ein Feld #B zu übertragen, würden Sie folgendes angeben:

```
MOVE SUBSTRING(#A,5,8) TO #B
```

Ist *operand1* eine dynamische Variable, muss der angegebene und zu übertragene Feldteil im Bereich seiner aktuellen Länge sein; sonst tritt ein Laufzeit-Fehler auf.

Sie können einen Wert eines alphanumerischen, binären oder numerischen Feldes auch in einen bestimmten Teil des Zielfeldes übertragen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand2*) zunächst die erste Stelle (*operand5*) und dann die Länge (*operand6*) des Feldteils an, in den der Wert übertragen werden soll.

Wenn das zugrundeliegende Feldformat von *operand2*:

- alphanumerisch (A) oder numerisch (N) ist, gelten die mit *operand5* oder *operand6* angegebenen Werte als Byte-Zahlen.
- Unicode (U) ist, gelten die mit *operand5* oder *operand6* angegebenen Werte als Zahl der Unicode-Codeeinheiten, d.h. als Doppelbytes.

Um zum Beispiel den Wert eines Feldes #A in die 3. bis einschließlich 6. Stelle eines Feldes #B zu übertragen, würden Sie folgendes angeben:

```
MOVE #A TO SUBSTRING(#B,3,4)
```

Wenn *operand2* eine dynamische Variable ist, darf die erste Stelle (*operand5*) nicht größer sein als die aktuelle Länge der Variable plus 1; eine höhere erste Stelle würde einen Laufzeit-Fehler zur Folge haben, weil dies zu einer nicht definierten Lücke im Inhalt von *operand2* führen würde.

Wenn *operand3* bzw. *operand5* oder *operand4* bzw. *operand6* eine binäre Variable ist, kann sie nur mit einer Länge kleiner gleich 4 benutzt werden

Wenn Sie *operand3* bzw. *operand5* weglassen, wird ab Anfang des Feldes übertragen. Wenn Sie *operand4* bzw. *operand6* weglassen, wird ab der angegebenen Stelle (*operand3* bzw. *operand5*) bis zum Ende des Feldes übertragen.

Wenn *operand2* eine dynamische Variable ist und die erste Stelle (*operand5*) der aktuellen Länge der Variable plus 1 entspricht, was bedeutet, dass die MOVE-Operation verwendet wird, um die Länge der Variablen zu vergrößern, muss *operand6* angegeben werden, um die neue Länge der Variablen zu bestimmen.

Anmerkung: MOVE mit SUBSTRING-Option ist eine Byte-für-Byte-Übertragung (d.h. die unter *Arithmetische Operationen im Leitfaden zur Programmierung* beschriebenen Regeln gelten hierbei nicht).

Syntax 3 - MOVE BY NAME / POSITION

```
MOVE BY { [NAME]
          POSITION } operand1 TO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>		G	ja	nein
<i>operand2</i>		G	ja	nein

Syntax-Element-Beschreibung:

MOVE BY NAME <i>operand1 TO</i> <i>operand2</i>	<p>Positionsunabhängige Übertragung:</p> <p>Mit dieser Option können Sie einzelne in einer Datenstruktur enthaltene Felder in eine andere Datenstruktur übertragen, und zwar unabhängig von ihrer Position innerhalb der Struktur.</p> <p>Ein Feld kann nur übertragen werden, wenn sein Name in beiden Datenstrukturen vorkommt (dies gilt auch für mit einem REDEFINE-Statement redefinierte Felder sowie Felder, die aus einer Redefinition resultieren). Die einzelnen Felder können jedes beliebige Format haben. Die beiden Operanden können auch Views sein.</p> <p>Anmerkung: Die Reihenfolge der einzelnen Übertragungen ergibt sich aus der Reihenfolge der Felder in <i>operand1</i></p> <p>Siehe auch Beispiel 2 – Statement MOVE BY NAME.</p> <p>MOVE BY NAME mit Arrays:</p> <p>Enthalten die Datenstrukturen Arrays, so werden diese bei der Übertragung intern mit Index (*) versehen; dies kann zu einem Fehler führen, falls die Arrays nicht den Zuweisungsbedingungen für Arrays (siehe Abschnitt <i>Verarbeitung von Arrays im Leitfaden zur Programmierung</i>) entsprechen.</p> <p>Siehe auch Beispiel 3 – MOVE BY NAME mit Arrays.</p>
MOVE BY POSITION	<p>Positionsabhängige Übertragung:</p> <p>Mit dieser Option können Sie Werte von Feldern einer Gruppe in Felder einer anderen Gruppe übertragen, und zwar unabhängig von den Namen der Felder.</p>

<p><i>operand1</i> TO <i>operand2</i></p>	<p>Die Werte werden Feld für Feld von einer Gruppe in die andere übertragen, und zwar in der Reihenfolge, in der die Felder definiert sind (Felder, die aus einer Redefinition resultieren, werden dabei nicht berücksichtigt).</p> <p>Die einzelnen Felder können jedes beliebige Format haben. Die Anzahl der Felder in beiden Gruppen muss gleich sein; auch die Level-Struktur und die Array-Dimensionen der Felder müssen einander entsprechen. Format- Umsetzungen erfolgen entsprechend der im Natural Statements-Handbuch beschriebenen Regeln für arithmetische Operationen. Die beiden Operanden können auch Views sein.</p> <p>Siehe auch Beispiel 4 – MOVE BY POSITION.</p>
---	---

Syntax 4 – MOVE EDITED (Editiermaske mit operand2 angegeben)

```
MOVE EDITED operand1 TO operand2 { (EM=value) }
                                { (EMU=value) }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition								
<i>operand1</i>	C	S	A			A	U					B												ja	nein
<i>operand2</i>		S	A			A	U	N	P	I	F	B	D	T	L									ja	ja

Syntax-Element-Beschreibung:

<p>MOVE EDITED</p>	<p>Übertragung mit Editiermaske:</p> <p>Ist für <i>operand2</i> eine Editiermaske definiert, so wird der Wert von <i>operand1</i> unter Verwendung dieser Editiermaske in das Feld <i>operand2</i> übertragen.</p> <p>Die Editiermaske kann als eine Eingabe-Editiermaske für <i>operand2</i> angesehen werden, die dazu dient anzugeben, an welchen Stellen in dem alphanumerischen Inhalt von <i>operand1</i> der signifikante Input für <i>operand2</i> zu finden ist.</p> <p>Wenn die Editiermaske mehr Zeichen oder Ziffern enthält, als in <i>operand2</i> vorhanden sind, erfolgt eine entsprechende Abschneidung. Die Länge von <i>operand1</i> darf nicht kleiner sein als die von der Editiermaske dargestellte Länge des Eingabewertes. Wenn die Länge von <i>operand1</i> die Länge der Editiermaske übersteigt, werden alle darüber hinaus gehenden Daten ignoriert.</p>
---------------------------	--

	<p>Unter der Voraussetzung, dass die Länge von <i>operand1</i> nicht die Länge der Editiermaske übersteigt, kann man die Operation</p> <pre>MOVE EDITED <i>operand1</i> TO <i>operand2</i> (EM=<i>value</i>)</pre> <p>als Ausführung der folgenden Operation ansehen:</p> <pre>STACK TOP DATA <i>operand1</i> INPUT <i>operand2</i> (EM=<i>value</i>)</pre> <p>Siehe auch Beispiel 1 – Verschiedene Beispiele für die Benutzung des MOVE-Statements.</p>
EM	<p>Parameter für Editiermaske:</p> <p>Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der <i>Parameter-Referenz</i>.</p>
EMU	<p>Parameter für Editiermaske:</p> <p>Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EMU in der <i>Parameter-Referenz</i>.</p>

Syntax 5 – MOVE EDITED (Editiermaske mit operand1 angegeben)

```
MOVE EDITED operand1 { (EM=value)
                       (EMU=value) } TO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate										Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	C	S	A	N	A	U	N	P	I	F	B	D	T	L			ja	nein
<i>operand2</i>		S	A		A	U					B						ja	ja

Syntax-Element-Beschreibung:

MOVE EDITED	<p>Übertragung mit Editiermaske:</p> <p>Ist für <i>operand1</i> eine Editiermaske definiert, wird diese Editiermaske auf <i>operand1</i> angewandt und der Wert anschließend in <i>operand2</i> übertragen.</p> <p>Die Editiermaske kann als eine Ausgabe-Editiermaske für <i>operand1</i> angesehen werden, die dazu dient, eine alphanumerische Zeichenkette mit dem/der durch die Editiermaske festgelegten Layout/Länge zu erstellen. Außer den aus <i>operand1</i> stammenden Datenzeichen oder -ziffern können Sie zusätzliche ausschmückende Zeichen in die Ausgabe-Zeichenkette aufnehmen.</p>
--------------------	---

	<p>Wenn die Editiermaske mehr Zeichen referenziert, als in <i>operand1</i> vorhanden sind, erfolgt eine entsprechende Abschneidung. Die Länge der erstellten Ausgabe-Zeichenkette (die sich nach Anwendung der Editiermaske aus dem Wert von <i>operand1</i> ergibt) darf nicht die Länge von <i>operand2</i> übersteigen.</p> <p>Unter der Voraussetzung, dass die Länge von <i>operand2</i> nicht kleiner ist als die Länge der Editiermaske, kann man die Operation</p> <pre>MOVE EDITED <i>operand1</i> (EM=<i>value</i>) TO <i>operand2</i></pre> <p>als eine Operation</p> <pre>WRITE <i>operand1</i> (EM=<i>value</i>)</pre> <p>betrachten, bei der die Ausgabe nicht auf den Schirm erfolgt, sondern in die Variable <i>operand2</i> geschrieben wird.</p> <p>Siehe auch Beispiel 1 – Verschiedene Beispiele für die Benutzung des MOVE-Statements.</p>
EM	<p>Parameter für Editiermaske:</p> <p>Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der <i>Parameter-Referenz</i>.</p>
EMU	<p>Parameter für Editiermaske:</p> <p>Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EMU in der <i>Parameter-Referenz</i>.</p>

Syntax 6 - MOVE LEFT / RIGHT JUSTIFIED

(Wertübertragung mit links- oder rechtsbündiger Ausrichtung)

```
MOVE { LEFT
      RIGHT } [JUSTIFIED] operand1 [(parameter)] TO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C	S	A	N A U N P I F B D T L	ja	nein
<i>operand2</i>		S	A	A U	ja	ja

Syntax-Element-Beschreibung:

MOVE LEFT / RIGHT JUSTIFIED	<p>Ausrichtung:</p> <p>Mit dieser Option können Sie festlegen, ob der übertragene Wert in <i>operand2</i> links- oder rechtsbündig ausgerichtet werden soll.</p> <p>MOVE LEFT/RIGHT JUSTIFIED kann nicht benutzt werden, wenn <i>operand2</i> eine dynamische Variable ist.</p>
MOVE LEFT JUSTIFIED	<p>Linksbündig:</p> <p>Bei MOVE LEFT JUSTIFIED werden vorangestellte Leerzeichen in <i>operand1</i> entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), bevor der Wert linksbündig in <i>operand2</i> übertragen wird. Der Rest von <i>operand2</i> wird dann gegebenenfalls mit Leerzeichen aufgefüllt. Falls der Wert länger als <i>operand2</i> ist, wird der Wert rechts abgeschnitten.</p>
MOVE RIGHT JUSTIFIED	<p>Rechtsbündig:</p> <p>Bei MOVE RIGHT JUSTIFIED werden nachfolgende Leerzeichen in <i>operand1</i> entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), bevor der Wert rechtsbündig in <i>operand2</i> übertragen wird. Der Rest von <i>operand2</i> wird dann gegebenenfalls mit Leerzeichen aufgefüllt. Falls der Wert länger als <i>operand2</i> ist, wird der Wert links abgeschnitten.</p> <p>Siehe auch Beispiel 1 – Verschiedene Beispiele für die Benutzung des MOVE-Statements.</p>
<i>parameter</i>	<p>Invertierte Anzeigerichtung:</p> <p>Wenn Sie MOVE LEFT/RIGHT JUSTIFIED in Verbindung mit PM=I verwenden, erfolgt die Übertragung in folgenden Schritten:</p> <ol style="list-style-type: none"> 1. Falls <i>operand1</i> nicht alphanumerisches Format hat, wird der Wert in alphanumerisches Format umgesetzt. 2. Nachfolgende Leerzeichen in <i>operand1</i> werden entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt). 3. Bei LEFT JUSTIFIED werden außerdem vorangestellte Leerzeichen in <i>operand1</i> entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt). 4. Der Wert wird umgedreht und dann in <i>operand2</i> übertragen. 5. Gegebenenfalls wird <i>operand2</i> mit Leerzeichen aufgefüllt oder der Wert abgeschnitten (vgl. oben).

Syntax 7 - MOVE NORMALIZED

Das Statement `MOVE NORMALIZED` konvertiert eine Unicode-Zeichenkette in die Unicode Normalization Form C (NFC). Die sich daraus ergebende Unicode-Zeichenkette enthält keine Kombinationssequenzen für Zeichen mehr, die als vordefinierte Zeichen zur Verfügung stehen.

Wenn das Format des Zieloperanden selbst kein Unicode ist, findet eine implizite Konvertierung von Unicode in das Codepage-Format des Zieloperanden statt – während dieser Konvertierung wird die Standard-Codepage (siehe Systemvariable `*CODEPAGE`) benutzt.

Weitere Informationen zum Statement `MOVE NORMALIZED` siehe Abschnitt *Statements* in der *Unicode and Code Page Support*-Dokumentation.

Syntax-Diagramm:

```
MOVE NORMALIZED operand1 TO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A	U	ja	nein
<i>operand2</i>	S A	A U	ja	ja

Syntax-Element-Beschreibung:

MOVE NORMALIZED	<p>Konvertierung von Unicode-Feldern:</p> <p>Diese Option dient zum Konvertieren von Unicode-Feldern mit potenziell unnormalisiertem Inhalt in die Unicode Normalization Form C (NFC).</p> <p>Diese zusammengesetzte Form einer Unicode-Zeichenkette enthält keine Kombinationssequenzen für Zeichen, die als vordefinierte Zeichen zur Verfügung stehen.</p> <p>Siehe auch http://www.unicode.org/reports/tr15/#Canonical_Composition_Examples (<i>Normalization Forms D and C Examples</i>).</p>
------------------------	---

	Beispiel: <pre>MOVE NORMALIZED #SCR TO #TGT</pre>
<i>operand1</i>	Zu konvertierende Unicode-Zeichenkette: <i>operand1</i> enthält die zu konvertierende Unicode-Zeichenkette.
<i>operand2</i>	Konvertierte Zeichenkette: <i>operand2</i> (Zieloperand) dient zur Aufnahme der konvertierten Zeichenkette.

Beispiel:

Einige Codepunkte haben unterschiedliche Darstellungen im Unicode. Zum Beispiel der deutsche Buchstabe Ä: die aufgelöste Darstellung im Unicode ist U+0041, gefolgt von U+0308, bei der ein Kombinationszeichen (U+0308) benutzt wird; eine andere Darstellung ist das vordefinierte Zeichen U+00C4. Das Statement `MOVE NORMALIZED` konvertiert die Unicode-Darstellung mit Kombinationszeichen in eine standardisierte Unicode-Darstellung mittels vordefinierter Zeichen, wo möglich.

Syntax 8 - MOVE ENCODED

Dieser Abschnitt erläutert die Syntax des Statements `MOVE ENCODED`.

Weitere Informationen zum Statement `MOVE ENCODED` siehe Abschnitt *Statements* in der *Unicode and Code Page Support*-Dokumentation.

Syntax-Diagramm:

```
MOVE ENCODED
  operand1 [[IN] CODEPAGE operand2] TO
  operand3 [[IN] CODEPAGE operand4]
  [GIVING operand5]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A	A U B	ja	nein
<i>operand2</i>	S	A U	ja	nein
<i>operand3</i>	S	A U B	ja	ja
<i>operand4</i>	S A	A U	ja	nein
<i>operand5</i>	S	I4	ja	ja

Syntax-Element-Beschreibung:

MOVE ENCODED	<p>Codepage-Umsetzung:</p> <p>Das Statement <code>MOVE ENCODED</code> setzt eine in einer Codepage kodierte Zeichenkette in die äquivalente Zeichenkette einer anderen Codepage um.</p> <p>Anmerkung: Natural verwendet für die Unicode-Konvertierung die International Components for Unicode (ICU) Library. Weitere Informationen siehe http://icu.sourceforge.net/userguide/.</p>
<i>operand1</i>	<p>Umzusetzende Zeichenkette:</p> <p><i>operand1</i> enthält die umzusetzende Zeichenkette.</p>
[IN] CODEPAGE <i>operand2</i>	<p>Codepage der umzusetzenden Zeichenkette:</p> <p>Die Codepage von <i>operand1</i> kann nur angegeben werden, wenn <i>operand1</i> das Format A oder B hat. Siehe Anmerkung 1.</p>
TO <i>operand3</i>	<p>Umgesetzte Zeichenkette:</p> <p>Wenn das Ergebnis der Umsetzung nicht in das Zielfeld (<i>operand3</i>) passt, wird das Ergebnis aufgefüllt bzw. abgeschnitten, und als Füllzeichen wird das Leerzeichen der sich daraus ergebenden Codepage benutzt.</p> <p>Wenn das Zielfeld als eine dynamische Variable definiert wird, ist kein Auffüllen oder Abschneiden erforderlich, da die Länge der dynamischen Variablen automatisch an die Länge des Ergebnisses der Konvertierung angepasst wird.</p>
[IN] CODEPAGE <i>operand4</i>	<p>Codepage der umgesetzten Zeichenkette:</p> <p>Die Codepage von <i>operand3</i> kann nur angegeben werden, wenn <i>operand3</i> das Format A oder B hat. Siehe Anmerkung 1.</p>
GIVING <i>operand5</i>	<p>GIVING-Klausel:</p> <p>Ohne das Schlüsselwort <code>GIVING</code> wird im Falle eines Fehlers eine Natural- Fehlermeldung zurückgegeben. Wenn das Schlüsselwort <code>GIVING</code> benutzt wird, gibt <i>operand5</i> eine 0 oder den Natural-Fehlercode anstatt der Natural-Fehlermeldung zurück.</p> <p>Wenn der Zieloperand abgeschnitten wird, wird keine Natural-Fehlermeldung ausgegeben, aber wenn das Schlüsselwort <code>GIVING</code> benutzt wird, enthält <i>operand5</i> einen entsprechenden Fehlerkode, um auf ein Abschneiden hinzuweisen.</p>

**Anmerkungen:**

1. Wenn kein Codepage-Operand angegeben wird, dann wird die voreingestellte (Default-)Codepage (Wert der Systemvariablen `*CODEPAGE`) benutzt.
2. Wenn der Session-Parameter `CPCVERR` in dem Statement `SET GLOBALS` oder in dem Systemkommando `GLOBALS` auf `ON` gesetzt ist, wird ein Fehler ausgegeben, wenn mindestens ein Zeichen des Ausgangsfeldes nicht ordnungsgemäß in die Ziel-Codepage umgesetzt werden konnte, sondern im Zielfeld durch ein Ersetzungszeichen ersetzt wurde.

Beispiele:

```
MOVE ENCODED A-FIELD1 TO A-FIELD2
```

Ungültig: Dies führt zu einem Syntaxfehler, da die Codepage-Namen standardmäßig übernommen werden und für *operand1* und *operand3* identisch sind.

```
MOVE ENCODED A-FIELD1 CODEPAGE 'IBM01140' TO A-FIELD2 CODEPAGE 'IBM01140'
```

Ungültig: Dies führt zu einem Fehler, da die kodierten Codepage-Namen für *operand1* und *operand3* identisch sind.

```
MOVE ENCODED A-FIELD1 CODEPAGE 'IBM01140' TO A-FIELD2 CODEPAGE 'IBM037'
```

Gültig: Die Zeichenkette in A-FIELD1 (kodiert in IBM01140) wird in A-FIELD2 (kodiert in IBM037) konvertiert.

```
MOVE ENCODED U-FIELD TO U-FIELD
```

Ungültig: Dies führt zu einem Fehler, da mindestens ein Operand vom Format A oder B sein muss.

```
MOVE ENCODED U-FIELD TO A-FIELD
```

Gültig: Die Unicode-Zeichenkette in U-FIELD wird angesichts der Tatsache, dass sie in UTF-16 kodiert ist, in das alphanumerische A-FIELD in der Standard-Codepage (*CODEPAGE) konvertiert.

```
MOVE ENCODED A-FIELD TO U-FIELD
```

Gültig: Die Zeichenkette in A-FIELD wird angesichts der Tatsache, dass sie in der Standard-Codepage (*CODEPAGE) kodiert ist, in das Unicode-Feld U-FIELD konvertiert.

```
MOVE ENCODED A100-FIELD CODEPAGE 'IBM1140' TO A50-FIELD CODEPAGE 'IBM037'
```

Gültig: Die Konvertierung erfolgt mittels der betreffenden Codepages von A100-FIELD (Format/Länge: A100) in A50-FIELD (Format/Länge: A50). Der Zieloperand wird abgeschnitten. Keine Natural-Fehlermeldung wird zurückgegeben.

```
MOVE ENCODED A100-FIELD CODEPAGE 'IBM1140' TO A50-FIELD CODEPAGE 'IBM037' GIVING
RC-FIELD
```

Gültig: Die Konvertierung erfolgt mittels der betreffenden Codepages von A100-FIELD (Format/Länge: A100) in A50-FIELD (Format/Länge: A50). Das Ziel wird abgeschnitten. Da eine GIVING-Klausel angegeben wird, erhält das RC-FIELD einen Fehlercode, der darauf hinweist, dass in diesem Fall Werte abgeschnitten wurden.

Beispiele

- [Beispiel 1 — Verschiedene Beispiele für die Benutzung des MOVE-Statements](#)
- [Beispiel 2 — MOVE BY NAME](#)
- [Beispiel 3 — MOVE BY NAME mit Arrays](#)
- [Beispiel 4 — MOVE BY POSITION](#)

Beispiel 1 — Verschiedene Beispiele für die Benutzung des MOVE-Statements

```
** Example 'MOVEX1': MOVE
*****
DEFINE DATA LOCAL
1 #A (N3)
1 #B (A5)
1 #C (A2)
1 #D (A7)
1 #E (N1.0)
1 #F (A5)
1 #G (N3.2)
1 #H (A6)
END-DEFINE
*
MOVE 5 TO #A
WRITE NOTITLE 'MOVE 5 TO #A'          30X '=' #A
*
MOVE 'ABCDE' TO #B #C #D
WRITE 'MOVE ABCDE TO #B #C #D'      20X '=' #B '=' #C '=' #D
*
MOVE -1 TO #E
WRITE 'MOVE -1 TO #E'                28X '=' #E
*
MOVE ROUNDED 1.995 TO #E
WRITE 'MOVE ROUNDED 1.995 TO #E'    18X '=' #E
*
*
MOVE RIGHT JUSTIFIED 'ABC' TO #F
WRITE 'MOVE RIGHT JUSTIFIED 'ABC'' TO #F'    10X '=' #F
*
MOVE EDITED '003.45' TO #G (EM=999.99)
```

```

WRITE 'MOVE EDITED ''003.45'' TO #G (EM=999.99)' 4X '=' #G
*
MOVE EDITED 123.45 (EM=999.99) TO #H
WRITE 'MOVE EDITED 123.45 (EM=999.99) TO #H'      6X '=' #H
*
END

```

Ausgabe des Programms MOVEX1:

```

MOVE 5 TO #A           #A: 5
MOVE ABCDE TO #B #C #D #B: ABCDE #C: AB #D: ABCDE
MOVE -1 TO #E         #E: -1
MOVE ROUNDED 1.995 TO #E #E: 2
MOVE RIGHT JUSTIFIED 'ABC' TO #F #F: ABC
MOVE EDITED '003.45' TO #G (EM=999.99) #G: 3.45
MOVE EDITED 123.45 (EM=999.99) TO #H #H: 123.45

```

Beispiel 2 — MOVE BY NAME

```

** Example 'MOVEX2': MOVE BY NAME
*****
DEFINE DATA LOCAL
1 #SBLOCK
  2 #FIELD A (A10) INIT <'AAAAAAAAAA'>
  2 #FIELD B (A10) INIT <'BBBBBBBBBB'>
  2 #FIELD C (A10) INIT <'CCCCCCCCCC'>
  2 #FIELD D (A10) INIT <'DDDDDDDDDD'>
1 #TBLOCK
  2 #FIELD1 (A15) INIT <' '>
  2 #FIELD A (A10) INIT <' '>
  2 #FIELD2 (A10) INIT <' '>
  2 #FIELD B (A10) INIT <' '>
  2 #FIELD3 (A20) INIT <' '>
  2 #FIELD C (A10) INIT <' '>
END-DEFINE
*
MOVE BY NAME #SBLOCK TO #TBLOCK
*
WRITE NOTITLE 'CONTENTS OF #TBLOCK AFTER MOVE BY NAME:'
  // '=' #TBLOCK.#FIELD1
  / '=' #TBLOCK.#FIELD A
  / '=' #TBLOCK.#FIELD2
  / '=' #TBLOCK.#FIELD B
  / '=' #TBLOCK.#FIELD3
  / '=' #TBLOCK.#FIELD C
*
END

```

Inhalt von #TBLOCK nach der MOVE BY NAME-Verarbeitung:

```
CONTENTS OF #TBLOCK AFTER MOVE BY NAME:
```

```
#FIELD1:
#FIELD A: AAAAAAAAAA
#FIELD2:
#FIELD B: BBBB BBBBBB
#FIELD3:
#FIELD C: CCCCCCCCCC
```

Beispiel 3 — MOVE BY NAME mit Arrays

```
DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD (A10/1:10)
  1 #GROUP2
    2 #FIELD (A10/1:10)
END-DEFINE
...
MOVE BY NAME #GROUP1 TO #GROUP2
...
```

In diesem Beispiel würde das MOVE-Statement intern wie folgt aufgelöst:

```
MOVE #GROUP1.#FIELD (*) TO #GROUP2.#FIELD (*)
```

Wenn ein Teil einer indizierten Gruppe in einen anderen Teil derselben Gruppe übertragen wird, kann dies zu unerwarteten Ergebnissen führen, wie im folgenden Beispiel veranschaulicht.

```
DEFINE DATA LOCAL
  1 #GROUP1 (1:5)
    2 #FIELD A (N1) INIT <1,2,3,4,5>
    2 REDEFINE #FIELD A
      3 #FIELD B (N1)
END-DEFINE
...
MOVE BY NAME #GROUP1 (2:4) TO #GROUP1 (1:3)
...
```

In diesem Beispiel würde das MOVE-Statement intern wie folgt aufgelöst:

```
MOVE #FIELD A (2:4) TO #FIELD A (1:3)
MOVE #FIELD B (2:4) TO #FIELD B (1:3)
```

Zunächst wird der Inhalt der Ausprägungen 2 bis 4 von #FIELD A in die Ausprägungen 1 bis 3 von #FIELD A übertragen; d.h. die Ausprägungen erhalten folgende Werte:

Ausprägung:	1.	2.	3.	4.	5.
Wert vorher:	1	2	3	4	5
Wert nachher:	2	3	4	4	5

Dann wird der Inhalt der Ausprägungen 2 bis 4 von #FIELD B in die Ausprägungen 1 bis 3 von #FIELD B übertragen; d.h. die Ausprägungen erhalten folgende Werte:

Ausprägung:	1.	2.	3.	4.	5.
Wert vorher:	2	3	4	4	5
Wert nachher:	3	4	4	4	5

Beispiel 4 — MOVE BY POSITION

```
DEFINE DATA LOCAL
  1 #GROUP1
    2 #FIELD1A (N5)
    2 #FIELD1B (A3/1:3)
    2 REDEFINE #FIELD1B
      3 #FIELD1BR (A9)
  1 #GROUP2
    2 #FIELD2A (N5)
    2 #FIELD2B (A3/1:3)
    2 REDEFINE #FIELD2B
      3 #FIELD2BR (A9)
END-DEFINE
...
MOVE BY POSITION #GROUP1 TO #GROUP2
...
```

In diesem Beispiel wird der Inhalt von #FIELD1A in #FIELD2A übertragen, und der Inhalt von #FIELD1B in #FIELD2B; die Felder #FIELD1BR und #FIELD2BR sind davon nicht betroffen.

82

MOVE ALL

▪ Funktion	540
▪ Syntax-Beschreibung	540
▪ Beispiel	541

```
MOVE ALL operand1 TO operand2 [UNTIL operand3]
```

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: [Arithmetische Funktionen und Datenzuweisungen](#)

Funktion

Das Statement `MOVE ALL` dient dazu, den Wert von *operand1* so oft nach *operand2* zu übertragen, bis *operand2* voll ist.

Bei Verwendung der UNTIL-Option kann in *operand3* angegeben werden, wieviele Stellen in *operand2* gefüllt werden sollen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A U N B	ja	nein
<i>operand2</i>	S A	A U B	ja	ja
<i>operand3</i>	C S	N P I	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Ausgangsoperand:</p> <p>Dieser Operand enthält den zu übertragenden Wert.</p> <p>Bei einem numerischen Operanden werden alle Stellen einschließlich vorangestellter Nullen übertragen.</p>
TO <i>operand2</i>	<p>Zieloperand:</p> <p>In diesen Operanden wird der Wert übertragen. Der vorherige Inhalt des Operanden wird vor der <code>MOVE ALL</code>-Operation nicht zurückgesetzt. Dies ist insbesondere bei Verwendung der UNTIL-Option zu beachten, da bereits in <i>operand2</i> enthaltene Daten dort behalten</p>

	werden, es sei denn, sie werden während der MOVE ALL-Operation ausdrücklich überschrieben.
UNTIL <i>operand3</i>	<p>UNTIL-Option:</p> <p>Mit der UNTIL-Option können Sie die MOVE ALL-Operation auf eine bestimmte Anzahl von Stellen in <i>operand2</i> begrenzen. Mit <i>operand3</i> geben Sie an, wieviele Stellen in <i>operand2</i> gefüllt werden sollen; ist diese Anzahl erreicht, wird die MOVE ALL-Operation beendet.</p> <p>Übersteigt die Länge von <i>operand3</i> die Länge von <i>operand2</i>, wird die MOVE ALL-Operation beendet, sobald <i>operand2</i> vollständig gefüllt ist.</p> <p>Die Option UNTIL kann auch verwendet werden, um einer dynamischen Variable einen Startwert zuzuweisen. Wenn <i>operand2</i> eine dynamische Variable ist, entspricht ihre Länge nach der MOVE ALL-Operation dem Wert von <i>operand3</i>. Die aktuelle Länge einer dynamischen Variable kann unter Verwendung der Systemvariable *LENGTH festgestellt werden. Allgemeine Informationen zu dynamischen Variablen finden Sie im Abschnitt <i>Benutzung dynamischer Variablen..</i></p>

Beispiel

```

** Example 'MOAEX1': MOVE ALL
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 4
RD. READ EMPLOY-VIEW BY NAME
  SUSPEND IDENTICAL SUPPRESS
/*
FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
  IF NO RECORDS FOUND
    MOVE ALL '*' TO FIRST-NAME (RD.)
    MOVE ALL '*' TO CITY (RD.)
    MOVE ALL '*' TO MAKE (FD.)
  END-NOREC
/*
DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
  NAME (RD.) FIRST-NAME (RD.)
  CITY (RD.)

```

MOVE ALL

```
                MAKE (FD.) (IS=OFF)
/*
END-FIND
END-READ
END
```


Ausgabe des Programms MOAEX1:

NAME	FIRST-NAME	CITY	MAKE
ABELLAN	*****	*****	*****
ACHIESON	ROBERT	DERBY	FORD
ADAM	*****	*****	*****
ADKINSON	JEFF	BROOKLYN	GENERAL MOTORS

83

MOVE INDEXED

Das `MOVE INDEXED`-Statement wird nur noch aus Kompatibilitätsgründen unterstützt.

 **Vorsicht:** Im Gegensatz zu einem `MOVE`-Statement mit Array-Operanden sind Prüfungen auf außerhalb der Grenzen liegenden Indexwerte nicht möglich, wenn ein `MOVE INDEXED`-Statement ausgeführt wird. Als Folge davon können Sie bei der Ausführung eines nicht korrekten `MOVE INDEXED`-Statements ohne Absicht Benutzerdaten zerstören.

Deshalb empfiehlt die Software AG, vorhandene `MOVE INDEXED`-Statement durch `MOVE`-Statements zu ersetzen.

Siehe Statement [MOVE](#).

84 MULTIPLY

▪ Funktion	546
▪ Syntax-Beschreibung	546
▪ Beispiel	548

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE](#) | [MOVE ALL](#) | [RESET](#) | [SEPARATE](#) | [SUBTRACT](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das Statement `MULTIPLY` dient dazu, zwei Operanden miteinander zu multiplizieren. Das Ergebnis der Multiplikation kann entweder in *operand1* oder in *operand3* ausgegeben werden.

Wird ein Datenbankfeld als Ergebnisfeld verwendet, so ändert sich durch die Multiplikation nur der programmintern benutzte Wert des Feldes. Der in der Datenbank gespeicherte Feldwert wird davon nicht beeinflusst.

Zu Multiplikationen mit Arrays siehe auch den Abschnitt *Arithmetische Operationen mit Arrays* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung

Bei diesem Statement sind unterschiedliche Strukturen möglich.

- [Syntax 1](#) — `MULTIPLY` ohne `GIVING`-Klausel
- [Syntax 2](#) — `MULTIPLY` mit `GIVING`-Klausel

Syntax 1 — `MULTIPLY` ohne `GIVING`-Klausel

Wenn Syntax 1 benutzt wird, kann das Ergebnis der Multiplikation in *operand1* gespeichert werden.

```
MULTIPLY [ROUNDED] operand1 BY operand2
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Operanden-Definitionstabelle (Syntax 1):

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>		S	A		M	N	P	I	F							ja	nein
<i>operand2</i>	C	S	A		N	N	P	I	F							ja	nein

Syntax-Element-Beschreibung (Syntax 1):

<i>operand1</i> BY <i>operand2</i>	<p><i>operand1</i> ist der Multiplikand, <i>operand2</i> ist der Multiplikator. Da keine GIVING-Klausel benutzt wird, wird das Ergebnis in <i>operand1</i> gespeichert, folglich ist das Statement äquivalent zu:</p> <pre><oper1> := <oper1> * <oper2></pre>
ROUNDED	<p>Wenn Sie das Schlüsselwort ROUNDED angeben, wird der Wert auf- oder abgerundet, bevor er dann <i>operand1</i> zugewiesen wird. Informationen zum Runden, siehe <i>Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten im Leitfaden zur Programmierung</i>.</p>

Syntax 2 — MULTIPLY mit GIVING-Klausel

Wenn Syntax 2 benutzt wird, kann das Ergebnis der Multiplikation in *operand3* gespeichert werden.

```
MULTIPLY [ROUNDED] operand1 BY operand2 GIVING operand3
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Operanden-Definitionstabelle (Syntax 2):

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>	C	S	A		M		N	P	I	F						ja	nein
<i>operand2</i>	C	S	A		N		N	P	I	F						ja	nein
<i>operand3</i>		S	A		M	A	U	N	P	I	F	B*	T			ja	ja

* Format B von *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung (Syntax 2):

<i>operand1</i> BY	<i>operand1</i> ist der Multiplikand, <i>operand2</i> ist der Multiplikator.
<i>operand2</i> GIVING	Da die GIVING-Klausel benutzt wird, wird <i>operand1</i> nicht geändert, und das Ergebnis wird in <i>operand3</i> gespeichert, folglich ist das Statement äquivalent zu:
<i>operand3</i>	<div style="background-color: #e0e0e0; padding: 5px; margin: 5px 0;"><code><oper3> := <oper1> * <oper2></code></div> Ist <i>operand1</i> eine numerische Konstante, ist die GIVING-Klausel erforderlich.
ROUNDED	Wenn Sie das Schlüsselwort ROUNDED angeben, wird der Wert auf- oder abgerundet, bevor er <i>operand1</i> zugewiesen wird. Informationen zum Runden siehe Abschnitt <i>Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten im Leitfaden zur Programmierung.</i>

Beispiel

```

** Example 'MULEX1': MULTIPLY
*****
DEFINE DATA LOCAL
1 #A      (N3) INIT <20>
1 #B      (N5)
1 #C      (N3.1)
1 #D      (N2)
1 #ARRAY1 (N5/1:4,1:4) INIT (2,*) <5>
1 #ARRAY2 (N5/1:4,1:4) INIT (4,*) <10>
END-DEFINE
*
MULTIPLY #A BY 3
WRITE NOTITLE 'MULTIPLY #A BY 3'          25X '=' #A
*
MULTIPLY #A BY 3 GIVING #B
WRITE 'MULTIPLY #A BY 3 GIVING #B'      15X '=' #B
*
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C
WRITE 'MULTIPLY ROUNDED 3 BY 3.5 GIVING #C' 6X '=' #C
*
MULTIPLY 3 BY -4 GIVING #D
WRITE 'MULTIPLY 3 BY -4 GIVING #D'      14X '=' #D
*
MULTIPLY -3 BY -4 GIVING #D
WRITE 'MULTIPLY -3 BY -4 GIVING #D'     14X '=' #D
*
MULTIPLY 3 BY 0 GIVING #D
WRITE 'MULTIPLY 3 BY 0 GIVING #D'       14X '=' #D

```



```

*
WRITE / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)
WRITE / 'MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)'
      / '=' #ARRAY1 (2,*) '=' #ARRAY2 (4,*)
*
END

```

Ausgabe des Programms MULEX1:

```

MULTIPLY #A BY 3 #A: 60
MULTIPLY #A BY 3 GIVING #B #B: 180
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C #C: 10.5
MULTIPLY 3 BY -4 GIVING #D #D: -12
MULTIPLY -3 BY -4 GIVING #D #D: 12
MULTIPLY 3 BY 0 GIVING #D #D: 0

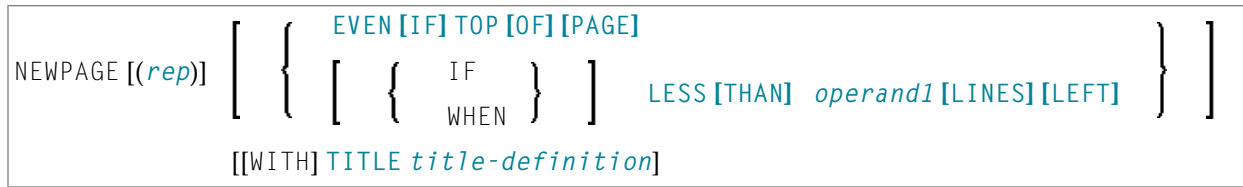
#ARRAY1: 5 5 5 5 #ARRAY2: 10 10 10 10

MULTIPLY #ARRAY1 (2,*) BY #ARRAY2 (4,*)
#ARRAY1: 50 50 50 50 #ARRAY2: 10 10 10 10

```


85 NEWPAGE

▪ Funktion	552
▪ Syntax-Beschreibung	553
▪ Beispiel	554



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [AT END OF PAGE](#) | [AT TOP OF PAGE](#) | [CLOSE PRINTER](#) | [DEFINE PRINTER](#) | [DISPLAY](#) | [EJECT](#) | [FORMAT](#) | [PRINT](#) | [SKIP](#) | [SUSPEND IDENTICAL SUPPRESS](#) | [WRITE](#) | [WRITE TITLE](#) | [WRITE TRAILER](#)

Gehört zur Funktionsgruppe: [Erstellen von Ausgabe-Reports](#)

Funktion

Das Statement `NEWPAGE` dient dazu, einen Seitenvorschub auszulösen und eine neue Seite zu beginnen. `NEWPAGE` bewirkt außerdem, dass etwaige `AT END OF PAGE`- und `WRITE TRAILER`-Statements ausgeführt werden. Ist eine Seitenüberschrift-Verarbeitung aber nicht ausdrücklich definiert (`WRITE TITLE`, `WRITE NOTITLE` oder `DISPLAY NOTITLE`), erhält jede neue Seite eine Standardüberschrift mit Datum, Uhrzeit und laufender Seitennummer.



Anmerkungen:

1. Der Seitenvorschub wird dann nicht vorgenommen, wenn das `NEWPAGE`-Statement ausgeführt wird, sondern nur, wenn ein anschließendes, eine Ausgabe erzeugendes Statement ausgeführt wird.
2. Wird kein `NEWPAGE`-Statement verwendet, so wird der Seitenvorschub automatisch in Abhängigkeit von der mit dem Profil-/Session-Parameter `PS` definierten Seitenlänge gesteuert.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	N P I	ja	nein

Syntax-Element-Beschreibung:

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement NEWPAGE beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls mit (<i>rep</i>) nichts anderes angegeben wird, bezieht sich das NEWPAGE-Statement auf den ersten Report (Report 0).</p> <p>Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
EVEN IF TOP OF PAGE	Mit dieser Option bewirken Sie, dass das NEWPAGE-Statement einen Seitenvorschub (mit entsprechender AT TOP OF PAGE - und Seitenüberschrift-Verarbeitung) auslöst, auch wenn unmittelbar vorher bereits ein Seitenvorschub erfolgt ist.
WHEN LESS THAN <i>operand1 LINES LEFT</i>	Mit dieser Option bewirken Sie, dass das NEWPAGE-Statement ausgeführt wird, wenn auf der aktuellen Seite weniger als <i>operand1</i> Zeilen übrig sind. Der interne Zeilenzähler orientiert sich hierbei am Wert des Profil-/Session-Parameters PS.
WITH TITLE <i>title-definition</i>	Diese Option können Sie verwenden, um auf der generierten neuen Seite eine Überschrift auszugeben. Für die <i>title-definition</i> gilt dieselbe Syntax wie für das Statement WRITE TITLE , außer dass die SKIP -Klausel in einem NEWPAGE WITH TITLE <i>title-definition</i> -Statement nicht erlaubt ist.

Beispiel

```

** Example 'NWPEX1': NEWPAGE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 SALARY (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 15
READ EMPLOY-VIEW BY CITY FROM 'DENVER'
  DISPLAY CITY (IS=ON) NAME SALARY (1) CURR-CODE (1)
  AT BREAK OF CITY
  SKIP 1
  /*
  NEWPAGE WHEN LESS THAN 10 LINES LEFT
  WRITE '*****'
  / 'SUMMARY FOR ' OLD(CITY)
  / '*****'
  / '*****'
  / 'SUM OF SALARIES:' SUM(SALARY(1))
  / 'AVG OF SALARIES:' AVER(SALARY(1))
  / '*****'
  NEWPAGE
  /*
  END-BREAK
END-READ
END

```

Ausgabe des Programms NWPEX1 - Seite 1:

```

Page      1                                05-01-18  10:01:45
          CITY                             NAME          ANNUAL   CURRENCY
                   SALARY                   CODE
-----
DENVER      TANIMOTO      33000   USD
            MEYER        50000   USD

*****
SUMMARY FOR DENVER
*****
*****

```

SUM OF SALARIES: 83000
 AVG OF SALARIES: 41500

Ausgabe des Programms NWPEX1 - Seite 2:

Page 2 05-01-18 10:01:45

CITY	NAME	ANNUAL SALARY	CURRENCY CODE
DERBY	DEAKIN	8750	UKL
	GARFIELD	6750	UKL
	MUNN	8800	UKL
	MUNN	5650	UKL
	GREBBY	9550	UKL
	WHITT	8650	UKL
	PONSONBY	5500	UKL
	MAGUIRE	4150	UKL
	HEYWOOD	3900	UKL
	BRYDEN	6750	UKL
	SMITH	39000	UKL
	CONQUEST	45000	UKL
ACHIESON	11300	UKL	

 SUMMARY FOR DERBY

Ausgabe des Programms NWPEX1 - Seite 3:

DERBY	DEAKIN	8750	UKL
	GARFIELD	6750	UKL
	MUNN	8800	UKL
	MUNN	5650	UKL
	GREBBY	9550	UKL
	WHITT	8650	UKL
	PONSONBY	5500	UKL
	MAGUIRE	4150	UKL
	HEYWOOD	3900	UKL
	BRYDEN	6750	UKL
	SMITH	39000	UKL
	CONQUEST	45000	UKL
ACHIESON	11300	UKL	

```
SUMMARY FOR  DERBY
*****
*****
SUM OF SALARIES:      163750
AVG OF SALARIES:      12596
*****
```


86

OBTAIN

▪ Funktion	558
▪ Einschränkung	558
▪ Syntax-Beschreibung	559
▪ Beispiele	563

OBTAIN *operand1* ...

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `OBTAIN` dient dazu, ein oder mehrere Datenbankfelder zu lesen. Das `OBTAIN`-Statement erzeugt keinen ausführbaren Code im Natural-Objektprogramm. Es wird in erster Linie benutzt, um einen Wertebereich eines multiplen Feldes oder einen Bereich von Ausprägungen einer Periodengruppe zu lesen, so dass Teile dieser Bereiche nacheinander im Programm referenziert werden können.

Ein `OBTAIN`-Statement ist für jedes im Programm zu referenzierende Datenbankfeld nicht erforderlich, da Natural ja automatisch jedes in einem nachfolgenden Statement referenzierte Datenbankfeld liest (zum Beispiel, ein `DISPLAY`- oder `COMPUTE`-Statement).

Wenn multiple Felder oder Periodengruppen in der Form eines Arrays referenziert werden, muss das Array mit einem `OBTAIN`-Statement definiert werden, um sicher zu stellen, dass es für alle Ausprägungen des Feldes erstellt ist. Wenn einzelne multiple Felder oder Periodengruppen referenziert werden, bevor das Array definiert wird, werden die Felder nicht innerhalb des Arrays positioniert und existieren unabhängig vom Array. Die Felder enthalten denselben Wert wie die entsprechende Ausprägung innerhalb des Arrays.

Einzelne Ausprägungen oder multiple Felder oder Periodengruppen oder Subarrays können innerhalb eines vorher definierten Arrays gehalten werden, wenn die Array-Dimensionen der zweiten individuellen Ausprägung oder das Array innerhalb des ersten Arrays enthalten sind.

Referenzen auf multiple Felder oder Periodengruppen mit eindeutigem variablen Index können nicht in einem Array von Werten enthalten sein. Wenn einzelne Ausprägungen eines Arrays mit einem variablen Index verarbeitet werden sollen, muss dem Index-Ausdruck der eindeutige variable Index vorausgehen, um das individuelle Array zu kennzeichnen.

Einschränkung

Das `OBTAIN`-Statement gilt nur für den Reporting Mode.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G	A U N P I F B D T L	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	Als <i>operand1</i> geben Sie die Felder an, die mit dem OBTAIN-Statement gelesen werden sollen.
-----------------	--

Beispiele:

```
READ FINANCE OBTAIN CREDIT-CARD (1-10)
DISPLAY CREDIT-CARD (3-5) CREDIT-CARD (6-8)
SKIP 1 END
```

Das obige Beispiel führt dazu, dass die ersten 10 Ausprägungen des Feldes CREDIT-CARD (das in einer Periodengruppe enthalten ist) gelesen werden, und die Ausprägungen (3-5) und (6-8) angezeigt werden, wo die nachfolgenden Subarrays im ersten Array (1-10) residieren.

```
READ FINANCE
MOVE 'ONE' TO CREDIT-CARD (1)
DISPLAY CREDIT-CARD (1) CREDIT-CARD (1-5)
```

Ausgabe:

```

CREDIT-CARD      CREDIT-CARD
-----
ONE              DINERS CLUB
                AMERICAN EXPRESS

ONE              AVIS
                AMERICAN EXPRESS
```

OBTAIN

```
ONE          HERTZ
             AMERICAN EXPRESS
```

```
ONE          UNITED AIR TRAVEL
```

Die erste Referenz auf CREDIT-CARD (1) ist nicht innerhalb des Arrays enthalten. Das Array, das nach der Referenz auf die eindeutige Ausprägung (1) definiert ist, kann rückwirkend keine eindeutige Ausprägung oder ein Array enthalten, das kürzer als das definierte Array ist.

```
READ FINANCE
OBTAIN CREDIT-CARD (1-5)
MOVE 'ONE' TO CREDIT-CARD (1)
DISPLAY CREDIT-CARD (1) CREDIT-CARD (1-5)
```

Ausgabe:

```
  CREDIT-CARD  CREDIT-CARD
-----
ONE           ONE
             AMERICAN EXPRESS

ONE           ONE
             AMERICAN EXPRESS

ONE           ONE
             AMERICAN EXPRESS

ONE           ONE
```

Die individuelle Referenz auf CREDIT-CARD (1) ist innerhalb des im OBTAIN-Statement definierten Arrays enthalten.

```
MOVE (1) TO INDEX
READ FINANCE
DISPLAY CREDIT-CARD (1-5) CREDIT-CARD (INDEX)
```

Ausgabe:

CREDIT-CARD	CREDIT-CARD
DINERS CLUB AMERICAN EXPRESS	DINERS CLUB
AVIS AMERICAN EXPRESS	AVIS
HERTZ AMERICAN EXPRESS	HERTZ
UNITED AIR TRAVEL	UNITED AIR TRAVEL

Die Referenz auf CREDIT-CARD mittels der variablen Index-Notation ist nicht innerhalb des Arrays enthalten.

```
RESET A(A20) B(A20) C(A20)
MOVE 2 TO I (N3)
MOVE 3 TO J (N3)
READ FINANCE
OBTAIN CREDIT-CARD (1:3) CREDIT-CARD (I:I+2) CREDIT-CARD (J:J+2)
FOR K (N3) = 1 TO 3
  MOVE CREDIT-CARD (1.K) TO A
  MOVE CREDIT-CARD (I.K) TO B
  MOVE CREDIT-CARD (J.K) TO C
  DISPLAY A B C
LOOP /* FOR
LOOP / * READ
END
```

Ausgabe:

A	B	C
CARD 01	CARD 02	CARD 03
CARD 02	CARD 03	CARD 04
CARD 03	CARD 04	CARD 05

Die drei Arrays können einzeln aufgerufen werden, indem Sie den eindeutigen Basis-Index als Kennzeichner für den Index-Ausdruck benutzen.

Ungültiges Beispiel 1

```
READ FINANCE
OBTAIN CREDIT-CARD (1-10)
FOR I 1 10
MOVE CREDIT-CARD (I) TO A(A20)
WRITE A
END
```

Das obige Beispiel erzeugt die Fehlermeldung NAT1006 (Wert für variablen Index = 0), weil als der Datensatz mit `READ` gelesen wurde, der Index `I` noch den Wert 0 enthielt.

Auf jeden Fall würden in dem obigen Beispiel nicht die ersten 10 Ausprägungen von `CREDIT-CARD` ausgegeben werden, weil die einzelne Ausprägung mit dem variablen Index nicht im Array enthalten sein kann und der variable Index (`I`) nur ausgewertet wird, wenn der nächste Datensatz gelesen wird.

Im Folgenden finden Sie die korrekte Methode, um das oben Beschriebene auszuführen:

```
READ FINANCE
OBTAIN CREDIT-CARD (1-10)
FOR I 1 10
MOVE CREDIT-CARD (1.I) TO A (A20)
WRITE A
END
```

Ungültiges Beispiel 2

```
READ FINANCE
FOR I 1 10
WRITE CREDIT-CARD (I)
END
```

Das obige Beispiel erzeugt die Fehlermeldung NAT1006, weil der Index I Null ist, wenn der Datensatz im `READ`-Statement gelesen wird.

Im Folgenden wird die korrekte Methode verwendet, um das oben Beschriebene auszuführen:

```
READ FINANCE
FOR I 1 10
GET SAME
WRITE CREDIT-CARD (0030/I)
END
```

Das `GET SAME`-Statement ist erforderlich, um den Datensatz erneut zu lesen, nachdem der variable Index in der `FOR`-Schleife aktualisiert worden ist.

Beispiele

- [Beispiel 1 — OBTAIN-Statement](#)
- [Beispiel 2 — OBTAIN-Statement mit mehreren Bereichen](#)

Beispiel 1 — OBTAIN-Statement

```
** Example 'OBTEX1': OBTAIN
*****
RESET #INDEX (I1)
*
LIMIT 5
READ EMPLOYEES BY CITY
  OBTAIN SALARY (1:4)
  /*
  IF SALARY (4) GT 0 DO
    WRITE '=' NAME / 'SALARIES (1:4):' SALARY (1:4)
    FOR #INDEX 1 TO 4
      WRITE 'SALARY' #INDEX SALARY (1.#INDEX)
    LOOP
  SKIP 1
DOEND
```

OBTAIN

```
LOOP
*
END
```

Ausgabe des Programms OBTEX1:

```
Page      1                                05-02-08  13:37:48
NAME: SENKO
SALARIES (1:4):      31500      29900      28100      26600
SALARY   1      31500
SALARY   2      29900
SALARY   3      28100
SALARY   4      26600

NAME: HAMMOND
SALARIES (1:4):      22000      20200      18700      17500
SALARY   1      22000
SALARY   2      20200
SALARY   3      18700
SALARY   4      17500
```

Beispiel 2 — OBTAIN-Statement mit mehreren Bereichen

```
** Example 'OBTEX2': OBTAIN (with multiple ranges)
*****
RESET #INDEX (I1) #K (I1)
*
#INDEX := 2
#K := 3
*
LIMIT 2
*
READ EMPLOYEES BY CITY
  OBTAIN SALARY (1:5)
    SALARY (#INDEX:#INDEX+3)
/*
IF SALARY (5) GT 0 DO
  WRITE '=' NAME
  WRITE 'SALARIES (1-5):' SALARY (1:5) /
  WRITE 'SALARIES (2-5):' SALARY (#INDEX:#INDEX+3)
  WRITE 'SALARIES (2-5):' SALARY (#INDEX.1:4) /
  WRITE 'SALARY 3:' SALARY (3)
  WRITE 'SALARY 3:' SALARY (#K)
  WRITE 'SALARY 4:' SALARY (#INDEX.#K)
DOEND
LOOP
```

Ausgabe des Programms OBTEX2:


```
Page      1                                     05-02-08  13:38:31
NAME: SENKO
SALARIES (1-5):      31500      29900      28100      26600      25200
SALARIES (2-5):      29900      28100      26600      25200
SALARIES (2-5):      29900      28100      26600      25200
SALARY 3:           28100
SALARY 3:           28100
SALARY 4:           26600
```

Weitere Beispiele zur Benutzung des OBTAIN-Statements, siehe *Datenbank-Array referenzieren im Leitfaden zur Programmierung*.

87 ON ERROR

▪ Funktion	568
▪ Einschränkung	568
▪ Syntax-Beschreibung	569
▪ ON ERROR-Verarbeitung in Unterprogrammen	569
▪ Systemvariablen *ERROR-NR und *ERROR-LINE	569
▪ Beispiel	570

Structured Mode-Syntax

```
ON ERROR
  statement ...
END-ERROR
```

Reporting Mode-Syntax

```
ON ERROR { statement ...
           DO statement ... DOEND }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [DECIDE FOR](#) | [DECIDE ON](#) | [IF](#) | [IF SELECTION](#)

Funktion

Das Statement `ON ERROR` dient dazu, Laufzeitfehler abzufangen, welche andernfalls eine Natural-Fehlermeldung, den Abbruch des gerade ausgeführten Programms und die Rückkehr zum Kommandoeingabe-Modus zur Folge hätten.

Sobald die Ausführung der in einem `ON ERROR`-Statement-Block angegebenen Statements beginnt, ist der normale Programmablauf unterbrochen und kann nicht fortgesetzt werden. Eine Ausnahme bildet Fehler 3145 (angeforderter Datensatz im Hold), bei dem über ein `RETRY`-Statement die Verarbeitung genau an der Stelle, an der sie unterbrochen wurde, fortgesetzt werden kann.

Dieses Statement ist nicht prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Einschränkung

Jedes Natural-Objekt darf maximal ein `ON ERROR`-Statement enthalten.

Syntax-Beschreibung

<i>statement...</i>	<p>Festlegung der ON ERROR-Verarbeitung:</p> <p>Um die Verarbeitung zu definieren, die stattfinden soll, wenn eine ON ERROR-Bedingung aufgetreten ist, können Sie eines oder mehrere Statements angeben.</p> <p>Verlassen eines ON ERROR-Blocks:</p> <p>Ein ON ERROR-Statement-Block kann mit einem der Statements <code>FETCH</code>, <code>STOP</code>, <code>TERMINATE</code>, <code>RETRY</code> or <code>ESCAPE ROUTINE</code> verlassen werden. Wird der Block nicht mit einem dieser Statements verlassen, gibt Natural eine entsprechende Fehlermeldung aus und bricht die Ausführung des Programms ab.</p>
END-ERROR	Das für Natural reservierte Wort <code>END-ERROR</code> muss zum Beenden eines ON-ERROR-Statement-Blocks benutzt werden.

ON ERROR-Verarbeitung in Unterprogrammen

Wird mittels `CALLNAT`, `PERFORM` oder `FETCH RETURN` eine Unterprogramm-Struktur aufgebaut, so darf jedes Modul dieser Struktur ein ON ERROR-Statement enthalten.

Tritt ein Fehler auf, so verfolgt Natural die Unterprogramm-Struktur automatisch zurück und wählt das erste in einem Unterprogramm gefundene ON ERROR-Statement zur Ausführung aus. Enthält keines der Module ein ON ERROR-Statement, gibt Natural eine entsprechende Fehlermeldung aus, und es erfolgt – wie oben beschrieben – ein Programmabbruch.

Systemvariablen *ERROR-NR und *ERROR-LINE

Die folgenden Natural-Systemvariablen können in Zusammenhang mit dem ON ERROR-Statement (wie in dem folgenden **Beispiel** gezeigt) benutzt werden:

*ERROR-NR	Enthält die Fehlernummer des von Natural entdeckten Fehlers
*ERROR-LINE	Enthält die Nummer der Sourcecode-Zeile, in der das Statement steht, das den Fehler verursacht hat.

Beispiel

```

** Example 'ONEEX1': ON ERROR
**
**
CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
*
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
*
REPEAT
  INPUT 'ENTER NAME:' #NAME
  IF #NAME = ' '
    STOP
  END-IF
  FIND EMPLOY-VIEW WITH NAME = #NAME
  INPUT (AD=M) 'ENTER NEW VALUES:' ///
    'NAME:' NAME /
    'CITY:' CITY

  UPDATE
  END TRANSACTION
/*
ON ERROR
  IF *ERROR-NR = 3009
    WRITE 'LAST TRANSACTION NOT SUCCESSFUL'
      / 'HIT ENTER TO RESTART PROGRAM'
    FETCH 'ONEEX1'
  END-IF
  WRITE 'ERROR' *ERROR-NR 'OCCURRED IN PROGRAM' *PROGRAM
    'AT LINE' *ERROR-LINE
  FETCH 'MENU'
END-ERROR
/*
END-FIND
END-REPEAT
END

```

88

OPEN CONVERSATION

▪ Funktion	572
▪ Syntax-Beschreibung	572
▪ Weitere Informationen und Beispiele	573

```
OPEN CONVERSATION USING [SUBPROGRAMS] {operand1} ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Gehört zur Funktionsgruppe: *Natural Remote Procedure Call*

Verwandte Statements: `CLOSE CONVERSATION` | `DEFINE DATA CONTEXT`

Funktion

Das Statement `OPEN CONVERSATION` wird im Zusammenhang mit Natural Remote Procedure Call (RPC) verwendet. Es ermöglicht dem Client, eine Konversation zu öffnen und die Remote-Subprogramme anzugeben, die an der Konversation beteiligt sein sollen.

Wenn das `OPEN CONVERSATION`-Statement ausgeführt wird, weist es der Systemvariablen `*CONVID` eine eindeutige ID zu, die die Konversation identifiziert.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A	A	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Subprogramm-Namen:</p> <p>Als <i>operand1</i> geben Sie die Namen der Subprogramme an, die an der Konversation beteiligt sein sollen.</p> <p>Der Name eines Subprogramms kann entweder als 1 bis 8 Zeichen lange Konstante oder als alphanumerische Variable mit Länge 1 bis 8 angegeben werden.</p>
-----------------	--

Weitere Informationen und Beispiele

Siehe folgende Abschnitte in der *Natural Remote Procedure Call (RPC)*-Dokumentation:

- *Natural RPC Operation in Conversational Mode*
- *Using a Conversational RPC*

89 OPEN DIALOG

▪ Funktion	576
▪ Syntax-Beschreibung	576
▪ Weitere Informationen und Beispiele	578

Syntax-Element-Beschreibung:

<i>operand1</i>	Dialogname: <i>operand1</i> ist der Name des zu öffnenden Dialogs. Wenn die PARAMETERS-Klause1 benutzt wird, muss <i>operand1</i> eine Konstante sein.	
<i>operand2</i>	Handle-Name: <i>operand2</i> ist der Handle-Name des Parent.	
<i>operand3</i>	Dialog-ID: <i>operand3</i> ist ein eindeutiger Bezeichner, der nach der Dialogerstellung zurückgegeben wird. Er muss mit Format/Länge I4 definiert werden.	
	Übergabe von Parametern an den Dialog: Wenn ein Dialog geöffnet wird, können Parameter an diesen Dialog übergeben werden.	
<i>operand4</i>	Als <i>operand4</i> geben Sie die Parameter an, die an den Dialog übergeben werden.	
<i>PARAMETERS-clause</i>	Selektive Übergabe von Parametern: Mit der <i>PARAMETERS-Klause1</i> können Parameter selektiv übergeben werden. Siehe PARAMETERS-Klausel unten. Anmerkung: Sie können die <i>PARAMETERS-Klause1</i> nur benutzen, wenn <i>operand1</i> eine Konstante ist und der Dialog katalogisiert ist.	
<i>nX</i>	Angabe zu überspringender Parameter: Mit der Notation <i>nX</i> können Sie angeben, dass die nächsten <i>n</i> Parameter übersprungen werden (zum Beispiel 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); dies bedeutet, dass für die nächsten <i>n</i> Parameter keine Werte an den Dialog übergeben werden. Ein zu überspringender Parameter muss im DEFINE DATA PARAMETER -Statement des Dialogs mit dem Schlüsselwort OPTIONAL definiert werden. OPTIONAL bedeutet, dass ein Wert von dem aufrufenden Objekt an einen solchen Parameter übergeben werden kann - aber nicht muss.	
AD=	Definition von Attributen: Wenn <i>operand4</i> eine Variable ist, können Sie sie folgendermaßen kennzeichnen:	
	AD=O	Nicht änderbar, siehe Session-Parameter AD=O.
	AD=M	Änderbar, siehe Session-Parameter AD=M.
	AD=A	Nur Eingabe, siehe Session-Parameter AD=A.
	<i>operand4</i> kann nicht explizit angegeben werden, wenn <i>operand4</i> eine Konstante ist. AD=O gilt immer für Konstanten.	

PARAMETERS-Klausel

```
PARAMETERS {parameter-name = operand4}...  
END-PARAMETERS
```

Syntax-Element-Beschreibung:

<i>parameter-name</i>	Der Name des Parameters wie er in der Parameter Data Area des Dialogs definiert ist. Anmerkung: Es führt zu einem Laufzeitfehler, wenn der Wert eines Parameters, der mit AD=0 markiert und „By Reference“ übergeben wird, im Dialog geändert wird.
<i>operand4</i>	Als <i>operand4</i> geben Sie die Parameter an, die an den Dialog übergeben werden.
END-PARAMETERS	Das für Natural reservierte Schlüsselwort END-PARAMETERS muss zum Beenden der <i>PARAMETERS-Klausel</i> verwendet werden.

Weitere Informationen und Beispiele

Siehe *Event-Driven Programming Techniques* im Leitfaden zur Programmierung.

90 OPTIONS

■ Funktion	580
------------------	-----

OPTIONS *parameter* ...

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `OPTIONS` kann verwendet werden, um Kompilierungsoptionen als Parameter für das aktuelle Natural-Programmierobjekt anzugeben. Es handelt sich um dieselben Optionen, die in einer Natural-Session mit dem Systemkommando `COMPOPT` verwendet werden können.



Anmerkung: Es stehen keine Großrechner-spezifischen Optionen zur Verfügung. Aus Kompatibilitätsgründen, zum Beispiel beim Programmieren einer plattformübergreifenden Anwendung, werden solche Optionen zur Kompilierzeit ignoriert.

91

PARSE XML

▪ Funktion	582
▪ Syntax-Beschreibung	583
▪ Beispiele	586

```

PARSE XML operand1 [INTO [PATH operand2] [NAME operand3] [VALUE operand4]]
  [[NORMALIZE] NAMESPACE operand5 PREFIX operand6]
  statement...
END-PARSE (structured mode only)
[LOOP] (reporting mode only)
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Gehört zur Funktionsgruppe: *Internet und XML*

Funktion

Das Statement `PARSE XML` ermöglicht es Ihnen, XML-Dokumente aus einem Natural-Programm zu parsen. Als Voraussetzung für die Benutzung dieses Statements muss die Library ICU installiert sein. Siehe auch *Statements für Internet- und XML-Zugang im Leitfaden zur Programmierung*.

Es empfiehlt sich, dynamische Variablen zu benutzen, wenn Sie das Statement `PARSE XML` verwenden. Der Grund dafür ist, dass es unmöglich ist, die Länge einer statischen Variablen zu ermitteln. Der Einsatz von statischen Variablen könnte wiederum zum Abschneiden des Wertes führen, der in die Variable geschrieben werden soll.

Informationen bezüglich Unicode-Support finden Sie im Abschnitt `PARSE XML` in der *Unicode and Code Page Support*-Dokumentation.

Markierungen

Im Folgenden finden Sie Bezeichner, die in Pfad-Zeichenketten zur Darstellung der verschiedenen Datentypen in einem XML-Dokument (auf ASCII-basierten Systemen) benutzt werden:

Markierung	XML-Daten	Position in der Pfad-Zeichenkette
?	Verarbeitungsanweisung (außer bei <code><?XML...?></code>)	Ende
!	Kommentar	Ende
C	CDATA-Abschnitt	Ende
@	Attribut (auf Großrechnern: § oder @, je nach Code Page und Terminal Emulation)	vor dem Attribut-Namen
/	Abschließender Tag und/oder Trennzeichen für Parent-Namen in einem Pfad	Ende oder zwischen Parent-Namen
\$	Geparste Daten-Zeichendatenkette	Ende

Wenn Sie diese zusätzlichen Markierungen im Pfad-String benutzen, ist es leichter, die verschiedenen Elemente des XML-Dokuments im Ausgabedokument zu identifizieren.

Global Namespace

Zur Angabe des Global Namespace verwenden Sie einen Doppelpunkt (:) als Präfix und eine leere URI.

Zugehörige Systemvariablen

Die folgenden Natural-Systemvariablen werden für jedes abgesetzte PARSE-Statement automatisch erzeugt:

- *PARSE - TYPE
- *PARSE - LEVEL
- *PARSE - ROW
- *PARSE - COL
- *PARSE - NAMESPACE - URI

Durch Angabe der Notation (*r*) hinter *PARSE - TYPE, *PARSE - LEVEL, *PARSE - ROW, *PARSE - COL und *PARSE - NAMESPACE - URI können Sie den Statement-Label bzw. die Sourcecode-Zeilenummer des Statements angeben, in dem bzw. in der die PARSE-Anweisung abgesetzt wurde. Wenn (*r*) nicht angegeben wird, stellt die betreffende Systemvariable die Systemvariable der XML-Daten dar, die gerade in der zur Zeit aktiven PARSE-Verarbeitungsschleife abgearbeitet werden.

Weitere Informationen über diese Systemvariablen finden Sie in der *Systemvariablen*-Dokumentation.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S			A	U	B											ja	nein
<i>operand2</i>		S			A	U	B											ja	ja
<i>operand3</i>		S			A	U	B											ja	ja
<i>operand4</i>		S			A	U	B											ja	ja
<i>operand5</i>		S	A		A	U	B											ja	ja
<i>operand6</i>		S	A		A	U	B											ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<p><i>operand1</i> stellt das betreffende XML-Dokument dar. Das XML-Dokument kann nicht geändert werden, während es vom Parser abgearbeitet wird.</p> <p>Wenn Sie versuchen, während des Parse-Vorgangs das XML-Dokument zu ändern (indem Sie es zum Beispiel überschreiben), wird eine Fehlermeldung angezeigt.</p>
<i>operand2</i>	<p><i>operand2</i> stellt den Pfad der Daten im XML-Dokument dar.</p> <p>Der Pfad (PATH) enthält den Namen des identifizierten XML-Teils, die Namen aller Parents (übergeordneten Elemente), sowie den Typ des XML-Teils.</p> <p>Anmerkung: Die mit dem Pfad angegebenen Informationen erleichtern den Aufbau einer Baumstruktur.</p> <p>Siehe auch Beispiel 1 – Benutzung von operand2.</p>
<i>operand3</i>	<p><i>operand3</i> stellt den Namen (NAME) eines Datenelements im XML-Dokument dar.</p> <p>Wenn NAME keinen Wert hat, dann wird die damit verbundene dynamische Variable auf *LENGTH()=0 gesetzt, welche eine mit einem Leerzeichen aufgefüllte statische Variable ist.</p> <p>Siehe auch Beispiel 2 – Benutzung von operand3.</p>
<i>operand4</i>	<p><i>operand4</i> stellt den Wert (VALUE) eines Datenelements im XML-Dokument dar.</p> <p>Ist kein Wert vorhanden, wird eine gegebene dynamische Variable auf *LENGTH()=0 gesetzt, welche eine mit einem Leerzeichen aufgefüllte statische Variable ist.</p> <p>Siehe auch Beispiel 3 – Benutzung von operand4.</p>
<i>operand5</i> und <i>operand6</i>	<p>Der eindeutige Namen gewährleistende Namespace-URI oder Uniform Resource Identifier (<i>operand5</i>) und das Namespace-PREFIX (<i>operand6</i>) werden während der Laufzeit kopiert. Deshalb beeinflusst eine Änderung der Arrays für Namespace-Zuordnungen in der PARSE-Schleife nicht den Parser.</p>
NORMALIZE NAMESPACE PREFIX	<p><i>operand5</i> und <i>operand6</i> sind eindimensionale Arrays mit einer gleichen Anzahl von Ausprägungen.</p> <p>Namespace-Normalisierung ist eine Funktion des PARSE-Statements. Mit XML können Namespaces für die Element-Namen definiert werden:.</p> <pre style="background-color: #f0f0f0; padding: 5px;"><code><myns:myentity xmlns:myns="http://myuri" /></code></pre> <p>Die Namespace-Definition besteht aus zwei Teilen:</p> <ul style="list-style-type: none"> ■ einem Namespace-PREFIX (myns in diesem Fall) und ■ ein URI (myuri) zur Definition des Namespace. <p>Der Namespace-PREFIX ist Teil des Elementnamens. Dies bedeutet, dass für das PARSE-Statement, vor allem für <i>operand2</i>, die generierten PATH-Strings vom Namespace-PREFIX abhängig sind. Wenn der Pfad in einem Natural-Programm zur Angabe</p>

bestimmter Tags benutzt wird, dann funktioniert das nicht, wenn ein XML-Dokument den korrekten Namespace (URI), jedoch einen anderen `PREFIX` verwendet..

Bei der Namespace-Normalisierung können alle Namespace-Präfixe auf Standardwerte gesetzt werden, die in der Namespace-Klausel definiert wurden. Der erste Eintrag wird dann benutzt, wenn ein URI mehr als einmal angegeben wird. Wenn mehr als ein Präfix im XML-Dokument benutzt wird, dann wird nur das erste für die Ausgabe berücksichtigt. Der Rest wird ignoriert.

Die `NAMESPACE`-Klausel enthält Paare von Namespace-URIs und -Präfixe. Zum Beispiel:

```
uri(1) := 'http://namespaces.softwareag.com/natural/demo'  
pre(1) := 'nat:'
```

Wenn der Namespace in einem XML-Dokument definiert wird, prüft der Parser, ob dieser Namespace (URI) in der Normalisierungs-Tabelle vorhanden ist. Das Präfix der Normalisierungs-Tabelle wird für alle Ausgabedaten vom `PARSE`-Statement anstatt des im XML-Dokument definierten Namespace benutzt.

Siehe auch:

- [Beispiel 4 – Benutzung von operand5 und operand6](#)
- [Beispiel 5 – Benutzung von operand5 und operand6 mit Namespace-Normalisierung](#)

Zusätzliche Informationen zu `PREFIX`:

Außerdem gilt Folgendes für die Präfix-Definition:

- Die Präfix-Definition beim Array für die Namespace-Normalisierung muss stets mit einem Doppelpunkt (:) enden, da dies die Zeichenkette ist, die ersetzt wird.
- Ein Präfix oder ein URI kann nur einmal bei einem Array für eine Namespace-Normalisierung vorkommen.
- Enthält ein Präfix oder der Namespace-URI nachfolgende Leerzeichen (z.B. wenn eine statische Variable verwendet wird), werden die nachfolgenden Leerzeichen entfernt, bevor der externe Parser aufgerufen wird.
- Wenn die Präfix-Definition bei der Namespace-Normalisierung nur einen Doppelpunkt (:) enthält, dann wird das Namespace-Präfix gelöscht.

Beispiele

- Beispiel 1 — Benutzung von operand2
- Beispiel 2 — Benutzung von operand3
- Beispiel 3 — Benutzung von operand4
- Beispiel 4 — Benutzung von operand5 und operand6
- Beispiel 5 — Benutzung von operand5 und operand6 mit Namespace-Normalisierung

Beispiel 1 — Benutzung von operand2

Der folgende XML-Code

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
        '<employee personnel-id="30016315" >'-
        '<full-name>'-
        '<!--this is just a comment-->'-
        '<first-name>RICHARD</first-name>'-
        '<name>FORDHAM</name>'-
        '</full-name>'-
        '</employee>'
```

wird durch folgenden Natural-Code verarbeitet:

```
PARSE XML myxml INTO PATH mypath
PRINT mypath
END-PARSE
```

und erzeugt die folgende Ausgabe:

```
employee
employee/@personnel-id
employee/full-name
employee/full-name/!
employee/full-name/first-name
employee/full-name/first-name/$
employee/full-name/first-name//
employee/full-name/name
employee/full-name/name/$
employee/full-name/name//
employee/full-name//
employee//
```

Beispiel 2 — Benutzung von operand3

Der folgende XML-Code

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
        '<employee personnel-id="30016315" >'-
        '<full-name>'-
        '<!--this is just a comment-->'-
        '<first-name>RICHARD</first-name>'-
        '<name>FORDHAM</name>'-
        '</full-name>'-
        '</employee>'
```

wird durch folgenden Natural-Code verarbeitet:

```
PARSE XML myxml INTO PATH mypath NAME myname
  DISPLAY (AL=39) mypath myname
END-PARSE
```

und erzeugt die folgende Ausgabe:

MYPATH	MYNAME
employee	employee
employee/@personnel-id	personnel-id
employee/full-name	full-name
employee/full-name/!	
employee/full-name/first-name	first-name
employee/full-name/first-name/\$	
employee/full-name/first-name//	first-name
employee/full-name/name	name
employee/full-name/name/\$	
employee/full-name/name//	name
employee/full-name//	full-name
employee//	employee

Beispiel 3 — Benutzung von operand4

Der folgende XML-Code

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
        '<employee personnel-id="30016315" >'-
        '<full-name>'-
        '<!--this is just a comment-->'-
        '<first-name>RICHARD</first-name>'-
        '<name>FORDHAM</name>'-
        '</full-name>'-
        '</employee>'
```

wird durch folgenden Natural-Code verarbeitet:

```
PARSE XML myxml INTO PATH mypath VALUE myvalue
  DISPLAY (AL=39) mypath myvalue
END-PARSE
```

und erzeugt die folgende Ausgabe:

MYPATH	MYVALUE
employee	
employee/@personnel-id	30016315
employee/full-name	
employee/full-name/!	this is just a comment
employee/full-name/first-name	
employee/full-name/first-name/\$	RICHARD
employee/full-name/first-name//	
employee/full-name/name	
employee/full-name/name/\$	FORDHAM
employee/full-name/name//	
employee/full-name//	
employee//	

Beispiel 4 — Benutzung von operand5 und operand6

Der folgende XML-Code

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>'-
        '<nat:employee nat:personnel-id="30016315" '-
        '  xmlns:nat="http://namespaces.softwareag.com/natural/demo">'-
        '<nat:full-Name>'-
        '<nat:first-name>RICHARD</nat:first-name>'-
        '<nat:name>FORDHAM</nat:name>'-
        '</nat:full-Name>'-
        '</nat:employee>'
```

wird durch folgenden Natural-Code verarbeitet:

```
PARSE XML myxml INTO PATH mypath
  PRINT mypath
END-PARSE
```

und erzeugt die folgende Ausgabe:

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
nat:employee/nat:full-Name//
nat:employee//
```

Beispiel 5 — Benutzung von operand5 und operand6 mit Namespace-Normalisierung

Wird `NORMALIZE NAMESPACE` verwendet, erzeugt dasselbe XML-Dokument wie in Beispiel 4 mit einem anderen `NAMESPACE PREFIX` genau dieselbe Ausgabe.

XML-Code:

```
myxml := '<?xml version="1.0" encoding="ISO-8859-1" ?>' -
        '<natural:employee natural:personnel-id="30016315"' -
        ' xmlns:natural="http://namespaces.softwareag.com/natural/demo">' -
        '<natural:full-Name>' -
        '<natural:first-name>RICHARD</natural:first-name>' -
        '<natural:name>FORDHAM</natural:name>' -
        '</natural:full-Name>' -
        '</natural:employee>'
```

Natural-Code:

```
uri(1) := 'http://namespaces.softwareag.com/natural/demo'
pre(1) := 'nat:'
*
PARSE XML myxml INTO PATH mypath NORMALIZE NAMESPACE uri(*) PREFIX pre(*)
PRINT mypath
END-PARSE
```

Ausgabe des obigen Programms

```
nat:employee
nat:employee/@nat:personnel-id
nat:employee/@xmlns:nat
nat:employee/nat:full-Name
nat:employee/nat:full-Name/nat:first-name
nat:employee/nat:full-Name/nat:first-name/$
nat:employee/nat:full-Name/nat:first-name//
nat:employee/nat:full-Name/nat:name
nat:employee/nat:full-Name/nat:name/$
nat:employee/nat:full-Name/nat:name//
nat:employee/nat:full-Name//
nat:employee//
```

92

PASSW

▪ Funktion	592
▪ Syntax-Beschreibung	592

```
PASSW=operand1
```

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | HISTOGRAM | GET | GET SAME | GET TRANSACTION | LIMIT | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Mit dem Statement `PASSW` geben Sie ein Passwort an, um auf eine passwortgeschützte Adabas-Datei zugreifen zu können.



Anmerkung: Dieses Passwort kann mittels der `PASSWORD`-Klausel der Datenbankzugriffs-Statements `FIND`, `GET`, `HISTOGRAM`, `READ`, `STORE` überschrieben werden.

Anmerkungen bezüglich Natural Security

Im Security-Profil einer Library können Sie ein standardmäßiges Adabas-Passwort angeben (wie in der *Natural Security*-Dokumentation beschrieben); dieses Passwort gilt für alle Datenbankzugriffs-Statements, für die weder ein eigenes Passwort angegeben ist noch ein `PASSW`-Statement gilt, und zwar nicht nur in der betreffenden Library, sondern darüber hinaus auch, wenn Sie anschließend in andere Libraries wechseln, in deren Security-Profilen kein Passwort angegeben ist.

Syntax-Beschreibung

Operanden-Definitionstabelle:

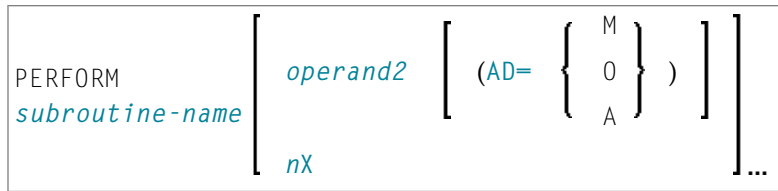
Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Passwort:</p> <p>Das Passwort (<i>operand1</i>) kann entweder als alphanumerische Konstante angegeben werden oder als alphanumerische Variable, welche das Passwort enthält. Das Passwort darf bis zu acht Zeichen lang sein und darf keine Sonderzeichen oder Leerzeichen enthalten. Wird es als Konstante angegeben, muss es in Apostrophen stehen.</p> <p>Das mit dem PASSW-Statement angegebene Passwort gilt für alle Datenbankzugriffs-Statements (FIND, GET, HISTOGRAM, READ, STORE), in denen kein eigenes Passwort angegeben ist. Es gilt, bis mit einem anderen PASSW-Statement ein anderes Passwort angegeben wird bzw. bis zum Ende der Natural-Session.</p> <p>Ein mit einem bestimmten Datenbank-Statement angegebenes Passwort gilt nur für das jeweilige Statement, nicht für irgendwelche nachfolgenden Statements.</p>
-----------------	---

93 PERFORM

▪ Funktion	596
▪ Syntax-Beschreibung	597
▪ Beispiele	599



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [CALL](#) | [CALL FILE](#) | [CALL LOOP](#) | [CALLNAT](#) | [DEFINE SUBROUTINE](#) | [ESCAPE](#) | [FETCH](#)

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Das Statement `PERFORM` dient dazu, eine Natural-**Subroutine** aufzurufen.

Verschachtelte `PERFORM`-Statements

Eine aufgerufene Subroutine kann ihrerseits mit einem `PERFORM`-Statement eine andere Subroutine aufrufen. Wieviele Ebenen eine derartige Verschachtelung mehrerer `PERFORM`-Statements erreichen darf, hängt vom benötigten Speicherplatz ab.

Eine Subroutine kann auch sich selbst aufrufen (rekursive Subroutine). Falls eine rekursive externe Subroutine Datenbankzugriffe beinhaltet, sorgt Natural automatisch dafür, dass diese als getrennte logische Operationen durchgeführt werden.

Parameter-Übertragung mit dynamischen Variablen

Siehe [CALLNAT](#)-Statement.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand2</i>	C S A G	A U N P I F B D T L C G O	ja	ja

Syntax-Element-Beschreibung:

<i>subroutine-name</i>	<p>Aufzurufende Subroutine:</p> <p>Für einen Subroutinen-Namen (maximal 32 Zeichen) gelten dieselben Namenskonventionen wie für Benutzervariablen (siehe <i>Namenskonventionen für Benutzervariablen</i> in der Dokumentation <i>Natural Studio benutzen</i>).</p> <p>Der Subroutine-Name ist unabhängig vom Namen des Moduls, in dem die Subroutine definiert wird (er kann identisch sein, muss es aber nicht unbedingt sein).</p> <p>Die aufzurufende Subroutine muss mit einem <code>DEFINE SUBROUTINE</code>-Statement definiert werden. Es kann sich dabei um eine interne oder externe Subroutine handeln (siehe <code>DEFINE SUBROUTINE</code>-Statement).</p> <p>Innerhalb eines Objekts können nicht mehr als 50 externe Subroutinen referenziert werden.</p> <p>Von einer Subroutine benutzbare Daten:</p> <ul style="list-style-type: none"> ■ Interne Subroutinen: Es ist nicht möglich, mit dem <code>PERFORM</code>-Statement Parameter explizit vom aufrufenden Programm an eine programmintern definierte Subroutine zu übergeben. Eine programminterne Subroutine kann auf die im selben Objektmodul verwendete Local Data Area sowie auf die derzeit verwendete Global Data Area zugreifen. ■ Externe Subroutinen: Eine programmextern definierte Subroutine kann Daten aus der Global Data Area des aufrufenden Objekts verwenden. Außerdem können Sie mit dem <code>PERFORM</code>-Statement Parameter vom aufrufenden Objekt an die aufgerufene Subroutine übergeben (siehe <i>operand2</i>); dadurch können Sie die Größe der Global Data Area entsprechend klein halten.
<i>operand2</i>	<p>Übergabe von Parametern an die externe Subroutine:</p> <p>Wenn Sie mit dem <code>PERFORM</code>-Statement eine externe Subroutine aufrufen, können Sie mit dem <code>PERFORM</code>-Statement einen oder mehrere Parameter (<i>operand2</i>) vom aufrufenden Objekt an die externe Subroutine übergeben. Für interne Subroutinen kann <i>operand2</i> nicht angegeben werden.</p>

Wenn Parameter übergeben werden, muss die Struktur der Parameterliste in einem `DEFINE DATA`-Statement definiert werden.

Standardmäßig erfolgt die Übergabe der Parameter durch Referenzierung (*By Reference*), d.h. die Daten werden über Adress-Parameter übergeben, die Parameterwerte selbst werden nicht übertragen. Es besteht aber auch die Möglichkeit, Parameter *By Value* zu übergeben, d.h. die Parameterwerte selbst zu übergeben. Hierzu definieren Sie die betreffenden Felder im `DEFINE DATA PARAMETER`-Statement der Subroutine mit der Option `BY VALUE` bzw. `BY VALUE RESULT`).

- Für die Parameterübergabe durch Referenzierung (*By Reference*) gilt: Reihenfolge, Format und Länge der Parameter im aufrufenden Objekt müssen genau den Angaben im `DEFINE DATA PARAMETER`-Statement der aufgerufenen Subroutine entsprechen. Die Namen der Variablen im aufrufenden Objekt und der Subroutine können unterschiedlich sein.
- Für die Parameterübergabe der Parameterwerte selbst (*By Value*) gilt: Die Reihenfolge der Parameter im aufrufenden Objekt muss der Reihenfolge im `DEFINE DATA PARAMETER`-Statement der aufgerufenen Subroutine entsprechen. Formate und Längen der Variablen im aufrufenden Objekt und in der Subroutine können unterschiedlich sein, müssen aber datenübertragungskompatibel sein. Die Namen der Variablen im aufrufenden Objekt und in der Subroutine können unterschiedlich sein.

Um Parameterwerte, die in der Subroutine verändert wurden, an das aufrufende Objekt zurückgeben zu können, müssen Sie die betreffenden Felder mit `BY VALUE RESULT` definieren.

Mit `BY VALUE` (ohne `RESULT`) ist es nicht möglich, veränderte Parameterwerte an das aufrufende Objekt zurückzugeben (unabhängig von der Attribut-Definition (AD-Parameterangabe; vgl. unten).

Anmerkung: Intern wird bei `BY VALUE` eine Kopie der Parameterwerte erzeugt. Die Subroutine greift auf diese Kopie zu und kann sie modifizieren, was aber keinen Einfluss auf die Originalparameterwerte im aufrufenden Objekt hat. Bei `BY VALUE RESULT` wird ebenfalls eine Kopie erzeugt, aber nach Beendigung der Subroutine überschreiben die (modifizierten) Werte der Kopie die Originalparameterwerte.

Bei beiden Arten der Parameterübergabe sind folgende Punkte zu beachten:

- Eine Gruppe darf in der Parameter Data Area der aufgerufenen Subroutine innerhalb eines `REDEFINE`-Statement-Blocks redefiniert werden.
- Bei der Übergabe eines Arrays muss die Anzahl seiner Dimensionen und Ausprägungen in der Parameter Data Area der Subroutine denen in der `PERFORM`-Parameterliste entsprechen.

Anmerkung: Wenn mehrere Ausprägungen eines Arrays, das als Teil einer indizierten Gruppe definiert ist, mit dem `PERFORM`-Statement übergeben werden, dürfen die entsprechenden Felder in der Parameter Data Area der Subroutine nicht redefiniert werden, da sonst die falschen Adressen übergeben werden.

AD=	Definition von Attributen:	
	Wenn <i>operand2</i> eine Variable ist, können Sie sie folgendermaßen kennzeichnen:	
	AD=0	Nicht modifizierbar, siehe Session-Parameter AD=0. Anmerkung: Intern wird AD=0 genauso verarbeitet wie BY VALUE (siehe Anmerkung unter <i>operand2</i>).
	AD=M	Modifizierbar, siehe Session-Parameter AD=M. Dies ist die Standardeinstellung.
	AD=A	Nur für Eingabe, siehe Session-Parameter AD=A.
Wenn <i>operand2</i> eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=0.		
nX	Angabe zu überspringender Parameter:	
Mit der Notation <i>nX</i> können Sie angeben, dass die nächsten <i>n</i> Parameter übersprungen werden sollen (zum Beispiel 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten 3 Parameter zu überspringen); dies bedeutet, dass für die nächsten <i>n</i> Parameter keine Werte an die externe Subroutine übergeben werden.		
Ein zu überspringender Parameter muss mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER -Statement der Subroutine definiert werden. OPTIONAL bedeutet, dass ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann, aber nicht muss.		

Beispiele

- Beispiel 1 — PERFORM als interne Subroutine
- Beispiel 2 — PERFORM als externe Subroutine

Beispiel 1 — PERFORM als interne Subroutine

```

** Example 'PEREX1': PERFORM (as inline subroutine)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
*
1 #ARRAY (A75/1:4)

```

PERFORM

```
1 REDEFINE #ARRAY
  2 #ALINE (A25/1:4,1:3)
1 #X      (N2) INIT <1>
1 #Y      (N2) INIT <1>
END-DEFINE
*
LIMIT 5
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
  MOVE NAME          TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE         TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    /*
    PERFORM PRINT
    /*
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
  /*
  PERFORM PRINT
  /*
  END-ENDDATA
END-FIND
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE
*
END
```

Ausgabe des Programms PEREX1:

```
JENSON          LAWLER          FORREST
2120 HASSELL    4588 CANDLEBERRY AVE  37 TENNYSON DRIVE
  #206          BALTIMORE          BALTIMORE
998-5038        629-0403          881-3609

ALEXANDER       NEEDHAM
409 SENECA DRIVE 12609 BUILDERS LANE
BALTIMORE       BALTIMORE
345-3690        641-9789
```

Beispiel 2 — PERFORM als externe Subroutine

Programm, das das PERFORM-Statement enthält:

```

** Example 'PEREX2': PERFORM (as external subroutine)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (A20/2)
  2 PHONE
*
1 #ALINE (A25/1:4,1:3)
1 #X (N2) INIT <1>
1 #Y (N2) INIT <1>
END-DEFINE
*
LIMIT 5
*
FIND EMPLOY-VIEW WITH CITY = 'BALTIMORE'
  MOVE NAME TO #ALINE (#X,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
  MOVE PHONE TO #ALINE (#X+3,#Y)
  IF #Y = 3
    RESET INITIAL #Y
    /*
    PERFORM PEREX2E #ALINE(*,*)
    /*
  ELSE
    ADD 1 TO #Y
  END-IF
  AT END OF DATA
  /*
  PERFORM PEREX2E #ALINE(*,*)
  /*
  END-ENDDATA
END-FIND
*
END

```

Externe Subroutine PEREX3 mit vom Programm PEREX2 aufgerufenen Parametern:

```
** Example 'PEREX3': SUBROUTINE (external subroutine with parameters)
*****
DEFINE DATA
PARAMETER
1 #ALINE (A25/1:4,1:3)
END-DEFINE
*
DEFINE SUBROUTINE PEREX2
WRITE NOTITLE (AD=OI) #ALINE(*,*)
RESET #ALINE(*,*)
SKIP 1
END-SUBROUTINE
*
END
```

Ausgabe des Programms PEREX2:

```
JENSON                LAWLER                FORREST
2120 HASSELL          4588 CANDLEBERRY AVE 37 TENNYSON DRIVE
#206                  BALTIMORE             BALTIMORE
998-5038              629-0403             881-3609

ALEXANDER             NEEDHAM
409 SENECA DRIVE     12609 BUILDERS LANE
BALTIMORE            BALTIMORE
345-3690             641-9789
```

94 PERFORM BREAK PROCESSING

- Funktion 604
- Syntax-Beschreibung 604
- Beispiel 605

```
PERFORM BREAK [PROCESSING] [(r)]
  AT BREAK statement ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement PERFORM BREAK PROCESSING dient dazu, bei Verarbeitungsschleifen, die mit FOR, REPEAT, CALL LOOP oder CALL FILE ausgelöst wurden, dort eine Gruppenwechsel-Verarbeitung auszulösen, wo keine automatische Gruppenwechsel-Verarbeitung durchgeführt wird, oder wenn eine Gruppenwechsel-Verarbeitung gewünscht wird.

Im Gegensatz zu einer automatischen Gruppenwechsel-Verarbeitung, die ausgeführt wird, unmittelbar nachdem der Datensatz gelesen wurde, wird ein PERFORM BREAK PROCESSING-Statement dann ausgeführt, wenn es im normalen Programmablauf auftaucht.

Das PERFORM BREAK PROCESSING-Statement überprüft anhand des Wertes eines Kontrollfeldes, ob eine Gruppenwechsel-Bedingung erfüllt wird, und bewirkt außerdem eine Auswertung der Natural-Systemfunktionen. Diese Prüfung und Auswertung findet jedesmal, wenn das Statement ausgeführt wird, statt. Die Ausführung eines PERFORM BREAK PROCESSING-Statements kann an eine mit einem IF-Statement angegebene logische Bedingung geknüpft werden.

Syntax-Beschreibung

(r)	<p>Statement-Referenzierung:</p> <p>Normalerweise wird die PERFORM BREAK PROCESSING-Verarbeitung zum letztenmal ausgeführt, wenn die Ausführung des Programms/Subprogramms bzw. der Subroutine beendet ist.</p> <p>Durch Verwendung eines Statement-Labels oder Angabe der Sourcecode-Zeilenummer mittels Notation (r) kann eine bestimmte Verarbeitungsschleife referenziert werden, auf die sich die abschließende PERFORM BREAK PROCESSING-Verarbeitung beziehen soll; in diesem Falle ist sie Teil der schleifenbeendenden Verarbeitung, d.h. die letzte PERFORM</p>
-----	---

	BREAK-Verarbeitung wird nach der letzten automatischen Gruppenwechsel-Verarbeitung und vor den AT END OF DATA -Statements ausgeführt.
AT BREAK <i>statement...</i>	Siehe Syntax des AT BREAK -Statements.

Beispiel

```

** Example 'PBPEX1S': PERFORM BREAK PROCESSING (structured mode)
*****
DEFINE DATA LOCAL
1 #INDEX (N2)
1 #LINE (N2) INIT <1>
END-DEFINE
*
FOR #INDEX 1 TO 18
  PERFORM BREAK PROCESSING
  /*
  AT BREAK OF #INDEX /1/
    WRITE NOTITLE / 'PLEASE COMPLETE LINES 1-9 ABOVE' /
    RESET INITIAL #LINE
  END-BREAK
  /*
  WRITE NOTITLE '_' (64) '=' #LINE
  ADD 1 TO #LINE
END-FOR
*
END

```

Ausgabe des Programms PBPEX1S:

```

_____ #LINE: 1
_____ #LINE: 2
_____ #LINE: 3
_____ #LINE: 4
_____ #LINE: 5
_____ #LINE: 6
_____ #LINE: 7
_____ #LINE: 8
_____ #LINE: 9

PLEASE COMPLETE LINES 1-9 ABOVE

_____ #LINE: 1
_____ #LINE: 2
_____ #LINE: 3
_____ #LINE: 4

```

PERFORM BREAK PROCESSING

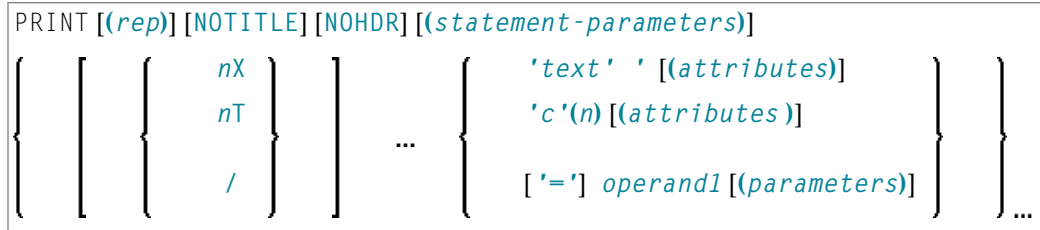
	#LINE :	5
	#LINE :	6
	#LINE :	7
	#LINE :	8
	#LINE :	9

PLEASE COMPLETE LINES 1-9 ABOVE

Äquivalentes Reporting-Mode-Beispiel: [PBPEX1R](#).

95 PRINT

▪ Funktion	608
▪ Syntax-Beschreibung	609
▪ Beispiel	614



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [AT END OF PAGE](#) | [AT TOP OF PAGE](#) | [CLOSE PRINTER](#) | [DEFINE PRINTER](#) | [DISPLAY](#) | [EJECT](#) | [FORMAT](#) | [NEWPAGE](#) | [SKIP](#) | [SUSPEND IDENTICAL SUPPRESS](#) | [WRITE](#) | [WRITE TITLE](#) | [WRITE TRAILER](#)

Gehört zur Funktionsgruppe: [Erstellen von Ausgabe-Reports](#)

Funktion

Das Statement `PRINT` dient dazu, Ausgaben im freien Format zu erzeugen.

Das `PRINT`-Statement unterscheidet sich vom `WRITE`-Statement in folgenden Punkten:

- Die Ausgabelänge der einzelnen Operanden ergibt sich aus der Länge der tatsächlich ausgegebenen Werte und nicht aus der Länge der verwendeten Felder. Vorangestellte Nullen (bei numerischen Werten) und nachgestellte Leerzeichen (bei alphanumerischen Werten) werden nicht mit ausgegeben.

Mit dem Session-Parameter `AD` können Sie festlegen, ob numerische Werte links- oder rechtsbündig ausgegeben werden sollen: mit `AD=L` werden einem numerischen Wert nachfolgende Leerstellen nicht ausgegeben; mit `AD=R` werden einem numerischen Wert vorangestellte Leerzeichen mit ausgegeben.

- Überschreitet die Ausgabe die vorgegebene Zeilenlänge (Parameter `LS`), wird die Ausgabe in der nächsten Zeile wie folgt fortgesetzt:

Eine alphanumerische Konstante oder der Inhalt einer alphanumerischen Variablen (ohne Editiermaske) wird ab dem letzten auf der aktuellen Zeile ausgegebenen Leerzeichen oder Zeichen, das weder ein Buchstabe noch eine Ziffer ist, abgetrennt. Der erste Teil des Wertes verbleibt auf der aktuellen Zeile, der abgetrennte Teil wird in der nächsten Zeile ausgegeben. Führende Leerzeichen im zweiten Teil werden entfernt und Leerzeilen werden dadurch unterdrückt.

Bei allen anderen Operanden wird der gesamte Wert, der nicht mehr in die aktuelle Zeile passt, in der nächsten Zeile ausgegeben.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A U N P I F B D T L G O	ja	nein

Syntax-Element-Beschreibung:

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls nichts anderes angegeben wird, bezieht sich das PRINT-Statement auf den ersten Report (Report 0).</p> <p>Wenn diese Druckdatei für Natural als PC definiert wird, wird der Report auf den PC heruntergeladen, siehe Beispiel 2. Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, siehe <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
NOTITLE	<p>Unterdrückung der Standard-Seitenüberschrift:</p> <p>Für jede über ein PRINT-Statement ausgegebene Seite generiert Natural eine Titelzeile, die die laufende Seitennummer, die Uhrzeit und das Datum enthält. Die Uhrzeit wird zu Beginn der Session (TP-Betrieb) oder zu Beginn des Jobs (Batch-Betrieb) gesetzt. Die generierte Titelzeile kann entweder durch eine eigene mit einem WRITE TITLE-Statement angegebene Titelzeile überschrieben oder durch eine NOTITLE-Klausel im PRINT-Statement unterdrückt werden.</p> <p>Beispiele:</p>

	<ul style="list-style-type: none"> ■ Generierte Titelzeile wird ausgegeben: <pre>PRINT NAME</pre> ■ Eigene Titelzeile wird ausgegeben: <pre>PRINT NAME WRITE TITLE 'user-title'</pre> ■ Keine Titelzeile wird ausgegeben: <pre>PRINT NOTITLE NAME</pre> <p>Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.</p>
<p>NOHDR</p>	<p>Unterdrückung der Spaltenüberschrift:</p> <p>Das PRINT-Statement selbst erzeugt keine Spaltenüberschriften. Wenn Sie allerdings das PRINT-Statement zusammen mit einem DISPLAY-Statement verwenden, können Sie mit der Option NOHDR des PRINT-Statements die vom DISPLAY-Statement generierten Spaltenüberschriften unterdrücken:</p> <p>Die NOHDR-Option ist nur relevant, wenn das PRINT-Statement nach einem DISPLAY-Statement steht, die Ausgabe sich insgesamt über mehr als eine Seite erstreckt und die Ausführung des PRINT-Statements zur Ausgabe einer neuen Seite führt.</p> <p>Ohne NOHDR-Option würden auf dieser neuen Seite die DISPLAY-Spaltenüberschriften ausgegeben, mit NOHDR werden sie dort nicht ausgegeben.</p>
<p><i>statement-parameters</i></p>	<p>Parameter-Definition auf Statement-Ebene:</p> <p>Unmittelbar nach dem Schlüsselwort PRINT selbst oder nach einem der auszugebenden Felder können auf Statement-Ebene in Klammern Session-Parameter gesetzt werden.</p> <p>Diese Parameter haben dann für das jeweilige Statement oder Feld Gültigkeit statt der betreffenden mit einem GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT-Statement gesetzten Parameter. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht, haben aber keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen Sie explizit für dieses Element gesetzt werden, siehe Parameter-Definition auf Element-Ebene.</p>

	Siehe auch: <ul style="list-style-type: none"> ■ Liste der Parameter ■ Beispiel für Parameter-Benutzung auf Statement- und Element-Ebene.
<i>nX, nT, l</i>	Siehe Feldpositionierung , Text , Attributzuweisung weiter unten.

Liste der Parameter

Parameter, die beim PRINT-Statement angegeben werden können:		Spezifikation:
		S = auf Statement-Ebene
		E =auf Element-Ebene
AD	Attribute Definition	SE
AL	Alphanumeric Length for Output	SE
CD	Color Definition	SE
CV	Control Variable	SE
DF	Date Format	SE
DL	Display Length for Output	SE
DY	Dynamic Attributes	SE
EM	Edit Mask	SE
EMU	Unicode Edit Mask	E
FL	Floating Point Mantissa Length	SE
MC	Multiple-Value Field Count	S
MP	Maximum Number of Pages of a Report	S
NL	Numeric Length for Output	SE
PC	Periodic Group Count	S
PM	Print Mode	SE
SG	Sign Position	SE
ZP	Zero Printing	SE

Beschreibungen der einzelnen Parameter entnehmen Sie bitte der *Parameter-Referenz*.

Beispiel für Parameter-Benutzung auf Statement- und Element-Ebene

Feldpositionierung, Text, Attributzuweisung

```
{ { nX } { 'text' [(attributes)] } }
{ { nT } { 'c' (n) [(attributes)] } }
{ { / } { '=' operand1 [(parameters)] } } ...
```

Feldpositionierungsnotationen

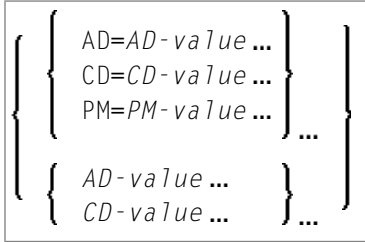
<i>nX</i>	<p>Spaltenabstand:</p> <p>Mit dieser Notation können Sie zwischen den auszugebenden Werten <i>n</i> Leerstellen einfügen. Beispiel:</p> <pre>PRINT NAME 5X SALARY</pre>
<i>nT</i>	<p>Setzen von Tabulatoren:</p> <p>Mit dieser Notation können Sie Tabulatoren setzen, d.h. die Ausgabe eines Wertes beginnt ab Spalte <i>n</i>. Wird ein Tabulator gesetzt, dessen Position bereits durch einen anderen ausgegeben Wert besetzt ist, erfolgt ein Zeilenvorschub.</p> <p>Im folgenden Beispiel wird NAME ab Spalte 25 ausgegeben und SALARY ab Spalte 50:</p> <pre>PRINT 25T NAME 50T SALARY</pre>
<i>/</i>	<p>Zeilenvorschub – Schrägstrich-Notation:</p> <p>Mit einem Schrägstrich (/) bewirken Sie zwischen zwei Feldern oder Textelementen einen Zeilenvorschub. Beispiel:</p> <pre>PRINT NAME / SALARY</pre>

Text-/Attributzuweisung

'text'	<p>Zuweisung von Text:</p> <p>Eine in Apostrophen angegebene Zeichenkette 'text' wird als Text ausgegeben. Beispiel:</p> <pre>PRINT 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT</pre>
'c' (n)	<p>Wiederholung von Zeichen:</p> <p>Wie 'text'. Ausnahme: das Zeichen c wird n-mal unmittelbar vor dem Feldwert ausgegeben. Beispiel:</p> <pre>PRINT '*' (5) '=' NAME</pre>
'='	<p>Position des Feldinhalts hinter Feldüberschrift:</p> <p>Ein Gleichheitszeichen in Apostrophen unmittelbar vor einem Feld bewirkt, dass unmittelbar vor dem Feldwert der Name des Feldes ausgegeben wird (wie im DEFINE DATA-Statement oder im DDM definiert). Beispiel:</p> <pre>PRINT '=' NAME</pre>
operand1	<p>Auszugebendes Feld:</p> <p>Als <i>operand1</i> geben Sie das auszugebende Feld an.</p>
parameters	<p>Parameter-Definition auf Elementebene (Feldebene):</p> <p>Unmittelbar nach <i>operand1</i> können Sie in Klammern einen oder mehrere Parameter (siehe obige Tabelle) angeben. Diese Parameter haben dann für das jeweilige Feld Gültigkeit statt der betreffenden, auf Statement-Ebene mit einem <code>GLOBALS</code>-Kommando, <code>SET GLOBALS</code>- (nur im Reporting Mode) oder <code>FORMAT</code>-Statement gesetzten Parameter.</p> <p>Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Statement-Parameter ■ Beispiel der Parameter-Benutzung auf Statement- und Element-Ebene

Ausgabeattribute

attributes dient dazu, den ausgegebenen Feldern/Textelementen Anzeige- und Farbattribute zuzuordnen. Sie können die folgenden Attribute angeben:



Die möglichen Parameterwerte sind in den folgenden Abschnitten der *Parameter-Referenz* aufgeführt:

- *AD - Attribute Definition, Abschnitt Feldanzeige*
- *CD - Color Definition*
- *PM - Print Mode*



Anmerkung: Der Compiler akzeptiert mehr als einen Attributwert für ein Ausgabefeld. Beispielsweise können Sie angeben: *AD=BDI*. In einem solchen Fall gilt allerdings nur der letzte Wert. In dem vorliegenden Beispiel greift nur der Wert *I*, und das Ausgabefeld wird intensiviert dargestellt.

Beispiel

- [Beispiel 1 — PRINT-Statement](#)
- [Beispiel 2 — PRINT-Statement mit auf den PC herunterzuladendem Report](#)

Beispiel 1 — PRINT-Statement

```

** Example 'PRTEX1': PRINT
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
  2 ADDRESS-LINE (2)
END-DEFINE
*
LIMIT 1
READ EMPLOY-VIEW BY CITY
    
```

```

/*
WRITE NOTITLE 'EXAMPLE 1:'
           // 'RESULT OF WRITE STATEMENT:'
WRITE      /  NAME  ',' FIRST-NAME ':' JOB-TITLE '*' (30)
WRITE      /  'RESULT OF PRINT STATEMENT:'
PRINT      /  NAME  ',' FIRST-NAME ':' JOB-TITLE '*' (30)
/*
WRITE      // 'EXAMPLE 2:'
           // 'RESULT OF WRITE STATEMENT:'
WRITE      /  NAME 60X ADDRESS-LINE (1:2)
WRITE      /  'RESULT OF PRINT STATEMENT:'
PRINT      /  NAME 60X ADDRESS-LINE (1:2)
/*
END-READ
END

```

Ausgabe des Programms PRTXEX1:

EXAMPLE 1:

RESULT OF WRITE STATEMENT:

```

SENKO           , WILLIE           : PROGRAMMER
*****

```

RESULT OF PRINT STATEMENT:

```

SENKO , WILLIE : PROGRAMMER *****

```

EXAMPLE 2:

RESULT OF WRITE STATEMENT:

```

SENKO
2200 COLUMBIA PIKE   #914

```

RESULT OF PRINT STATEMENT:

```

SENKO                                     2200 COLUMBIA
PIKE #914

```

Beispiel 2 — PRINT-Statement mit auf den PC herunterzuladendem Report

```
** Example 'PCPIEX1': PRINT to PC
**
** NOTE: Example requires that Natural Connection is installed.
*****
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'           /* Data selection
  PRINT (7) 5T CITY 20T NAME 40T PERSONNEL-ID /* (7) designates
                                           /* the output file
                                           /* (here the PC).
END-FIND
END
```

96 PROCESS

▪ Funktion	618
▪ Einschränkung	618
▪ Syntax-Beschreibung	618

```
PROCESS view-name USING operand1=operand2[,operand1=operand2]... [GIVING operand3...]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das Statement PROCESS wird in Verbindung mit Entire System Server eingesetzt. Mit Entire System Server können Sie auf verschiedene Funktionen des Betriebssystems zugreifen, zum Beispiel: Lesen/Beschreiben von Dateien, VTOC/Catalog-Management, JES-Queues usw.

Nähere Informationen zum PROCESS-Statement und seinen Klauseln finden Sie unter *Getting Started* im *Entire System Server User's Guide*.

Einschränkung

Dieses Statement steht nur für Entire System Server zur Verfügung.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate										Referenzierung erlaubt	Dynam. Definition					
<i>operand1</i>	C	S				A		N	P			B						ja	nein
<i>operand2</i>	C	S				A	U	N	P			B						ja	nein
<i>operand3</i>		S				A		N	P			B						ja	nein

Syntax-Element-Beschreibung:

<i>view-name</i>	Name des von Entire System Server benutzten Views.
USING	<p>Mit dieser Klausel können Parameter an den Entire System Server-Prozessor übergeben werden, indem einem Feld (<i>operand1</i>) eines unter Entire System Server definierten Views ein Wert (<i>operand2</i>) zugewiesen wird. View-Beschreibung siehe <i>Entire System Server-Dokumentation</i>.</p> <p>Anmerkung: Mehrfache Angaben von <i>operand1=operand2</i> müssen entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander getrennt werden. Ein Komma darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (mit dem Session-Parameter DC) definiert ist.</p>
GIVING	Mit der GIVING-Klausel können Sie Felder (<i>operand3</i>) angeben, an die vom Entire System Server-Prozessor Werte zurückgegeben werden. Jedes dieser Felder muss in einem von Entire System Server benutzten View definiert sein.

97 PROCESS COMMAND

▪ Funktion	623
▪ Syntax-Beschreibung	623
▪ Das DDM COMMAND	635
▪ Beispiele	636

Structured Mode-Syntax

PROCESS	COMMAND	ACTION	
}	{	CLOSE	
		CHECK	
		EXEC	PROCESSOR-NAME= <i>operand1</i>
		TEXT	COMMAND-LINE (<i>index[:index]</i>)= <i>operand2</i>
	HELP		
	GET USING	PROCESSOR-NAME= <i>operand1</i>	
	SET USING	GETSET-FIELD-NAME= <i>operand3</i>	
		PROCESSOR-NAME= <i>operand1</i>	
		GETSET-FIELD-NAME= <i>operand3</i>	
			GETSET-FIELD-VALUE= <i>operand4</i>

Reporting Mode-Syntax

PROCESS	COMMAND	ACTION	
}	{	CLOSE [GIVING NATURAL-ERROR]	
		CHECK	
		EXEC	PROCESSOR-NAME= <i>operand1</i>
		TEXT	COMMAND-LINE (<i>index[:index]</i>)= <i>operand2</i>
	HELP	GIVING RESULT-FIELD (<i>index[:index]</i>)	
	GET USING	PROCESSOR-NAME= <i>operand1</i>	
	SET USING	GETSET-FIELD-NAME= <i>operand3</i>	
		GETSET-FIELD-VALUE= <i>operand4</i> [NATURAL-ERROR]	
		PROCESSOR-NAME= <i>operand1</i>	
			GETSET-FIELD-NAME= <i>operand3</i>
		GETSET-FIELD-VALUE= <i>operand4</i> [GIVING NATURAL-ERROR]	

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Sobald ein Kommando-Prozessor mit der Natural-Utility `SYSNCP` erstellt worden ist, kann er von einem Natural-Programm mit dem Statement `PROCESS COMMAND` aufgerufen werden.

Näheres zur Erstellung eines Natural-Kommando-Prozessors finden Sie in der *SYSNCP Utility*-Dokumentation.



Anmerkung: Das Wort `COMMAND` im Statement `PROCESS COMMAND` ist eigentlich der Name eines Views. Der Name des verwendeten Views muss nicht unbedingt `COMMAND` sein; aber wir empfehlen die Verwendung von `COMMAND`, da ein **DDM dieses Namens** existiert. Dieses DDM muss im `DEFINE DATA`-Statement referenziert werden, zum Beispiel: `COMMAND VIEW OF COMMAND`.

Security-Hinweise

Mit Natural Security können Sie die Verwendung bestimmter in einem Kommando-Prozessor definierter Schlüsselwörter und/oder Funktionen einschränken. Schlüsselwörter und/oder Funktionen können für jeden einzelnen Benutzer (oder Gruppen von Benutzern) erlaubt bzw. verboten werden.

Weitere Informationen siehe *Natural Security*-Dokumentation.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S			A													nein	nein
<i>operand2</i>	C	S	A	G	A	N												nein	nein
<i>operand3</i>	C	S			A	N												nein	nein
<i>operand4</i>	C	S			A	N	P	I										nein	nein

Syntax-Element-Beschreibung:

CLOSE	<p>CLOSE beendet den Einsatz des Kommando-Prozessors und gibt den Kommando-Prozessor-Puffer wieder frei.</p> <p>Wenn der Kommando-Prozessor während einer Session benutzt und nicht mit CLOSE freigegeben wird, enthält Ihr Thread einen Puffer namens NCPWORK. Der Puffer wird vom Laufzeit-Teil des Kommando-Prozessors benötigt; er kann mit dem Statement <code>PROCESS COMMAND ACTION CLOSE</code> wieder freigegeben werden.</p> <p>Wenn auf dieses Statement ein anderes <code>PROCESS COMMAND</code>-Statement folgt, wird der Kommando-Prozessor-Puffer erneut geöffnet.</p> <p>Siehe auch Beispiel 1 – PROCESS COMMAND ACTION CLOSE.</p>
CHECK	<p>CHECK dient als Vorsichtsmaßnahme, um herauszufinden, ob ein Kommando mit dem Statement <code>PROCESS COMMAND EXEC</code> ausführbar ist. Für einen bestimmten Prozessor-Namen erfolgt zur Laufzeit eine Prüfung in zwei Schritten:</p> <ul style="list-style-type: none"> ■ Es wird geprüft, ob der Prozessor in der aktuellen Library oder einer ihrer Steplibs vorhanden ist. ■ Es wird geprüft, ob der Inhalt der Kommandozeile <code>COMMAND-LINE (1)</code> akzeptabel ist. <p>Außerdem werden die Laufzeit-Aktionen R, M und 1-9 in <code>RESULT-FIELD (1:9)</code> geschrieben.</p> <p>Wenn Sie das Feld <code>NATURAL-ERROR</code> im View oder in der <code>GIVING</code>-Klausel angeben, wird der Fehlercode in diesem Feld ausgegeben. Wird dieses Feld nicht angegeben und die Kommandoanalyse stößt auf einen Fehler, dann erzeugt Natural einen Systemfehler.</p> <p>Anmerkung: Da die Funktion der CHECK-Option auch als Teil der EXEC-Option ausgeführt wird (siehe unten), ist es nicht nötig, CHECK vor jeder EXEC-Option zu verwenden.</p>
EXEC	<p>EXEC funktioniert genau wie CHECK und bewirkt zusätzlich, dass die mit dem Runtime Action Editor angegebenen Laufzeit-Aktionen ausgeführt werden.</p> <p>Es wird nur <code>COMMAND-LINE (1)</code> benötigt. Sie können bis zu 9 Ausprägungen von <code>RESULT-FIELD</code> verwenden (im Hinblick auf optimale Verarbeitungszeit sollten Sie jedoch nur so viele Ausprägungen verwenden wie Sie tatsächlich benötigen).</p> <p>Anmerkung: EXEC ist die einzige Option, mit der das gerade aktive Programm verlassen werden kann. Dies ist der Fall, wenn die Laufzeit-Aktion ein <code>FETCH</code>- oder <code>STOP</code>-Statement enthält.</p> <p>Siehe auch Beispiel 2 – PROCESS COMMAND ACTION EXEC.</p>
HELP	<p>Mit HELP erhalten Sie eine Liste aller gültigen Schlüsselwörter, Synonyme und Funktionen, um beispielsweise Online-Hilfe-Fenster zu erzeugen. Die Liste ist in dem Feld bzw. den Feldern von <code>RESULT-FIELD</code> enthalten. Die Art der erhaltenen Hilfe hängt vom Inhalt der Kommandozeilen ab:</p> <ul style="list-style-type: none"> ■ <code>COMMAND-LINE (1)</code> muss die Suchkriterien enthalten. ■ <code>COMMAND-LINE (2)</code> (falls angegeben) muss den Startwert oder einen Suchwert enthalten. ■ <code>COMMAND-LINE (3)</code> (falls angegeben) muss einen Startwert enthalten.

	<p>Weitere Informationen siehe die folgenden Abschnitte:</p> <ul style="list-style-type: none"> ■ <i>HELP für Schlüsselwörter</i> ■ <i>HELP für Synonyme</i> ■ <i>HELP für globale Funktionen</i> ■ <i>HELP für lokale Funktionen</i> ■ <i>HELP für IKN</i> ■ <i>HELP für IFN</i> <p>Anmerkung: Im Hinblick auf optimale Verarbeitungszeit sollte das Feld RESULT - FIELD nicht mehr Ausprägungen haben als Zeilen auf dem Schirm angezeigt werden sollen. Mindestens eine Ausprägung ist erforderlich.</p>
TEXT	<p>Mit der Option TEXT erhalten Sie allgemeine Informationen über den Prozessor sowie Text zu einem Schlüsselwort bzw. einer Funktion. Der Text ist derselbe, der bei der Definition eines Kommando-Prozessors mit der SYSNCP Utility im Keyword Editor oder Action Editor eingegeben wurde.</p> <p>Weitere Informationen siehe die folgenden Abschnitte:</p> <ul style="list-style-type: none"> ■ <i>TEXT für allgemeine Informationen</i> ■ <i>TEXT für Schlüsselwort-Informationen</i> ■ <i>TEXT für Funktions-Informationen</i> <p>Anmerkung: Um auf Text zu Schlüsselwörtern oder Funktionen zugreifen zu können, muss im Feld <code>Catalog user texts</code> auf dem Schirm <i>Processor Header Maintenance 3</i> der SYSNCP-Utility ein Y (ja) eingetragen sein, siehe Abschnitt <i>Miscellaneous Options - Header 3</i>.</p>

HELP für Schlüsselwörter

Diese Option liefert eine alphabetisch sortierte Liste von Schlüsselwörtern bzw. Synonymen und ihren IKNs (IKN = Internal Keyword Number = Interne Schlüsselwort-Nummer)

Kommandozeile	Inhalt
1	Muss mit Indikator K (für <i>Keyword</i>) anfangen.
	Die Typen der gewünschten Schlüsselwörter:
*	Schlüsselwörter aller Typen
1	Schlüsselwörter vom Typ 1
2	Schlüsselwörter vom Typ 2
3	Schlüsselwörter vom Typ 3
P	Schlüsselwörter vom Typ P (Parameter)
	Optionen:
I	Gibt zusätzlich zu Schlüsselwörtern die IKN zurück.

Kommandozeile	Inhalt	
	T	Zeigt das Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
	S	Gibt zusätzlich zu Schlüsselwörtern Synonyme zurück.
	X	Gibt nur Synonyme der angegebenen Schlüsselwörter zurück.
	A	Interne Schlüsselwörter werden auch zurückgegeben.
	+	Suche schließt Startwert nicht mit ein.
2	Startwert für Schlüsselwort-Suche (optional). Standardmäßig beginnt die Suche ab dem Startwert. Wenn Sie jedoch Option + angeben, schließt die Suche den Startwert selbst nicht mit ein, sondern beginnt ab dem nächsthöheren Wert.	

Im Feld **RESULT-FIELD (1:n)** erhalten Sie die angegebene Liste.

Beispiel:

```
Command Line 1: K*X
```

Gibt alle Synonyme aller Schlüsselworttypen zurück.

```
Command Line 1: K123S
```

Gibt alle Schlüsselwörter vom Typ 1, 2 und 3 einschließlich ihrer Synonyme zurück.

HELP für Synonyme

Für eine bestimmte IKN (Internal Keyword Number = Interne Schlüsselwort-Nummer) liefert diese Option das ursprüngliche Schlüsselwort sowie alle Synonyme.

Kommandozeile	Inhalt	
1	Muss mit Indikator S anfangen.	
	Option:	
	T	Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
2	IKN des Schlüsselworts im Format N4.	

Im Feld **RESULT-FIELD (1)** erhalten Sie das Schlüsselwort selbst. In den Feldern **RESULT-FIELD (2:n)** erhalten Sie die Synonyme des Schlüsselworts.

Beispiel:

Eingabe:		Ausgabe:	
Command Line 1:	S	Result-Field 1:	Edit
Command Line 2:	1003	Result-Field 2:	Maintain
		Result-Field 3:	Modify

HELP für globale Funktionen

Diese Option liefert eine Liste aller globalen Funktionen.

Kommandozeile	Inhalt	
1	Muss mit Indikator G anfangen.	
	Optionen:	
	I	Die Interne Funktionsnummer (IFN) wird auch zurückgegeben.
	T	Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
	S	Die Schlüsselwörter werden in RESULT-FIELD in Spaltenform ausgegeben.
	A	Interne Schlüsselwörter werden auch zurückgegeben.
	1	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 1 enthalten, werden zurückgegeben.
	2	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 2 enthalten, werden zurückgegeben.
	3	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 3 enthalten, werden zurückgegeben.
+	Suche schließt Startwert nicht mit ein.	
2	Startwert für Suche nach globalen Funktionen. Schlüsselwörter müssen in der Reihenfolge 123 angegeben werden. Standardmäßig beginnt die Suche ab dem Startwert. Wenn Sie jedoch Option + angeben, schließt die Suche den Startwert selbst nicht mit ein, sondern beginnt ab dem nächsthöheren Wert.	
3	Muss leer sein.	
4	Wenn Sie nur nach globalen Funktionen eines bestimmten Schlüsselworts suchen möchten, geben Sie hier das betreffende Schlüsselwort an. Gleichzeitig müssen Sie den Schlüsselwort-Typ (1, 2 oder 3) als Option (siehe oben) angeben.	

Im Feld **RESULT-FIELD (1:n)** erhalten Sie die angegebene Liste.

Beispiel:

Eingabe:		Ausgabe:	
Command Line 1:	G	Result-Field 1:	ADD CUSTOMER
Command Line 2:	ADD	Result-Field 2:	ADD FILE
		Result-Field 3:	ADD USER

HELP für lokale Funktionen

Diese Option liefert eine Liste aller lokalen Funktionen für einen bestimmten Platz.

Kommandozeile	Inhalt	
1	Muss mit Indikator L anfangen.	
	Optionen:	
	I	Die Interne Funktionsnummer (IFN) wird auch zurückgegeben.
	T	Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).
	S	Die Schlüsselwörter werden in RESULT-FIELD in Spaltenform ausgegeben.
	A	Interne Schlüsselwörter werden auch zurückgegeben.
	1	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 1 enthalten, werden zurückgegeben.
	2	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 2 enthalten, werden zurückgegeben.
	3	Nur Funktionen, die das angegebene Schlüsselwort vom Typ 3 enthalten, werden zurückgegeben.
	C	Nur Funktionen, die für den aktuellen Platz definiert sind, werden zurückgegeben (Kommandozeile 3 wird ignoriert).
F	Ruft rekursive Liste lokaler Funktionen auf, d.h. alle lokalen Kommandos, die zum aktuellen/angegebenen Platz führen, werden zurückgegeben.	
2	Startwert für Suche nach lokalen Funktionen (optional).	
	Schlüsselwörter müssen in der Reihenfolge 123 angegeben werden.	
3	Der Platz, für den die Liste gewünscht wird.	
	Schlüsselwörter müssen in der Reihenfolge 123 angegeben werden.	
	Wenn kein Platz angegeben wird, wird der aktuelle Platz des Kommando-Prozessors genommen.	

Kommandozeile	Inhalt
4	Schlüsselwort-Einschränkung (optional): Wenn Sie ein Schlüsselwort oder eine IKN mit Format N4 angeben, werden nur Funktionen mit diesem Schlüsselwort zurückgegeben.

Im Feld **RESULT-FIELD (1:n)** erhalten Sie die angegebene Liste.

HELP für IKN

Für eine bestimmte Interne Schlüsselwortnummer (IKN) liefert diese Option das ursprüngliche Schlüsselwort.

Kommandozeile	Inhalt			
1	Muss mit IKN anfangen.			
	Optionen:			
	<table border="1"> <tr> <td>A</td> <td>Das interne Schlüsselwort wird gezeigt.</td> </tr> <tr> <td>T</td> <td>Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).</td> </tr> </table>	A	Das interne Schlüsselwort wird gezeigt.	T
A	Das interne Schlüsselwort wird gezeigt.			
T	Zeigt Schlüsselwort teilweise in Großbuchstaben (um mögliche Abkürzung zu zeigen).			
2	Die zu übersetzende IKN im Format N4.			

Im Feld **RESULT-FIELD (1)** erhalten Sie das Schlüsselwort.

Beispiel:

Eingabe:	Ausgabe:
Command Line 1: IKN	Result-Field 1: CUSTOMER
Command Line 2: 0000002002	

HELP für IFN

Für eine bestimmte Interne Funktionsnummer (IFN) liefert diese Option die Schlüsselwörter einer Funktion.

Kommandozeile	Inhalt		
1	Muss mit IFN anfangen.		
	Option:		
	<table border="1"> <tr> <td>A</td> <td>Funktionen mit internen Schlüsselwörtern werden nicht unterdrückt.</td> </tr> </table>	A	Funktionen mit internen Schlüsselwörtern werden nicht unterdrückt.
A	Funktionen mit internen Schlüsselwörtern werden nicht unterdrückt.		
2	Die zu übersetzende IFN im Format N10.		
3	Weitere Optionen:		

Kommandozeile	Inhalt	
	S	Gibt die zu der IFN gehörenden Schlüsselwörter in RESULT-FIELD (1:3) zurück.
	T	Zeigt Schlüsselwörter teilweise in Großbuchstaben (um mögliche Abkürzungen zu zeigen).
	L	Die IFN wird zurückgegeben, wenn die IFN als Platz verwendet wird.
	C	Die IFN wird zurückgegeben, wenn die IFN als Kommando verwendet wird.

Im Feld **RESULT-FIELD(1)** erhalten Sie die Funktion. Wenn Sie Option S verwenden, erhalten Sie die Funktion in **RESULT-FIELD (1:3)**.

Beispiel:

Eingabe:	Ausgabe:
Command Line 1: IFN Command Line 2: 0001048578	Result-Field 1: DISPLAY INVOICE

TEXT für allgemeine Informationen

Bei allgemeinen Informationen muss **COMMAND-LINE (*)**, d.h. alle Kommandozeilen, leer sein. In den bis zu 9 Feldern von **RESULT-FIELD** erhalten Sie folgende Informationen:

RESULT-FIELD	Inhalt	Format
1	Header 1 for User Text (Kopfzeile 1 für Benutzertext)	Text (A40)
2	Header 2 for User Text (Kopfzeile 2 für Benutzertext)	Text (A40)
3	„First Entry used as“ text (Erster Eintrag benutzt als)	Text (A16)
4	„Second Entry used as“ text (Zweiter Eintrag benutzt als)	Text (A16)
5	„Third Entry used as“ text (Dritter Eintrag benutzt als)	Text (A16)
6	Anzahl der Eintrag-1-Schlüsselwörter.	Numerisch (N3)
7	Anzahl der Eintrag-2-Schlüsselwörter.	Numerisch (N3)
8	Anzahl der Eintrag-3-Schlüsselwörter.	Numerisch (N3)
9	Anzahl katalogisierter Funktionen.	Numerisch (N7)

TEXT für Schlüsselwort-Informationen

Bei Schlüsselwort-Informationen muss COMMAND-LINE (1) das betreffende Schlüsselwort enthalten; COMMAND-LINE (2) kann bei Bedarf den Schlüsselwort-Typ (1, 2, 3 oder P) enthalten; COMMAND-LINE (3:6) muss leer sein.

RESULT-FIELD	Inhalt	Format
1	Schlüsselwort-Kommentartext	Text (A40)
2	Schlüsselwort in voller Länge	Text (A16)
3	Schlüsselwort eindeutig abgekürzt	Text (A16)
4	„Keyword used as“-Eintrag (Schlüsselwort benutzt als)	Text (A16)
5	Interne Schlüsselwortnummer (IKN)	Numerisch (N4)
6	Mindestlänge des Schlüsselworts	Numerisch (N2)
7	Maximallänge des Schlüsselworts	Numerisch (N2)
8	Schlüsselwort-Typ (1, 2, 3, 1S, 2S, 3S, P)	Text (A2)

TEXT für Funktionsinformationen

Bei Funktionsinformationen muss COMMAND-LINE (1:3) die Schlüsselwörter enthalten, die den gewünschten Platz bestimmen. COMMAND-LINE (4:6) muss die Schlüsselwörter enthalten, die die gewünschte Funktion bestimmen. Falls beispielsweise Informationen über das globale Kommando ADD USER gewünscht werden, müssen die Kommandozeilen 1, 2, 3 und 6 leer sein, in Kommandozeile 4 muss ADD stehen und in Kommandozeile 5 USER.

RESULT-FIELD	Inhalt	Format
1	Text wie mit Option T in Laufzeit-Aktion definiert.	Text (A40)
2	Interne Funktionsnummer (IFN) des angegebenen Platzes.	Numerisch (N10)
3	Interne Funktionsnummer (IFN) der angegebenen Funktion.	Numerisch (N10)

GET-Option

Die Option GET dient dazu, interne Kommando-Prozessor-Informationen und die aktuellen Kommando-Prozessor-Einstellungen aus dem dynamisch zugewiesenen NCPWORK-Puffer zu lesen. Folgende Felder werden verwendet:

Feldname	Inhalt
GETSET-FIELD-NAME (A32)	Der Name der Variablen, die gelesen werden soll.
GETSET-FIELD-VALUE (A32)	Der Wert der angegebenen Variablen nach der Ausführung von PROCESS COMMAND ACTION GET.

Eine Liste der möglichen Werte von GETSET-FIELD-NAME finden Sie [weiter unten](#).

SET-Option

Die Option SET dient dazu, interne Einstellungen des Kommando-Prozessors im NCPWORK-Puffer zu ändern.

Feldname	Inhalt
GETSET-FIELD-NAME (A32)	Der Name der Variablen, die geändert werden soll.
GETSET-FIELD-VALUE (A32)	Der Wert, der der angegebenen Variablen zugewiesen werden soll.

Die möglichen Werte von GETSET-FIELD-NAME sind:

Feldname	Format	G/S*	Inhalt
NAME	A8	G	Aktueller Prozessor-Name.
LIBRARY	A8	G	Geladen aus Library.
FNR	N10	G	Geladen aus Datei.
DBID	N10	G	Geladen aus Datenbank.
TIMESTAMP	A8	G	Zeitstempel des aktuellen Prozessors.
COUNTER	N10	G	Zugriffszähler.
BUFFER-LENGTH	N10	G	Für NCPWORK allozierte Bytes.
C-DELIMITER	A1	G/S	Delimiter für mehrere Kommandos.
DATA-DELIMITER	A1	G	Präfix für Daten.
PF-KEY	A1	G/S	PF-Taste kann Kommando sein (Y/N).
UPPER-CASE	A1	G	Schlüsselwörter in Großbuchstaben (Y/N).
UQ-KEYWORDS	A1	G	Eindeutige Schlüsselwörter (Y/N).
IMPLICIT-KEYWORD	A1	G/S	Identifiziert impliziten Schlüsselwort-Eintrag.
MIN-LEN	N10	G	Schlüsselwort-Mindestlänge.
MAX-LEN	N10	G	Schlüsselwort-Maximallänge.
KEYWORD-SEQ	A8	G/S	Schlüsselwort-Reihenfolge.
ALT-KEYWORD-SEQ	A8	G/S	Alternative Schlüsselwort-Reihenfolge.
USER-SEQUENCE	A1	G	Benutzer darf KEYWORD-SEQ überschreiben (Y/N).
CURR-LOCATION	N10	G/S	Aktueller Platz (IFN).
CURR-IKN1	N10	G/S	IKN1 des aktuellen Platzes.

Feldname	Format	G/S*	Inhalt
CURR-IKN2	N10	G/S	IKN2 des aktuellen Platzes.
CURR-IKN3	N10	G/S	IKN3 des aktuellen Platzes.
CHECK-LOCATION	N10	G	Letzter geprüfter Platz (IFN).
CHECK-IKN1	N10	G	IKN1 von CHECK-LOCATION.
CHECK-IKN2	N10	G	IKN2 von CHECK-LOCATION.
CHECK-IKN3	N10	G	IKN3 von CHECK-LOCATION.
TOP-IKN1	N10	G	IKN1 des obersten Schlüsselworts.
TOP-IKN2	N10	G	IKN2 des obersten Schlüsselworts.
TOP-IKN3	N10	G	IKN3 des obersten Schlüsselworts.
KEY1-TOTAL	N10	G	Anzahl an Schlüsselwörtern vom Typ 1.
KEY2-TOTAL	N10	G	Anzahl an Schlüsselwörtern vom Typ 2.
KEY3-TOTAL	N10	G	Anzahl an Schlüsselwörtern vom Typ 3.
FUNCTIONS-TOTAL	N10	G	Anzahl der katalogisierten Funktionen.
LOCAL-GLOBAL-SEQ	A8	G/S	Lokale/globale Funktionsvalidierung.
ERROR-HANDLER	A8	G/S	Allgemeines Fehlerprogramm.
SECURITY	A1	G	Natural Security installiert (Y/N).
SEC-PREFETCH	A1	G	Natural Security-Daten sollen gelesen werden (Y/N) bzw. wurden gelesen (D = Done/erledigt).
PREFIX1	A1	G	Entspricht dem Feld Prefix Character 1 des Schirms Processor Header Maintenance 2 der SYSNCP-Utility, siehe Abschnitt <i>Keyword Editor Options - Header 2</i>
PREFIX2	A1	G	Entspricht dem Feld Prefix Character 2 des Schirms Processor Header Maintenance 2 der SYSNCP-Utility.
HEX1	A1	G	Entspricht dem Feld Hex. Replacement 1 des Schirms Processor Header Maintenance 2 der SYSNCP-Utility.
HEX2	A1	G	Entspricht dem Feld Hex. Replacement 2 des Schirms Processor Header Maintenance 2.
DYNAMIC	A32	G	Dynamischer Teil (:n:) der letzten Fehlermeldung.
LAST	-	G	Letztes oben auf dem Stack als Daten abgelegtes Kommando.
LAST-ALL	-	G	Letzte oben auf dem Stack als Daten abgelegte Kommandos.
LAST-COM	-	G	Letztes in *COM gestelltes Kommando.
MULTI	-	G	Legt das letzte von mehreren Kommandos als Daten oben auf dem Stack ab.
MULTI-COM	-	G	Legt das letzte von mehreren Kommandos in der Systemvariablen *COM ab.

*G = Kann mit der **GET**-Option verwendet werden.

*S = Kann mit der **SET**-Option verwendet werden.

USING-Klausel

Die Inhalte der Felder in der USING-Klausel bestimmen zum Beispiel den Prozessor-Namen und die Kommandozeile.

In der USING-Klausel werden die Felder angegeben, die an den Kommando-Prozessor übergeben werden.

Option	Feldname			
	PROCESSOR-NAME	COMMAND-LINE	GETSET-FIELD-NAME	GETSET-FIELD-VALUE
CLOSE				
CHECK	M	M		
EXEC	M	M		
TEXT	M	M		
HELP	M	M		
GET	M		M	
SET	M		M	M

M = Feld muss angegeben werden.

R = Feld sollte angegeben werden (muss aber nicht).

GIVING-Klausel



Anmerkung: Diese Klausel kann nur im Reporting Mode verwendet werden.

In der GIVING-Klausel werden die Felder angegeben, die vom Kommando-Prozessor gefüllt werden, wenn eine Option verarbeitet wird.

Option	Feldname			
	NATURAL-ERROR	RETURN-CODE	RESULT-FIELD	GETSET-FIELD-VALUE
CLOSE	R			
CHECK	R	M	M	
EXEC	R	M	M	
TEXT	R	M	M	
HELP	R	M	M	
GET	R			M
SET	R			

M = Feld muss angegeben werden.

R = Feld sollte angegeben werden (muss aber nicht).



Anmerkung: Die GIVING-Klausel kann im Structured Mode nicht verwendet werden, da es eine implizite GIVING-Klausel gibt, die sich aus allen im DEFINE DATA-Statement angegebenen Feldern zusammensetzt, die für gewöhnlich in der GIVING-Klausel des Reporting Modes referenziert werden. Das bedeutet, dass im Structured Mode alle in obiger Tabelle mit M markierten Felder im DEFINE DATA-Statement definiert sein müssen.

Das DDM COMMAND

Das DDM COMMAND wurde speziell zur Verwendung mit dem PROCESS COMMAND-Statement erstellt:

```

DB:      1 File:      1 - COMMAND                      Default Sequence: ?

TYL  DB  NAME                                F  LENG  S  D  REMARKS
-----
  1  AA  PROCESSOR-NAME                          A    8  N  D  DE  USING
M  1  AB  COMMAND-LINE                              A   80  N  D  MU/DE  USING
  1  AF  GETSET-FIELD-NAME                      A   32  N  D  DE  USING
  1  BA  NATURAL-ERROR                          N   4.0  N           GIVING
  1  BB  RETURN-CODE                              A    4  N           GIVING
M  1  BC  RESULT-FIELD                              A   80  N  MU  GIVING
  1  BD  GETSET-FIELD-VALUE                      A   32  N  D           USING; GIVING
***** DDM OUTPUT TERMINATED *****

```



Anmerkung: Um mögliche Kompilierungs- bzw. Laufzeitfehler zu vermeiden, prüfen Sie bitte, ob das DDM COMMAND als Typ C (Feld DDM Type auf dem SYSDDM-Menü) katalogisiert ist, bevor Sie es verwenden. (Falls Sie es neu katalogisieren, werden hierbei etwaige DBID/FNR-Angaben in SYSDDM ignoriert.)

Das DDM COMMAND enthält folgende Felder:

DDM-Feld	Erläuterung
PROCESSOR-NAME	Der Name des Kommando-Prozessors, für den das PROCESS COMMAND-Statement ausgeführt wird. Der Prozessor muss katalogisiert sein.
COMMAND-LINE	Die Kommandozeile, die vom Kommando-Prozessor verarbeitet werden soll (Optionen CHECK, EXEC) bzw. das Schlüsselwort/Kommando, für das Benutzertext oder Hilfe-Text an das Programm zurückgegeben werden soll (Optionen TEXT, HELP). Bitte beachten Sie, dass dieses Feld aus mehr als einer Zeile bestehen kann.
GETSET-FIELD-NAME	Dieses Feld wird mit den Optionen GET und SET verwendet und dient zur Angabe des Namens der Konstanten/Variablen, die gelesen (GET) oder geschrieben (SET) werden soll.

DDM-Feld	Erläuterung
RETURN-CODE	Dieses Feld enthält den Return Code einer Aktion aufgrund der Option EXEC oder CHECK , wie in einer Laufzeit-Aktion angegeben (siehe SYSNCP-Utility).
NATURAL-ERROR	Dieses Feld wird mit allen Optionen verwendet. Wenn es im DEFINE DATA -Statement angegeben wird, enthält es den Natural-Fehlercode für den Kommando-Prozessor. Wenn das Feld fehlt, ist die normale Natural-Fehlerbehandlung aktiv, wenn ein Fehler auftritt.
RESULT-FIELD	Dieses Feld enthält Informationen, die aus der Verwendung verschiedener Optionen, wie in einer Laufzeit-Aktion angegeben, resultieren (siehe den Abschnitt <i>Runtime Actions</i> in der SYSNCP-Utility-Dokumentation). Bitte beachten Sie, dass dieses Feld aus mehr als einer Zeile bestehen kann.
GETSET-FIELD-VALUE	Dieses Feld wird mit den Optionen GET und SET verwendet und enthält den Wert der im Feld GETSET-FIELD-NAME angegebenen Konstanten/Variablen (siehe oben).

Beispiele

- [Beispiel 1 — PROCESS COMMAND ACTION CLOSE](#)
- [Beispiel 2 — PROCESS COMMAND ACTION EXEC2](#)

Beispiel 1 — PROCESS COMMAND ACTION CLOSE

```

/* EXAM-CLS - Example for PROCESS COMMAND ACTION CLOSE (Structured Mode)
/*****
DEFINE DATA LOCAL
  01 COMMAND VIEW OF COMMAND
END-DEFINE
/*
PROCESS COMMAND ACTION CLOSE
/*
DEFINE WINDOW CLS
INPUT WINDOW = 'CLS'
  'NCPWORK has just been released.'
/*
END

```


Beispiel 2 — PROCESS COMMAND ACTION EXEC2

```

/* EXAM-EXS - Example for PROCESS COMMAND ACTION EXEC (Structured Mode)
/*****
DEFINE DATA LOCAL
  01 COMMAND VIEW OF COMMAND
    02 PROCESSOR-NAME
    02 COMMAND-LINE (1)
    02 NATURAL-ERROR
    02 RETURN-CODE
    02 RESULT-FIELD (1)
  01 MSG (A65) INIT <'Please enter a command.'>
END-DEFINE
/*
REPEAT
  INPUT (AD=MIT' ' IP=OFF) WITH TEXT MSG
    'Example for PROCESS COMMAND ACTION EXEC (Structured Mode)' (I)
  / 'Command ==>' COMMAND-LINE (1) (AL=64)
  /*****
  PROCESS COMMAND ACTION EXEC
  USING
    PROCESSOR-NAME = 'DEMO'
    COMMAND-LINE (1) = COMMAND-LINE (1)
  /*****
  COMPRESS 'NATURAL-ERROR =' NATURAL-ERROR TO MSG
END-REPEAT
END

```



Anmerkung: Weitere Beispielprogramme finden Sie in der Library SYSNCP. Die Namen der Beispielprogramme beginnen alle mit EXAM.

98

PROCESS GUI

▪ Funktion	640
▪ Syntax-Beschreibung	640

```
PROCESS GUI ACTION action-name WITH { { operand1 } ... } [GIVING operand2]
                                { nX }
                                { PARAMETERS-clause }
```

Dieses Kapitel behandelt folgende Themen:

Verwandtes Statement: [OPEN DIALOG](#) | [CLOSE DIALOG](#) | [SEND EVENT](#)

Gehört zur Funktionsgruppe: [Ereignisgesteuerte Programmierung](#)

Funktion

Das `PROCESS GUI`-Statement dient dazu, eine Aktion auszuführen. Eine Aktion ist in diesem Zusammenhang eine Prozedur, die in ereignisgesteuerten Anwendungen häufig benötigt wird.

Allgemeine Informationen zu diesen Standardprozeduren finden Sie unter *Event-Driven Programming Techniques* (im Leitfaden zur Programmierung).

Informationen zu den einzelnen zur Verfügung stehenden Aktionen, deren Parameter und Beispiele finden Sie unter *PROCESS GUI Statement Actions* (in der *Dialog Component Reference*).

Syntax-Beschreibung

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i> *	C S A	A U N P I F B D T L G	ja	nein
<i>operand2</i>	S	N P I	ja	nein

* Die wirklich mögliche Struktur und das wirklich mögliche Format sind abhängig von der auszuführenden Aktion.

<i>action-name</i>	Auszuführende Aktion Als <i>action-name</i> geben Sie den Namen der auszuführenden Aktion ein.
<i>operand1</i>	Parameterübergabe an die Aktion: Als <i>operand1</i> geben Sie die Parameter ein, die an die Aktion übergeben werden sollen. Die Parameter werden in der Reihenfolge übergeben, wie sie angegeben wurden.
<i>PARAMETERS</i>	Siehe Parameter mit dem Namen übergeben unten.

nX	<p>Angabe zu überspringender Parameter:</p> <p>Mit der Notation nX können Sie angeben, dass die nächsten n Parameter übersprungen werden sollen (zum Beispiel $1X$, um den nächsten Parameter zu überspringen, oder $3X$, um die nächsten 3 Parameter zu überspringen); dies bedeutet, dass für die nächsten n Parameter keine Werte an die Aktion übergeben werden. Dies ist nur möglich bei Aktionen, die bei ActiveX-Controls angewendet werden.</p> <p>Ein zu überspringender Parameter muss als „optional“ in der Methode des ActiveX-Controls definiert sein. Wenn ein Parameter als „optional“ definiert ist, bedeutet dies, dass ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann, aber nicht muss.</p>
GIVING <i>operand2</i>	<p>Feld für for Response-Code:</p> <p>Als <i>operand2</i> können Sie ein Feld angeben, das den Response-Code der aufgerufenen Aktion empfangen soll nachdem die Aktion ausgeführt wurde.</p>

Parameter mit dem Namen übergeben:

Bei der Aktion „ADD“ können Sie die Parameter auch mit dem Namen übergeben (statt mit der Position); hierzu verwenden Sie die *PARAMETERS-Klausel*:

```
PARAMETERS {parameter-name=operand1} ...
END-PARAMETERS
```

Diese Klausel kann nur mit der Aktion „ADD“ verwendet werden und nicht mit einer anderen Aktion.

Wenn die Aktion optionale Parameter hat (d.h. Parameter, die nicht angegeben werden müssen), können Sie die Notation nX als Platzhalter für n nicht angegebene Parameter verwenden. Die einzigen Aktionen, die zur Zeit optionale Parameter haben können, sind die Methoden und parametrisierten Properties der ActiveX-Controls.

99

PROCESS PAGE

▪ Funktion	644
▪ Syntax 1 — PROCESS PAGE	644
▪ Syntax 2 — PROCESS PAGE USING	647
▪ Syntax 3 — PROCESS PAGE UPDATE	650
▪ Syntax 4 — PROCESS PAGE MODAL	653
▪ Beispiele	655

Dieses Kapitel behandelt folgende Themen:

Funktion

Das `PROCESS PAGE`-Statement bildet eine allgemeine Schnittstellen-Beschreibung zu einer externen Rendering-Maschine, wie z.B. Natural for Ajax, wobei somit die interne Natural-Datendarstellung mit einer externen Datendarstellung verknüpft wird. Über diese Verknüpfung werden Daten und Ereignisse, aber keine Rendering-Informationen an und von eine/r externen browser-basierte/n Anwendung versandt.

Weitere Informationen entnehmen Sie der *Natural for Ajax*-Dokumentation.

Syntax 1 — PROCESS PAGE

```
PROCESS PAGE [(parameter)] operand1
  [WITH PARAMETERS
    {[NAME] operand3 [VALUE] operand4 [(parameters)]} ...
  END-PARAMETERS]
[GIVING operand11]
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Gehört zur Funktionsgruppe: [Bildschirmgenerierung für interaktive Verarbeitung](#)

Syntax-Beschreibung — Syntax 1

Die Syntax 1 des `PROCESS PAGE`-Statements wird normalerweise nur in einem Natural-Adapter benutzt. Ein Adapter ist ein Natural-Objekt, das die Schnittstelle zwischen dem Natural-Anwendungscode und der Webseite bildet. Er wird automatisch von Natural for Ajax erstellt/aktualisiert, wenn das Layout gespeichert wird.

Operanden-Definitionstabelle:

Operand	Possible Structure			Possible Formats											Referencing Permitted	Dynamic Definition			
<i>operand1</i>	C	S				A	U											yes	no
<i>operand2</i>		S	A														C	no	no
<i>operand3</i>	C	S				A	U											yes	no
<i>operand4</i>	C	S	A			A	U	N	P	I	F	B	D	T	L			yes	yes
<i>operand5</i>		S	A														C	no	no
<i>operand11</i>		S								I4								yes	yes

Syntax-Element-Beschreibung:

<i>parameter</i>	Den Parameter CV können Sie in Klammern angeben, um eine oder mehrere der in <i>operand2</i> angegebenen Kontrollvariablen zu referenzieren: (CV= <i>operand2</i>) Siehe auch <i>Logische Bedingungen</i> , <i>MODIFIED-Option</i> im <i>Leitfaden zur Programmierung</i> .	
<i>operand1</i>	Enthält den Namen des externen Seiten-Layouts.	
<i>operand2</i>	<i>operand2</i> enthält den Namen der Kontrollvariablen, muss Format C haben und entweder ein Skalar oder eine einzelne Array-Ausprägung sein.	
<i>operand3</i>	Enthält den/die Namen des bzw. der externen Datenfelder, in die oder aus denen <i>operand4</i> übertragen wird.	
<i>operand4</i>	Enthält den/die Namen des bzw. der externen Natural-Datenfelder, die übertragen werden.	
<i>parameters</i>	Unmittelbar nach <i>operand4</i> können Sie einen oder mehrere Parameter in Klammern angeben: EM or EMU Während des Datentransfers verwendete Editiermaske. Weitere Informationen siehe Session-Parameter EM in der <i>Parameter-Referenz</i> . Weitere Informationen zu Unicode-Editiermasken siehe Session-Parameter EMU in der <i>Parameter-Referenz</i> .	

	CV	<p>Den Parameter <i>CV</i>, können Sie angeben, um eine oder mehrere in <i>operand5</i> angegebene Kontrollvariable zu referenzieren:</p> <p><i>(CV=operand5)</i></p> <p>Siehe auch <i>Logische Bedingungen</i>, <i>MODIFIED-Option</i> im <i>Leitfaden zur Programmierung</i>.</p>
<i>operand5</i>	<p><i>operand5</i> enthält den Namen der Kontrollvariablen. Die Variable muss Format C haben.</p> <p>Falls <i>operand4</i> ein Skalar-Ausdruck oder eine einzelne Array-Ausprägung ist, muss <i>operand5</i> folgendes sein:</p> <ul style="list-style-type: none"> ■ ein Skalar-Ausdruck ■ oder eine einzelne Array-Ausprägung. <p>Falls es sich bei <i>operand4</i> um den vollen Bereich eines Array der Dimension 1 handelt, muss <i>operand5</i> folgendes sein:</p> <ul style="list-style-type: none"> ■ ein Skalar-Ausdruck ■ oder eine einzelne Array-Ausprägung ■ oder der volle Bereich eines Array der Dimension 1 mit der gleichen Größe. <p>Falls es sich bei <i>operand4</i> um den vollen Bereich eines Array der Dimension 2 handelt, muss <i>operand5</i> folgendes sein:</p> <ul style="list-style-type: none"> ■ ein Skalar-Ausdruck ■ oder eine einzelne Array-Ausprägung ■ oder der volle Bereich eines Array der Dimension 2 mit der gleichen Größe in beiden Dimensionen ■ oder der volle Bereich eines Array der Dimension 1 mit der gleichen Größe, die <i>operand4</i> in Dimension 2 hat. 	
GIVING <i>operand11</i>	<p>GIVING-Klausel:</p> <p><i>operand11</i> enthält den Natural-Fehler, wenn die Anfrage nicht ausgeführt werden konnte.</p>	

Beispiel eines Adapters, der vom Natural for Ajax erstellt wurde:

```

* PAGE1: PROTOTYPE      --- CREATED BY Natural for Ajax ---
* PROCESS PAGE USING 'XXXXXXXX' WITH
* INFOPAGENAME RESULT YOURNAME
DEFINE DATA PARAMETER
1 INFOPAGENAME (U) DYNAMIC
1 RESULT (U) DYNAMIC
1 YOURNAME (U) DYNAMIC
END-DEFINE
    
```

```

*
PROCESS PAGE U'/njxdemos/helloworld' WITH
PARAMETERS
  NAME U'infopagename'
  VALUE INFOPAGENAME
  NAME U'result'
  VALUE RESULT
  NAME U'yourname'
  VALUE YOURNAME
END-PARAMETERS
*
* TODO: Copy to your calling program and implement.
/*/*( DEFINE EVENT HANDLER
* DECIDE ON FIRST *PAGE-EVENT
* VALUE U'nat:page.end'
* /* Page closed.
* IGNORE
* VALUE U'onHelloWorld'
* /* TODO: Implement event code.
* PROCESS PAGE UPDATE FULL
* NONE VALUE
* /* Unhandled events.
* PROCESS PAGE UPDATE
* END-DECIDE
/*/*) END-HANDLER
*
END

```

Syntax 2 — PROCESS PAGE USING

```

PROCESS PAGE USING operand6
  [ { WITH {operand7} ... }
    NO PARAMETER ]
[GIVING operand11]

```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Gehört zur Funktionsgruppe: [Bildschirmgenerierung für interaktive Verarbeitung](#)

Syntax-Beschreibung — Syntax 2

Diese Syntax wird benutzt, um eine Rich-GUI-Eingabe/Ausgabeverarbeitung mittels eines Objekts des Typs Adapter auszuführen, das aus einem Seiten-Layout angelegt wurde, welches mit Natural for Ajax oder einem ähnlichen Tool erstellt wurde.

Operanden-Definitionstabelle:

Operand	Possible Structure	Possible Formats	Referencing Permitted	Dynamic Definition
<i>operand6</i>	C S	A	yes	no
<i>operand7</i>	S A G	A U N P I F B D T L C	yes	yes
<i>operand11</i>	S	I4	yes	yes

Syntax-Element-Beschreibung:

USING <i>operand6</i>	<p>Adapter-Name:</p> <p>Ruft eine Adapter-Definition auf, die vorher in einer Natural-Systemdatei gespeichert wurde. Siehe auch <i>Verarbeitung einer Rich GUI Page - Adapter</i> im Leitfaden zur Programmierung.</p> <p>Der Adapter-Name (<i>operand6</i>) kann eine 1 bis 8 Zeichen umfassende, alphanumerische Konstante oder Benutzervariable sein. Wenn eine Variable benutzt wird, muss sie vorher definiert worden sein.</p> <p>Der Adapter-Name kann ein Kaufmännisches Und (&) enthalten; zur Ausführungszeit wird dieses Zeichen durch den aktuellen Wert der Natural-Systemvariablen *LANGUAGE ersetzt. Diese Funktion gibt es aus historischen Gründen. Wenn Sie mehrsprachige Adapter benötigen, machen Sie sich die Funktionalität des externen Rendering-Systems (zum Beispiel: Natural for Ajax) zunutze.</p> <p>Anmerkung: Bei neuen Anwendungen muss die &-Funktion nicht mehrsprachig sein. Seiten, die zum Beispiel mit Natural for Ajax gestaltet wurden, können mehrsprachige Informationen als Bestandteil des Layout-Designs aufnehmen. Siehe <i>Multi Language Management</i> in der <i>Natural for Ajax</i>-Dokumentation.</p>
<i>operand7</i>	<p>Feld-Spezifikation:</p> <p>Eine Liste der Datenbank-Felder und/oder Benutzervariablen, die alle vorher definiert sein müssen. Die Felder müssen in Anzahl, Reihenfolge, Format, Länge und (für Arrays) der Anzahl der Ausprägungen mit den Feldern im referenzierten Adapter übereinstimmen; sonst tritt ein Fehler auf.</p> <p>Wenn der Inhalt eines Datenbank-Feldes als Ergebnis der PROCESS PAGE-Verarbeitung geändert wird, wird nur der Wert geändert, wie er in der Data Area abgelegt ist. Um den Inhalt der Datenbank zu ändern, müssen die passenden UPDATE-/STORE-Statements für die Datenbank benutzt werden.</p>

	Siehe PROCESS PAGE USING mit im Programm definierten Feldern .
NO PARAMETER	Siehe PROCESS PAGE USING ohne Parameter-Liste .
GIVING <i>operand11</i>	<p>GIVING-Klausel:</p> <p><i>operand11</i> enthält den Natural-Fehler, wenn die Anfrage nicht ausgeführt werden konnte.</p> <p>Anmerkung: Die GIVING-Klausel unterbricht die allgemeine Natural-Fehlerbehandlung, wenn ein Fehler auftritt, während das Adapter-Objekt aktiviert oder ausgeführt wird. Anstatt die Natural-Module zurückzuverfolgen, um eine ON ERROR-Klausel zu finden, wird der Natural-Fehlercode an eine Variable (<i>operand11</i>) übergeben, und die Ausführung wird mit dem nächsten Statement fortgesetzt.</p>

PROCESS PAGE USING ohne Parameter-Liste

Die folgenden Anforderungen müssen erfüllt sein, wenn PROCESS PAGE USING ohne Parameter-Liste benutzt wird:

- Der Adapter-Name (*operand7*) muss als eine alphanumerische Konstante (bis zu 8 Zeichen) angegeben werden.
- Der auf diese Art benutzte Adapter muss vor der Kompilierung des Programms erstellt worden sein, welches den Adapter referenziert.
- Die Namen der zu verarbeitenden Felder werden dynamisch aus der Adapter-Quelldefinition zur Kompilierungszeit übernommen. Die sowohl im Programm als auch im Adapter verwendeten Feldnamen müssen identisch sein.
- Alle im PROCESS PAGE-Statement zu referenzierenden Felder müssen an diesem Punkt aufrufbar sein.
- Im Structured Mode müssen die Felder vorher definiert worden sein (Datenbank-Felder müssen für Verarbeitungsschleifen oder Views ordnungsgemäß referenziert werden).
- Wenn das Seiten-Layout geändert wird, müssen die den Adapter verwendenden Programme nicht neu katalogisiert werden. Wenn aber die Array-Strukturen oder Namen, Formate/Längen der Felder geändert werden, oder wenn Felder zum Adapter hinzugefügt oder aus ihm gelöscht werden, müssen die den Adapter benutzenden Programme neu katalogisiert werden.
- Der Adapter-Quellcode muss zur Programm-Kompilierung zur Verfügung stehen; sonst kann das PROCESS PAGE USING-Statement nicht kompiliert werden.



Anmerkung: Wenn Sie das Programm kompilieren möchten, auch wenn der Adapter noch nicht zur Verfügung steht, geben Sie NO PARAMETER an. Das PROCESS PAGE USING-Statement kann dann kompiliert werden, auch wenn der Adapter noch nicht verfügbar ist.

PROCESS PAGE USING mit im Programm definierten Feldern

Wenn Sie die Namen der Felder angeben, die innerhalb des Programms (*operand7*) verarbeitet werden sollen, ist es möglich, dass Sie es so einrichten können, dass die Namen der Felder im Programm sich von den Namen der Felder im Adapter unterscheiden.

Die Reihenfolge der Felder im Programm muss mit der Reihenfolge im Adapter übereinstimmen. Wenn Sie Natural-Maps als Adapter-Objekte benutzen, beachten Sie, dass der Map-Editor die Felder so sortiert, wie in der Map angegeben, und zwar in alphabetischer Reihenfolge nach Feldnamen. Weitere Informationen siehe die Beschreibung des Map Editor in der *Editors*-Dokumentation.

Zur Ausführungszeit findet eine Überprüfung statt, um sicherzustellen, dass das im Programm angegebene Format und die im Programm angegebene Länge der Felder mit den Feldern übereinstimmt, die im Adapter spezifiziert sind. Falls die beiden Layouts nicht miteinander übereinstimmen, wird eine Fehlermeldung erzeugt.

Syntax 3 — PROCESS PAGE UPDATE

```
PROCESS PAGE UPDATE [FULL] [event-option]  
[GIVING operand11]
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Gehört zur Funktionsgruppe: *Bildschirmgenerierung für interaktive Verarbeitung*

Syntax-Beschreibung - Syntax 3

Das PROCESS PAGE UPDATE-Statement wird benutzt, um zu einem vorangegangenen PROCESS PAGE-Statement zurückzukehren und es neu auszuführen. Es wird im Allgemeinen benutzt, um von der Ereignisverarbeitung zurückzukehren, da die Dateneingabe-Verarbeitung des vorangegangenen PROCESS PAGE-Statements unvollständig war.



Anmerkung: Es kann kein INPUT-, WRITE-, PRINT- oder DISPLAY-Statement zwischen einem PROCESS PAGE- und seinem entsprechenden PROCESS PAGE UPDATE-Statement ausgeführt werden.

Wenn es erst einmal ausgeführt ist, repositioniert das PROCESS PAGE UPDATE-Statement den Programm-Status bezüglich Subroutine, Sonderbedingungen und die Schleifenverarbeitung, wie sie existierten, als das PROCESS PAGE-Statement ausgeführt wurde (solange der Status des PROCESS PAGE-Statements noch aktiv ist). Wenn die Schleife nach der Ausführung des PROCESS PAGE-Statements initialisiert wurde und sich das PROCESS PAGE UPDATE-Statement innerhalb dieser

Schleife befindet, wird die Schleife unterbrochen und dann neu gestartet, nachdem das `PROCESS PAGE`-Statement als infolge eines `PROCESS PAGE UPDATE`-Statements neu verarbeitet worden ist.

Wenn eine Hierarchie von Subroutinen nach der Ausführung des `PROCESS PAGE`-Statements aufgerufen wurde und wenn das `PROCESS PAGE UPDATE`-Statement innerhalb einer Subroutine ausgeführt wird, verfolgt Natural automatisch alle Subroutinen zurück und setzt den Programm-Status auf den des `PROCESS PAGE`-Statements zurück.

Es ist aber nicht möglich, es so einzurichten, dass ein `PROCESS PAGE`-Statement in einer Schleife, einer Subroutine oder einem speziellen Bedingungsblock positioniert wird, und dann das `PROCESS PAGE UPDATE`-Statement auszuführen, wenn der Status, unter dem das `PROCESS PAGE`-Statement ausgeführt wurde, bereits beendet worden ist. Eine Fehlermeldung wird erzeugt und die Programmausführung beendet, wenn eine solche Fehlerbedingung festgestellt wird.

Operanden-Definitionstabelle:

Operand	Possible Structure	Possible Formats	Referencing Permitted	Dynamic Definition
<i>operand11</i>	S	I4	yes	yes

Syntax-Element-Beschreibung:

FULL	<p>Wenn Sie die <code>FULL</code>-Option in einem <code>PROCESS PAGE UPDATE</code>-Statement angeben, wird das entsprechende <code>PROCESS PAGE</code>-Statement vollständig neu ausgeführt:</p> <ul style="list-style-type: none"> ■ Mit einem normalen <code>PROCESS PAGE UPDATE</code>-Statement (ohne <code>FULL</code>-Option) wird der Inhalt von Variablen, die zwischen dem <code>PROCESS PAGE</code>- und dem <code>PROCESS PAGE UPDATE</code>-Statement geändert wurden, nicht angezeigt; das heisst, alle Variablen auf dem Bildschirm zeigen den Inhalt an, den sie hatten, als das <code>PROCESS PAGE</code>-Statement ursprünglich ausgeführt wurde. ■ Mit einem <code>PROCESS PAGE UPDATE FULL</code>-Statement werden alle Änderungen, die nach der anfänglichen Ausführung des <code>PROCESS PAGE</code>-Statements erfolgt sind, auf das <code>PROCESS PAGE</code>-Statement angewandt, wenn es neu ausgeführt wird; das heisst, alle Variablen auf dem Bildschirm enthalten die Werte, die sie hatten, als das <code>PROCESS PAGE UPDATE</code>-Statement ausgeführt wurde.
<i>event-option</i>	<p>EVENT-Option:</p> <p>Siehe <i>EVENT-Option</i> weiter unten.</p>
GIVING <i>(operand11)</i>	<p>GIVING-Klausel:</p> <p><i>operand11</i> enthält den Natural-Fehler, wenn die Anfrage nicht durchgeführt werden konnte.</p>

Beispiel eines Benutzerprogrammfragments:

```

PROCESS PAGE USING "HELLOW-A"
*
/*( DEFINE EVENT HANDLER
DECIDE ON FIRST *PAGE-EVENT
VALUE U'nat:page.end'
/* Page closed.
IGNORE
VALUE U'onHelloWorld'
COMPRESS "HELLO WORLD" YOURNAME INTO RESULT
PROCESS PAGE UPDATE FULL
NONE VALUE
/* Unhandled events.
PROCESS PAGE UPDATE
END-DECIDE
/*) END-HANDLER
    
```

EVENT-Option

```

AND SEND EVENT operand8
[WITH PARAMETERS
{[NAME] operand9 [VALUE] operand10 [ { (EMU=value)
(EM=value) } ]...
END-PARAMETERS]
    
```

Mit dieser Option können Sie das externe E/A-System veranlassen, Sonderfunktionen zu starten. Diese Funktionen sind Bestandteil des externen E/A-Systems, oder implementieren Sonderfunktionen im Hinblick auf die Ausgabeverarbeitung, wie z.B. Fokus setzen, Meldungsfelder anzeigen usw.

Operanden-Definitionstabelle:

Operand	Possible Structure		Possible Formats										Referencing Permitted	Dynamic Definition							
<i>operand8</i>	C	S					A	U											yes	no	
<i>operand9</i>	C	S					A	U											yes	no	
<i>operand10</i>	C	S	A				A	U	N	P	I	F	B	D	T	L				yes	yes

Syntax-Element-Beschreibung:

<i>operand8</i>	<p>Vom externen E/A-System angefordertes Ereignis:</p> <p>Abhängig von der Implementierung des externen E/A-Systems sind Ereignisse verfügbar, siehe <i>Sending Events to the User Interface</i> in der <i>Natural for Ajax</i>-Dokumentation.</p>
<i>operand9</i>	<p>Externer Datenfeld-Name:</p> <p><i>operand9</i> enthält den externen Namen der Datenfelder, in die/aus denen <i>operand10</i> übertragen wird.</p>
<i>operand10</i>	<p>Natural-Datenfelder:</p> <p><i>operand10</i> enthält die Natural-Datenfelder, die übertragen werden.</p>
EMU= EM=	<p>Editiermaske:</p> <p>Bei der Datenübertragung verwendete Editiermaske.</p> <p>Einzelheiten zu Editiermasken siehe Session-Parameter EM in der <i>Parameter-Referenz</i>.</p> <p>Einzelheiten zu Unicode-Editiermasken siehe Session-Parameter EMU in der <i>Parameter-Referenz</i>.</p>

Syntax 4 — PROCESS PAGE MODAL

```
PROCESS PAGE MODAL
  statement ...
END-PROCESS
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [ESCAPE](#) | [PROCESS PAGE](#)

Gehört zur Funktionsgruppe:

- [Schleifenverarbeitung](#)
- [Bildschirmgenerierung für interaktive Verarbeitung](#)

Syntax-Beschreibung — Syntax 4

Das `PROCESS PAGE MODAL`-Statement wird benutzt, um einen Verarbeitungsblock zu initialisieren und die Anzeigedauer eines modalen Rich-GUI-Fensters zu steuern.



Anmerkung: Das `PROCESS PAGE MODAL`-Statement ist im Stapelbetrieb nicht gültig.

Wenn gerade der `PROCESS PAGE MODAL`-Statementblock abgearbeitet wird, erscheinen zuerst Daten vom Report 0, die noch nicht angezeigt worden sind.

Die Systemvariable `*PAGE-LEVEL` wird inkrementiert, und das Öffnen einer modalen Seite wird vorbereitet. Das physische Öffnen der modalen Seite wird mit dem nächsten Statement `PROCESS PAGE USING 'adapter' WITH` ausgeführt.



Anmerkung: Zwischen einem `PROCESS PAGE MODAL`-Statement und seinem entsprechenden `END-PROCESS`-Statement kann kein sich auf Report 0 beziehendes `PRINT-`, `WRITE-`, `INPUT-` oder `DISPLAY`-Statement ausgeführt werden.

Beim Verlassen des `PROCESS PAGE MODAL`-Statementblocks werden folgende Aktionen ausgeführt:

- Wurde eine modale Seite für diese Ebene geöffnet, wird die modale Seite geschlossen;
- die Systemvariable `*PAGE-LEVEL` wird dekrementiert, und die Systemvariable `*PAGE-EVENT` wird auf den Wert zurückgesetzt, den sie hatte, bevor mit der Abarbeitung des Statement-Block begonnen wurde.

Syntax-Element-Beschreibung:

<i>statement</i>	Anstelle eines Statements müssen Sie in Abhängigkeit von der Situation eines oder mehrere passende Statements angeben. Wenn Sie kein spezifisches Statement angeben möchten, können Sie das <code>IGNORE</code> -Statement angeben.
END-PROCESS	Das für Natural reservierte Wort <code>END-PROCESS</code> muss benutzt werden, um das <code>PROCESS PAGE MODAL</code> -Statement zu beenden.

Beispiel:

```
* Name: First Demo/Open modal!
*
PROCESS PAGE USING "EMPTY-A"
*
/*( DEFINE EVENT HANDLER
DECIDE ON FIRST *PAGE-EVENT
  VALUE U'nat:page.end', U'onClose'
  /* Page closed.
  IGNORE
  VALUE U'onNextLevel'
  PROCESS PAGE MODAL
```

```
    FETCH RETURN "EMPTY-P"  
    END-PROCESS  
    PROCESS PAGE UPDATE  
    NONE VALUE  
    PROCESS PAGE UPDATE  
END-DECIDE  
/*) END-HANDLER  
END
```

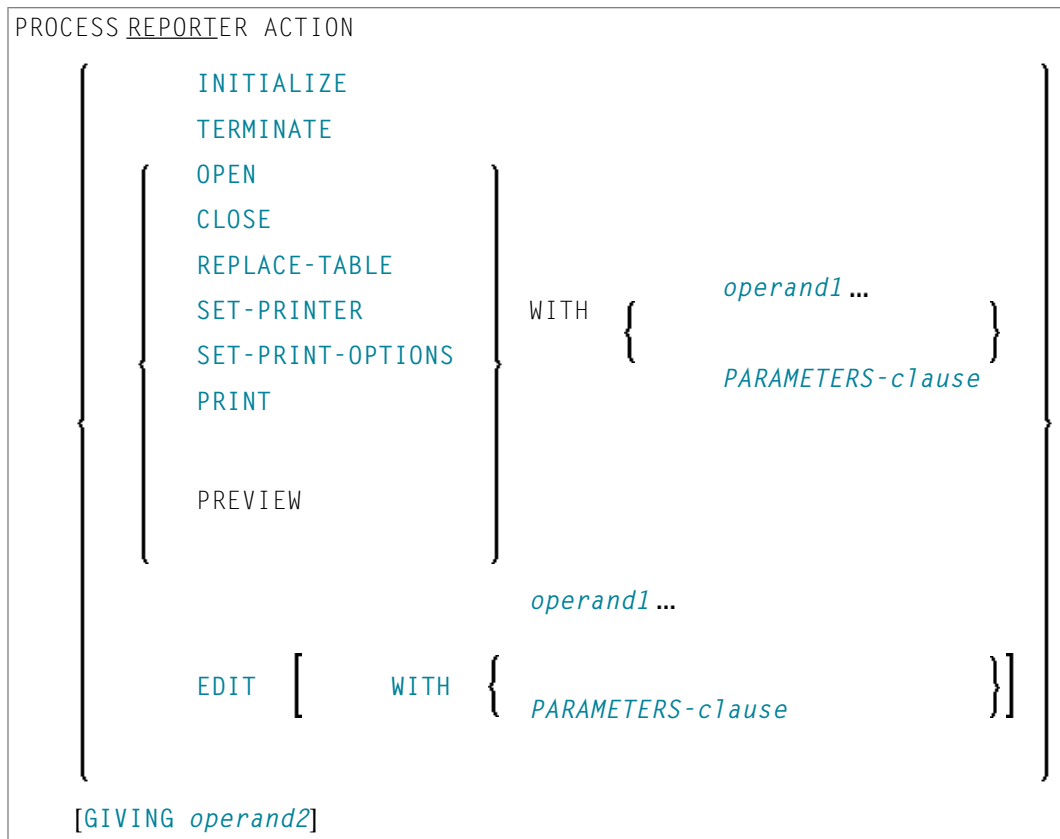
Beispiele

Weitere Beispiele zur Verwendung des PROCESS PAGE-Statements sind in der Library SYSEXNJX enthalten.

100

PROCESS REPORTER

▪ Funktion	658
▪ Syntax-Beschreibung	659
▪ Beispiele	663



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Das `PROCESS REPORTER`-Statement dient der Kommunikation mit dem Natural Reporter aus einem Programm heraus. Es weist den Reporter an, eine bestimmte Aktion auszuführen.

Eine Anleitung für die Benutzung des Reporters finden Sie in der Online-Hilfe, die mit Natural Reporter angeboten wird.



Anmerkung: Bei Aktionen, die sich auf einen bestimmten Report beziehen, können Sie das zweite Schlüsselwort auf `REPORT` verkürzen. Dies dient nur der Lesbarkeit Ihrer Programme; Natural unterscheidet nicht zwischen der ausgeschriebenen und abgekürzten Form des Schlüsselworts.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate												Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	C	S				A	N	P	I	F	B	D	T	L			ja	nein
<i>operand2</i>		S					N	P	I								ja	nein

Syntax-Element-Beschreibung:

ACTION	<p>Aktionen:</p> <p>Sie können eine der folgenden Aktionen angeben, die vom Reporter ausgeführt werden sollen.</p>
INITIALIZE	Diese Aktion initialisiert und lädt den Reporter. Dies muss immer die erste Aktion sein, die ausgeführt wird.
TERMINATE	Diese Aktion beendet und entlädt den Reporter. Dies muss immer die letzte Aktion sein, die ausgeführt wird.
OPEN	Diese Aktion öffnet einen angegebenen Report und gibt ein Handle zurück. Dieses Handle kann dazu benutzt werden, den Report bei nachfolgenden Aktionen zu identifizieren.
CLOSE	Diese Aktion schließt einen angegebenen Report. Danach kann das Report-Handle nicht mehr benutzt werden.
REPLACE-TABLE	Diese Aktion ersetzt den Pfadnamen einer Tabelle.
SET-PRINTER	Diese Aktion wählt einen Drucker aus, der zum Ausdrucken aller nachfolgenden Reports benutzt werden soll. Die Druckmethode für den ausgewählten Drucker muss in NATPARM auf "TTY" gesetzt sein.
SET-PRINT-OPTIONS	Diese Aktion setzt die Druckoptionen für einen angegebenen Report.
PRINT	Diese Aktion druckt einen angegebenen Report direkt auf dem ausgewählten Drucker.
PREVIEW	Diese Aktion bietet eine Vorschau des angegebenen Reports, basierend auf dem zur Zeit ausgewählten Drucker.
EDIT	Wenn kein Report angegeben ist, zeigt diese Aktion das Hauptfenster des Reporters. Wenn ein Report angegeben ist, zeigt diese Aktion das Hauptfenster des Reporters zusammen mit dem Editierfenster für den angegebenen Report.
WITH	<p>WITH-Klausel:</p> <p>Als <i>operand1</i> geben Sie die Parameter an, die an die Aktion übergeben werden sollen.</p>

<i>PARAMETERS-clause</i>	Als Alternative zur WITH-Klausel können Sie die unten beschriebene <i>PARAMETERS-Klausel</i> benutzen.
GIVING operand2	<p>GIVING-Klausel:</p> <p>Mit der GIVING-Klausel können Sie den Response-Code der aufgerufenen Aktion abfragen.</p> <p>Als <i>operand2</i> geben Sie das Feld an, das den Response-Code empfangen soll.</p> <p>Der Response-Code wird im Format/Länge I4 ausgegeben.</p> <p>Response-Code "0" bedeutet, dass die Aktion erfolgreich war. Jeder andere Response-Code entspricht einer Natural-Systemfehlernummer (NATnnnn).</p>

PARAMETERS -Klausel

```
PARAMETERS {parameter-name=operand1} ...
END-PARAMETERS
```

Mit dieser Klausel können Sie die Parameter auch mit dem Namen übergeben (statt mit der Position):

- Parameter für die OPEN-Aktion
- Parameter für die REPLACE-TABLE-Aktion
- Parameter für die SET-PRINTER-Aktion
- Parameter für die SET-PRINT-OPTIONS-Aktion
- Parameter für die CLOSE-, PRINT-, PREVIEW-, EDIT-Aktionen

Parameter für die OPEN-Aktion

Bei dieser Aktion geben Sie als ersten Parameter den Namen des zu öffnenden Reports an (ohne Erweiterung *.rpt* oder Pfadangabe). Als zweiten Parameter geben Sie das Feld an, das das Handle empfangen soll. Format/Länge des ersten Parameters muss mit A8 kompatibel sein. Format/Länge des zweiten Parameters muss mit I4 kompatibel sein.

Der Report wird zuerst im RES-Unterverzeichnis der Logon-Library gesucht, dann im RES-Unterverzeichnis jeder Steplib und dann im Verzeichnis, das mit der Umgebungsvariablen NATGUI_BMP definiert ist.

Die Reportdaten werden zuerst in dem Pfad gesucht, der bei der Erstellung des Reports angegeben wurde (wenn er existiert), und dann im Verzeichnis, in dem der Report gefunden wurde.

Wenn Sie die *PARAMETERS-Klausel* benutzen, muss *parameter-name* Folgendes enthalten:

- REPORT-NAME für den Reportnamen.
- REPORT-ID für das Handle-Feld.

Siehe auch [Beispiel 1 - Parameter für die OPEN-Aktion](#).

Parameter für die REPLACE-TABLE-Aktion

Bei dieser Aktion geben Sie als ersten Parameter das Handle an, das den Report identifiziert, auf den die Aktion angewandt werden soll. Als zweiten Parameter geben Sie die Arbeitsdateinummer an. Als dritten Parameter können Sie optional den Tabellennamen eingeben. Format/Länge der ersten beiden Parameter müssen mit I4 kompatibel sein. Format/Länge des dritten Parameters muss mit A8 kompatibel sein.

Wenn Sie die *PARAMETERS-Klausel* benutzen, muss *parameter-names* Folgendes enthalten: REPORT-ID, WORK-FILE beziehungsweise TABLE-NAME.

Siehe auch [Beispiel 2 - Parameter für die REPLACE-TABLE-Aktion](#).

Parameter für die SET-PRINTER-Aktion

Bei dieser Aktion geben Sie als *operand1* den logischen Gerätenamen (LPT1 bis LPT31) des ausgewählten Druckers an. Format/Länge von *operand1* muss mit A8 kompatibel sein.

Wenn Sie die *PARAMETERS-Klausel* benutzen, muss *parameter-name* Folgendes enthalten: DEVICE-NAME.

Siehe auch [Beispiel 3 - Parameter für die SET-PRINTER-Aktion](#).

Parameter für die SET-PRINT-OPTIONS-Aktion

Bei dieser Aktion geben Sie als ersten Parameter (Nummer 1 in der Tabelle unten) das Handle an, das den Report identifiziert, auf den die Aktion angewandt werden soll. Danach folgenden die zu setzenden Druckeroptionen, die alle optional sind. Wenn ein Parameter weggelassen wird, wird die entsprechende Option nicht geändert.

Sequenznummer	Parameter
1	Dieser Parameter (der mit Format/Länge I4 kompatibel sein muss) ist das Handle, das den Report identifiziert, auf den die Aktion angewandt werden soll. <i>parameter-name</i> muss REPORT-ID sein. *
2	Dieser Parameter (der mit Format/Länge I2 kompatibel sein muss) ist eine der Konstanten für die Papiergröße, die in der Local Data Area NGULKEY1 definiert ist. Die möglichen Werte sind: <ul style="list-style-type: none"> ■ CUSTOM-PAPER (explizite Papierbreite und -höhe verwenden) ■ LETTER (8.5 x 11 Zoll) ■ LEGAL (8.5 x 14 Zoll) ■ EXECUTIVE (7.25 x 10.5 Zoll)

Sequenznummer	Parameter
	<ul style="list-style-type: none"> ■ A4 (210 x 297 mm) ■ COM-10-ENVELOPE (4.125 x 9.5 Zoll) ■ DL-ENVELOPE (110 x 220 mm) ■ C5-ENVELOPE (162 x 229 mm) ■ B5-ENVELOPE (176 x 250 mm) ■ MONARCH-ENVELOPE (3.875 x 7.5 Zoll) <p><i>parameter-name</i> muss PAPER-SIZE sein. *</p>
3 und 4	<p>Diese Parameter (die mit Format/Länge I2 kompatibel sein müssen) sind die Papierbreite und -höhe (in Twips; 1 Twip = 1/1440 Zoll). Diese Parameter werden nur mit der Papiergröße CUSTOM-PAPER benutzt.</p> <p>Wenn Sie die <i>PARAMETERS-Klausel</i> benutzen, muss <i>parameter-names</i> Folgendes enthalten: PAPER-WIDTH und PAPER-HEIGHT.</p>
5, 6, 7 und 8	<p>Diese Parameter (die mit Format/Länge I2 kompatibel sein müssen) geben den linken, oberen, rechten und unteren Rand an (in Twips).</p> <p><i>parameter-names</i> muss Folgendes enthalten: LEFT-MARGIN, TOP-MARGIN, RIGHT-MARGIN, BOTTOM-MARGIN.</p>
9	<p>Dieser Parameter (der im Format L sein muss) gibt die Papierausrichtung an:</p> <ul style="list-style-type: none"> ■ TRUE = Querformat ■ FALSE = Hochformat <p>Dieser Parameter wird nicht mit der Papiergröße CUSTOM-PAPER benutzt.</p> <p><i>parameter-name</i> muss LANDSCAPE sein. *</p>
10	<p>Dieser Parameter (der im Format L sein muss) entspricht der Druckoption für schnelles Drucken (nur Text):</p> <ul style="list-style-type: none"> ■ TRUE = Grafiken unterdrücken ■ FALSE = nicht unterdrücken <p><i>parameter-name</i> muss FAST-PRINT sein. *</p>
11	<p>Dieser Parameter (der im Format L sein muss) bestimmt ob Datensätze, die nur aus Leerzeichen bestehen, bei der Ausgabe unterdrückt werden sollen:</p> <ul style="list-style-type: none"> ■ TRUE = unterdrücken ■ FALSE = nicht unterdrücken <p><i>parameter-name</i> muss SUPPRESS- BLANK-LINES sein. *</p>
12	<p>Dieser Parameter (der im Format L sein muss) bestimmt ob aufeinanderfolgende Datensätze mit identischen Daten ignoriert werden sollen:</p> <ul style="list-style-type: none"> ■ TRUE = ignorieren

Sequenznummer	Parameter
	<ul style="list-style-type: none"> ■ FALSE = nicht ignorieren <p><i>parameter-name</i> muss IGNORE-DUPLICATES sein. *</p>
13	<p>Dieser Parameter (der im Format L sein muss) bestimmt ob beim Drucken ein Dialog für die Druckerauswahl angezeigt werden soll:</p> <ul style="list-style-type: none"> ■ TRUE = anzeigen ■ FALSE = nicht anzeigen <p>The <i>parameter-name</i> must be SHOW-PRINT-DIALOG. *</p>
14	<p>Dieser Parameter (der mit Format/Länge I2 kompatibel sein muss) ist einer der Konstanten für die Papierquelle, die in der Local Data Area NGULKEY1 definiert ist. Die möglichen Werte sind:</p> <ul style="list-style-type: none"> ■ AUTOMATIC = automatische Papierzufuhr ■ MANUAL = manuelle Papierzufuhr <p><i>parameter-name</i> muss PAPER-SOURCE sein. *</p>



Anmerkung: * Wenn Sie die *PARAMETERS-Klausel* benutzen.

Siehe auch [Beispiel 4 - Parameter für die SET-PRINT-OPTIONS-Aktion](#).

Parameter für die CLOSE-, PRINT-, PREVIEW-, EDIT-Aktionen

Bei diesen Aktionen geben Sie als *operand1* das Handle an, das den Report identifiziert, auf den die Aktion angewandt werden soll. Format/Länge von *operand1* muss mit I4 kompatibel sein.

Wenn Sie die *PARAMETERS-Klausel* benutzen, muss REPORT-ID der *parameter-name* ein.

Siehe auch [Beispiel 5 - Parameter für die CLOSE-, PRINT-, PREVIEW-, EDIT-Aktionen](#).

Beispiele

- [Beispiel 1 - Parameter für die OPEN-Aktion](#)
- [Beispiel 2 - Parameter für die REPLACE-TABLE-Aktion](#)
- [Beispiel 3 - Parameter für die SET-PRINTER-Aktion](#)
- [Beispiel 4 - Parameter für die SET-PRINT-OPTIONS-Aktion](#)

- [Beispiel 5 - Parameter für die CLOSE-, PRINT-, PREVIEW-, EDIT-Aktionen](#)

Beispiel 1 - Parameter für die OPEN-Aktion

```
PROCESS REPORT ACTION OPEN WITH 'MYREPORT' #HANDLE
```

```
PROCESS REPORT ACTION OPEN WITH  
PARAMETERS  
  REPORT-NAME = 'MYREPORT'  
  REPORT-ID   = #HANDLE  
END-PARAMETERS
```

Beispiel 2 - Parameter für die REPLACE-TABLE-Aktion

```
PROCESS REPORT ACTION REPLACE-TABLE WITH  
PARAMETERS  
  REPORT-ID = #HANDLE  
  WORK-FILE = 5  
END-PARAMETERS
```

Beispiel 3 - Parameter für die SET-PRINTER-Aktion

```
PROCESS REPORTER ACTION SET-PRINTER WITH 'LPT1'
```

Beispiel 4 - Parameter für die SET-PRINT-OPTIONS-Aktion

```
DEFINE DATA LOCAL  
  USING 'NGLUKEY1'  
END-DEFINE  
...  
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH #HANDLE  
  A4 0 0 0 0 0 0 FALSE FALSE FALSE FALSE FALSE AUTOMATIC
```

```
DEFINE DATA LOCAL  
  USING 'NGLUKEY1'  
END-DEFINE  
...  
PROCESS REPORT ACTION SET-PRINT-OPTIONS WITH PARAMETERS  
  REPORT-ID = #HANDLE  
  PAPER-SIZE = A4  
  PAPER-WIDTH = 0
```

```
PAPER-HEIGHT = 0
LEFT-MARGIN = 0   TOP-MARGIN = 0
RIGHT-MARGIN = 0  BOTTOM-MARGIN = 0
LANDSCAPE = FALSE
FAST-PRINT = FALSE
SUPPRESS-BLANK-LINES = FALSE
IGNORE-DUPPLICATES = FALSE
SHOW-PRINT-DIALOG = FALSE
PAPER-SOURCE = AUTOMATIC
END-PARAMETERS
```

Beispiel 5 - Parameter für die CLOSE-, PRINT-, PREVIEW-, EDIT-Aktionen

```
PROCESS REPORT ACTION PRINT WITH #HANDLE
PROCESS REPORT ACTION PREVIEW WITH #HANDLE
PROCESS REPORT ACTION CLOSE WITH #HANDLE
PROCESS REPORT ACTION EDIT WITH #HANDLE
PROCESS REPORTER ACTION EDIT
```


101 PROPERTY

▪ Funktion	668
▪ Syntax-Beschreibung	668
▪ Beispiel	669

```
PROPERTY property-name
  OF [INTERFACE] interface-name
  IS operand
END-PROPERTY
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [CREATE OBJECT](#) | [DEFINE CLASS](#) | [INTERFACE](#) | [METHOD](#) | [SEND METHOD](#)

Gehört zur Funktionsgruppe: [Komponentenbasierte Programmierung](#)

Funktion

Das `PROPERTY`-Statement weist einen Objektdatenvariablen-Operanden als Implementierung für eine Property zu, und zwar außerhalb einer Interface-Definition.

Es wird verwendet, wenn die betreffende Interface-Definition von einem Copycode übernommen wird und auf eine klassenspezifische Art und Weise implementiert werden soll.

Es kann nur innerhalb des `DEFINE CLASS`-Statements und nach den Interface-Definitionen verwendet werden.

Die angegebenen Interface- und Property-Namen müssen in den Interface-Definitionen des `DEFINE CLASS`-Statements definiert sein.

Syntax-Beschreibung

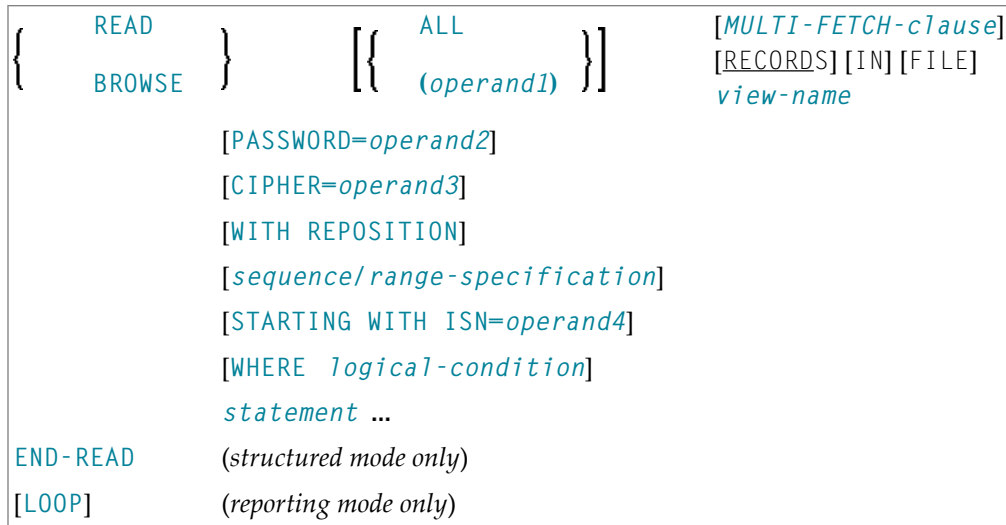
<code><i>property-name</i></code>	Dies ist der der Property zugewiesene Name.
<code>OF <i>interface-name</i></code>	Dies ist der dem Interface zugewiesene Name.
<code>IS <i>operand</i></code>	Der <i>operand</i> in der IS-Klausel weist eine Objektdaten-Variable als Platz zum Speichern des Property-Wertes zu.
<code>END-PROPERTY</code>	Das für Natural reservierte Wort <code>END-PROPERTY</code> muss zum Beenden des <code>PROPERTY</code> -Statements benutzt werden.

Beispiel

Das in der Dokumentation des `METHOD`-Statements enthaltene Beispiel zeigt, wie dasselbe Interface in zwei Klassen unterschiedlich implementiert wird, und wie das `PROPERTY`-Statement und das `METHOD`-Statement zu diesem Zweck benutzt werden.

102 READ

▪ Funktion	672
▪ Syntax-Beschreibung	673
▪ Bei READ verfügbare Systemvariablen	682
▪ Beispiele	683



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | GET TRANSACTION DATA | DELETE | END TRANSACTION | FIND | HISTOGRAM | GET | GET SAME | LIMIT | PASSW | PERFORM BREAK PROCESSING | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement READ dient dazu, Datensätze von der Datenbank zu lesen. Die Datensätze können in physischer Reihenfolge, in der Reihenfolge der Adabas-ISNs oder in der Reihenfolge der Werte eines Deskriptorfeldes gelesen werden.

Das READ-Statement initiiert eine Verarbeitungsschleife.

Siehe auch Abschnitt *READ-Statement im Leitfaden zur Programmierung*.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate												Referenzierung erlaubt	Dynam. Definition											
<i>operand1</i>	C	S							N	P	I	B*														ja	nein
<i>operand2</i>	C	S					A																			ja	nein
<i>operand3</i>	C	S							N																	ja	nein
<i>operand4</i>	C	S							N	P	I	B*														ja	nein

* Format B von *operand1* and *operand4* kann mit einer Länge von kleiner gleich 4 benutzt werden.


Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Anzahl der zu lesenden Datensätze:</p> <p>Sie können die Anzahl der Datensätze, die mit dem READ-Statement gelesen werden sollen, durch Angabe von <i>operand1</i> (in Klammern hinter dem Schlüsselwort READ) in Form einer numerischen Konstanten (0 bis 4294967295) oder über eine Benutzervariable (in Klammern hinter dem Schlüsselwort READ) begrenzen. Zum Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 10px;"> READ (5) IN EMPLOYEES ... MOVE 10 TO CNT(N2) READ (CNT) EMPLOYEES ... </pre> <p>Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.</p> <p>Ist mit dem Profil-/Session-Parameter LT ein kleineres Limit gesetzt, so gilt das LT-Limit.</p> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn Sie eine vierstellige Anzahl von Sätzen lesen möchten, geben Sie diese mit einer vorangestellten Null an: (0nnnn); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement. 2. <i>operand1</i> wird zu Beginn des ersten READ-Schleifendurchlaufs ausgewertet. Wird der Wert von <i>operand1</i> innerhalb der
-----------------	--


	<p>READ-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Datensätze.</p>
ALL	<p>Um hervorzuheben, dass <i>alle</i> Datensätze gelesen werden sollen, können Sie optional das Schlüsselwort ALL angeben.</p>
<i>MULTI-FETCH-clause</i>	<p>Siehe MULTI-FETCH-Klausel.</p>
<i>view-name</i>	<p>View-Name:</p> <p>Als <i>view-name</i> geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Statement oder in einer programmexternen Global oder Local Data Area definiert ist.</p> <p>Im Reporting Mode ist <i>view-name</i> der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.</p>
PASSWORD CIPHER	<p>PASSWORD- und CIPHER-Klauseln</p> <p>Diese Klauseln gelten nur für Zugriffe auf Adabas-Datenbanken. Mit Entire System Server können sie nicht verwendet werden.</p> <p>Die PASSWORD-Klausel dient dazu, ein Passwort anzugeben, um auf Daten einer passwortgeschützten Datei zugreifen zu können.</p> <p>Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Daten in entschlüsselter Form zu erhalten.</p> <p>Weitere Informationen hierzu siehe Statements FIND und PASSW.</p>
WITH REPOSITION	<p>Diese Option macht das READ-Statement empfänglich für Repositionierungsereignisse. Siehe WITH REPOSITION Option.</p>
<i>sequence/range-specification</i>	<p>Diese Option gibt Lese-Reihenfolge und -umfang an. Siehe Lesereihenfolge und -umfang.</p>
STARTING WITH ISN=operand4	<p>Diese Klausel gilt nur für Adabas-Datenbanken.</p> <p>Zugriff auf Adabas</p> <p>Diese Klausel kann in Verbindung mit einem READ-Statement in physischer oder logischer Reihenfolge (aufsteigend/absteigend) verwendet werden. Der angegebene Wert (<i>operand4</i>) steht für eine Adabas ISN und wird verwendet, um einen bestimmten Datensatz anzugeben, ab dem die READ-Leseschleife gestartet werden soll.</p> <p>Logische Reihenfolge</p> <p>Auch wenn das READ-Statement mit einem Gleichheitszeichen (=) versehen ist, gibt es nicht nur diejenigen Datensätze mit genau dem Startwert in dem betreffenden Deskriptorfeld zurück, sondern startet ab dem angegebenen Startwert einen logischen Suchlauf in aufsteigender oder absteigender Reihenfolge. Wenn einige Datensätze</p>

	<p>im Deskriptorfeld denselben Inhalt haben, werden sie in der Reihenfolge der ISNs sortiert zurückgegeben.</p> <p>Die Klausel <code>STARTING WITH ISN</code> ist so was wie ein „Selektionskriterium der zweiten Stufe“, das nur gilt, wenn der Startwert mit dem Deskriptorwert für den ersten Datensatz übereinstimmt.</p> <p>Alle Datensätze mit einem Deskriptorwert, der mit dem Startwert identisch ist, und mit einer ISN, die <i>kleiner gleich</i> (<i>größer gleich</i> für ein absteigendes <code>READ</code>) der Start-ISN ist, werden von Adabas ignoriert. Der erste in der <code>READ</code>-Schleife zurückgegebene Datensatz ist entweder</p> <ul style="list-style-type: none"> ■ der erste Datensatz mit Deskriptor = Startwert und einer ISN <i>größer</i> (<i>kleiner</i> für ein absteigendes <code>READ</code>) als die Start-ISN ■ oder wenn ein solcher Datensatz nicht vorhanden ist, der erste Datensatz mit einem Deskriptor <i>größer</i> (<i>kleiner</i> für ein absteigendes <code>READ</code>) als der Startwert. <p>Physische Reihenfolge</p> <p>Die Datensätze werden in der Reihenfolge zurückgegeben, in der sie physisch gespeichert sind. Wenn eine <code>STARTING WITH ISN</code>-Klausel angegeben wird, ignoriert Adabas alle Datensätze, bis der Datensatz mit der ISN, die mit der Start-ISN identisch ist, erreicht ist. Der erste zurückgegebene Datensatz ist der nächste auf den Datensatz mit der Start-ISN folgende Datensatz.</p> <p>Verwendung</p> <p>Diese Klausel kann zum Repositionieren innerhalb einer <code>READ</code>-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden, um auf einfache Weise den nächsten Datensatz zu bestimmen, mit dem die Verarbeitung fortgesetzt werden soll. Dies ist besonders hilfreich, wenn der nächste Datensatz sich nicht eindeutig durch einen seiner Deskriptorwerte ermitteln lässt.</p> <p>Die Klausel kann auch hilfreich sein in einer verteilten Client/Server-Anwendung, in der das Lesen der Datensätze von einem Server-Programm und die weitere Verarbeitung der Datensätze von einem Client-Programm durchgeführt werden, wobei die Datensätze nicht alle auf einmal, sondern stapelweise verarbeitet werden.</p> <p>Beispiel: Siehe Programm <code>REASISND</code> weiter unten.</p>
WHERE <i>logical-condition</i>	Siehe <i>WHERE-Klausel</i> .
END-READ	Das für Natural reservierte Schlüsselwort <code>END-READ</code> muss zum Beenden des <code>READ</code> -Statements benutzt werden.

MULTI-FETCH Clause


 **Anmerkung:** Diese Klausel kann nur bei Adabas-Datenbanken benutzt werden.

<pre>[MULTI-FETCH { ON OFF OF <i>multi-fetch-factor</i> }]</pre>
--

 **Anmerkung:** [MULTI-FETCH OF *multi-fetch-factor*] wird bei den Datenbanktypen ADA und ADA2 nicht ausgewertet. Es erfolgt die Standardverarbeitung (siehe Profilparameter MFSET. Beim Datenbanktyp ADA2 wird die MULTI-FETCH-Klausel komplett ignoriert; siehe *Database Management System Assignments* in der *Configuration Utility*-Dokumentation.

Ausführliche Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung*.


WITH REPOSITION-Option

 **Anmerkung:** Diese Option ist nur beim Zugriff auf Adabas-Datenbanken möglich.

Mit dieser Option können Sie innerhalb der aktiven READ-Schleife auf einen anderen Startwert für die zu lesenden Datensätze repositionieren. Die Verarbeitung des READ-Statements wird dann unter Verwendung des neuen Startwerts fortgesetzt.

Die Repositionierung kann auf zwei verschiedene Arten ausgelöst werden, wenn Sie ein READ-Statement in Zusammenhang mit der WITH REPOSITION-Option verwenden:

1. Wenn ein **ESCAPE TOP REPOSITION**-Statement ausgeführt wird, verzweigt Natural direkt zum Schleifenanfang und führt einen Neustart aus; d.h. dass die Datenbank im Einklang mit dem aktuellen Inhalt der Suchwert-Variable auf einen neuen Datensatz in der Datei repositioniert. Gleichzeitig wird der Schleifenzähler *COUNTER auf Null (0) zurückgesetzt.
2. Wenn eine READ-Schleife versucht, den nächsten Datensatz aus der Datenbank aufzurufen, und der Wert der Systemvariable *COUNTER Null (0) ist.

 **Anmerkung:** Wenn *COUNTER innerhalb der aktiven READ-Schleife auf Null gesetzt wird, wird die Verarbeitung des aktuellen Datensatzes fortgesetzt; es erfolgt keine sofortige Verzweigung zum Schleifenanfang. Sie können eine solche Repositionierung nicht auf diese Weise bei Natural für Windows, UNIX und OpenVMS auslösen. Diese Funktionalität wurde nur aus Kompatibilitätsgründen mit Natural Version 3.1 für Großrechner beibehalten. Aus diesem Grund wird nicht empfohlen, diesen Prozess zu verwenden.

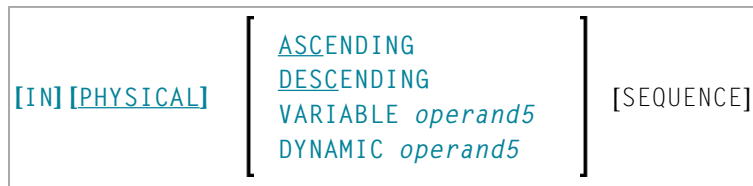
Funktionstechnische Überlegungen

- Wenn das READ-Statement ein Schleifen-Limit hat (z.B. READ (10) EMPLOYEES WITH REPOSITION ..) und ein Neustart-Event ausgelöst wurde, arbeitet die Schleife 10 neue Datensätze ab, egal wieviele Datensätze bereits abgearbeitet worden sind, bis die Repositionierung erfolgt ist.
- Wenn ein ESCAPE TOP REPOSITION-Statement ausgeführt wird, die innerste Schleife aber keine Repositionierung ausführen kann (da das Schlüsselwort WITH REPOSITION nicht im READ-Statement gesetzt ist, oder es sich beim abgesetzten Schleifen-Statement nicht um ein READ handelt), wird ein entsprechender Laufzeitfehler ausgegeben.
- Da das ESCAPE TOP-Statement keine Referenzen erlaubt, können Sie nur einen Repositionierungs-Event initiieren, wenn die innerste Verarbeitungsschleife ein READ .. WITH REPOSITION-Statement ist.
- Ein Repositionierungs-Event löst weder die Ausführung des AT START OF DATA-Programmabschnittes aus, noch löst er die erneute Verarbeitung des Schleifenlimit-Operanden aus (wenn es sich um eine Variable handelt).
- Wenn der Suchwert nicht geändert wurde, repositioniert die Schleife auf denselben Datensatz wie beim ursprünglichen Schleifenanfang.

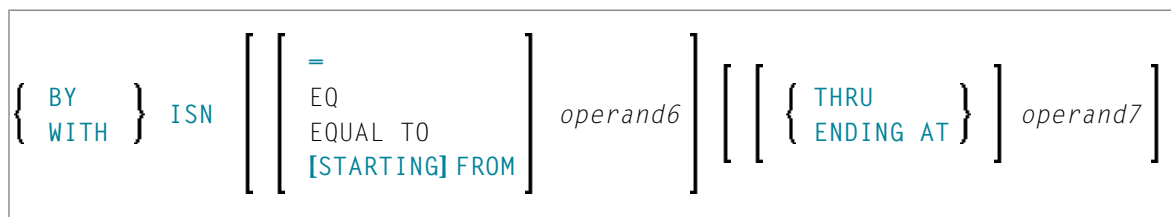
Lesereihenfolge und -umfang (sequence/range-specification)

Die folgenden Syntax-Optionen sind verfügbar, um Lese-Reihenfolge und/oder -Bereich anzugeben.

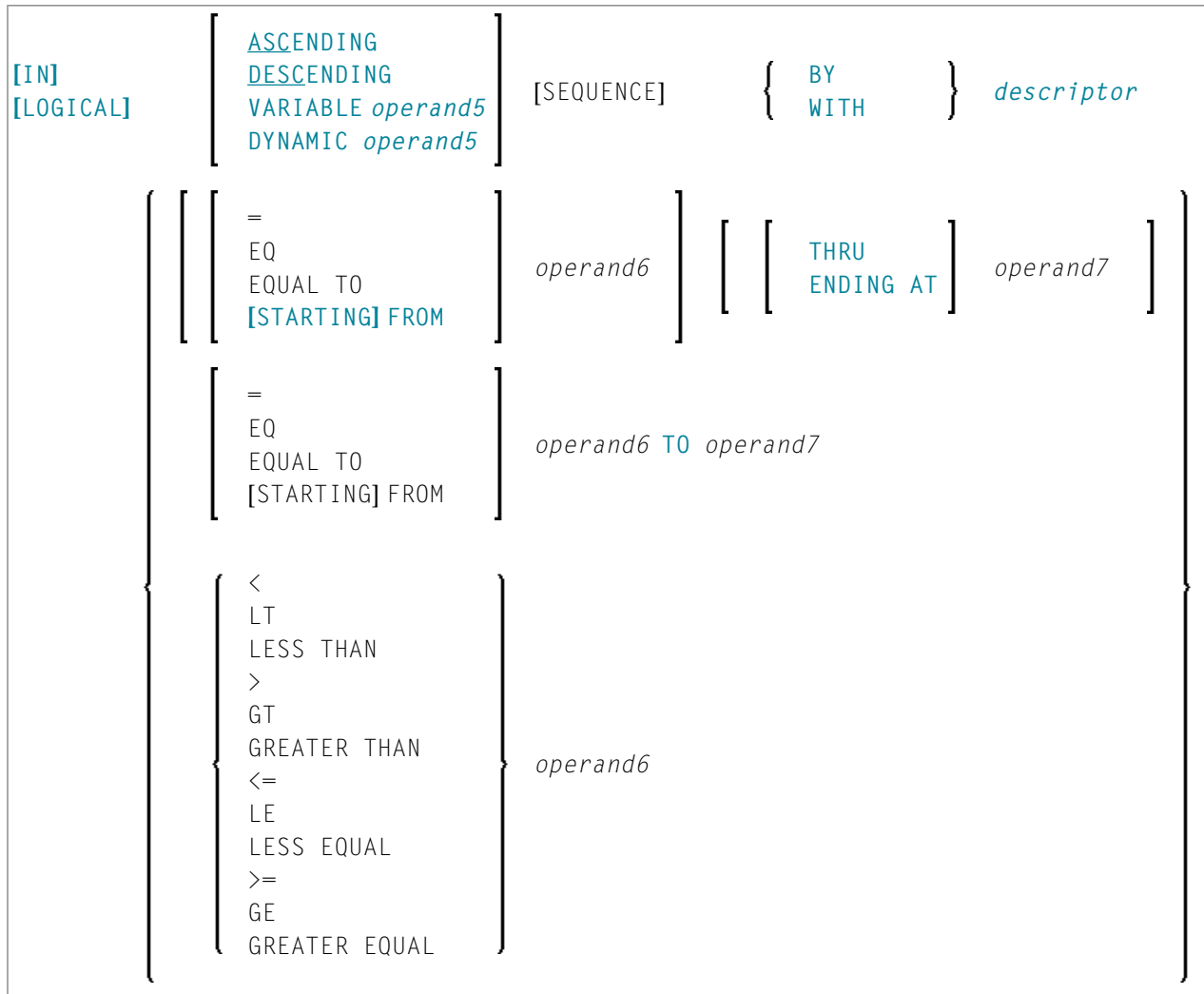
Syntax-Option 1:



Syntax-Option 2:



Syntax-Option 3:



Anmerkungen:

1. Die Syntax-Optionen 2 und 3 sind in Verbindung mit Entire System Server nicht verfügbar.
2. Im Diagramm zu Syntax-Option 3 finden Sie Vergleichsoperanden, die ab Natural Version 4 für Großrechner verwendet werden können. Wenn diese Vergleichsoperanden zum Einsatz kommen, dürfen die Optionen ENDING AT, THRU und TO nicht benutzt werden. Diese Vergleichsoperanden gelten auch beim HISTOGRAM-Statement.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition	
<i>operand5</i>		S			A									ja	nein
<i>operand6</i>	C	S			A	N	P	I	F	B*	D	T	L	ja	nein
<i>operand7</i>	C	S			A	N	P	I	F	B*	D	T	L	ja	nein

* Format B von *operand6* und *operand7* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

READ IN PHYSICAL SEQUENCE	<p>PHYSICAL SEQUENCE bedeutet, dass die Datensätze in der physischen Reihenfolge, in der sie auf der Datenbank gespeichert sind, gelesen werden.</p> <p>Ist nichts anderes angegeben, gilt standardmäßig IN PHYSICAL SEQUENCE.</p>
READ BY ISN	<p>READ BY ISN bedeutet, dass die Datensätze in der Reihenfolge der Adabas-ISNs (Interne Satznummern) gelesen werden. Anstelle des Schlüsselworts BY können Sie, mit gleichem Ergebnis, auch das Schlüsselwort WITH verwenden.</p> <p>READ BY ISN ist nur bei Adabas-Datenbanken möglich.</p> <p>Anmerkung: Bei XML-Datenbanken (z.B. Tamino) dient READ BY ISN zum Lesen von XML-Objekten entsprechend der Reihenfolge der Objekt-IDs.</p>
READ IN LOGICAL SEQUENCE	<p>LOGICAL SEQUENCE bedeutet, dass die Datensätze in der Reihenfolge der Werte eines bestimmten Deskriptorfeldes (Key) gelesen werden.</p> <p>Wenn Sie ein Deskriptorfeld angeben, werden die Datensätze in der Wertabfolge dieses Feldes gelesen. Als Feld können Sie einen Deskriptor, Subdeskriptor, Superdeskriptor oder Hyperdeskriptor verwenden, nicht aber einen phonetischen Deskriptor, einen Deskriptor innerhalb einer Periodengruppe oder einen Superdeskriptor, der ein Periodengruppenfeld enthält.</p> <p>Wenn Sie kein Deskriptorfeld angeben, wird der im verwendeten DDM eingetragene Standarddeskriptor (Feld Default Sequence) genommen.</p> <p>Informationen zu READ IN LOGICAL SEQUENCE finden Sie auch im <i>Leitfaden zur Programmierung</i>; siehe <i>Statements für Datenbankzugriffe, READ-Statement</i>.</p>
ASCENDING DESCENDING VARIABLE DYNAMIC SEQUENCE	<p>Diese Klausel gilt nur für Adabas-, XML- und SQL-Datenbanken. Bei einem READ PHYSICAL-Statement gilt sie nur für DB2-Datenbanken.</p> <p>Mit dieser Klausel können Sie bestimmen, ob die Datensätze in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen.</p> <ul style="list-style-type: none"> ■ Standardmäßig werden die Datensätze in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort ASCENDING auch ausdrücklich angeben können, aber nicht müssen).

	<ul style="list-style-type: none"> ■ Wenn die Datensätze in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort <code>DESCENDING</code> an. ■ Wenn erst zur Laufzeit bestimmt werden soll, ob die Datensätze in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort <code>VARIABLE</code> oder <code>DYNAMIC</code> gefolgt von einer Variablen (<i>operand5</i>) an. Der Wert von <i>operand5</i> zu Beginn der <code>READ</code>-Verarbeitungsschleife bestimmt dann die Reihenfolge. <i>operand5</i> muss Format/Länge <code>A1</code> haben und kann den Wert <code>A</code> (für Ascending/aufsteigend) oder <code>D</code> (für Descending/absteigend) enthalten. ■ Wenn das Schlüsselwort <code>VARIABLE</code> benutzt wird, wird die Leserichtung (Wert von <i>operand5</i>) am Anfang der <code>READ</code>-Verarbeitungsschleife ausgewertet, und sie bleibt bestehen, bis die Schleife beendet wird, ganz gleich ob das Feld von <i>operand5</i> in der <code>READ</code>-Schleife geändert wird oder nicht. ■ Wenn das Schlüsselwort <code>DYNAMIC</code> benutzt wird, wird die Leserichtung (Wert von <i>operand5</i>) vor jedem Aufruf eines Datensatzes in der <code>READ</code>-Verarbeitungsschleife ausgewertet und kann von Datensatz zu Datensatz geändert werden. Dies ermöglicht es überall in der <code>READ</code>-Schleife, die Reihenfolge beim Durchblättern von aufsteigend in absteigend (und umgekehrt) zu ändern. <p>Anmerkung:</p> <p>1. Bei XML-Datenbanken steht <code>DYNAMIC SEQUENCE</code> nicht zur Verfügung.</p>
<p>STARTING FROM ... ENDING AT/TO</p>	<p>Mit den Klauseln <code>STARTING FROM</code> und <code>ENDING AT</code> können Sie angeben, ab welchem Wert und bis zu welchem Wert des Deskriptorfeldes gelesen werden soll.</p> <p>Die <code>STARTING FROM</code>-Klausel (= oder <code>EQ</code> oder <code>EQUAL TO</code> oder <code>[STARTING] FROM</code>) legt den Startwert für die <code>READ</code>-Operation fest. Wenn ein Startwert angegeben wird, beginnt das Lesen mit dem angegebenen Wert. Wenn der Startwert in der Datei nicht vorhanden ist, wird der nächsthöhere (oder niedrigere bei einem absteigenden <code>READ</code>) Wert benutzt. Wenn kein höherer (oder niedrigerer für absteigendes <code>READ</code>) Wert vorhanden ist, wird die Schleife nicht durchlaufen.</p> <p>Um die Datensätze auf einen Endwert zu begrenzen, können Sie eine <code>ENDING AT</code>-Klausel mit den Bedingungen <code>THRU</code>, <code>ENDING AT</code> oder <code>TO</code> angeben, wobei dann bis einschließlich der angegebenen Werte gelesen wird. Immer wenn das <code>READ</code>-Deskriptorfeld den angegebenen Endwert überschreitet, wird die Schleife automatisch beendet. Obwohl die Basis-Funktionalität der Schlüsselwörter <code>TO</code>, <code>THRU</code> und <code>ENDING AT</code> sich jeweils ähnelt, so unterscheiden sie sich doch im Detail.</p>
<p>THRU/ENDING AT</p>	<p>Wenn <code>THRU</code> oder <code>ENDING AT</code> benutzt wird, wird nur der Startwert an die Datenbank übergeben, aber die Endwerte-Prüfung vom Natural-Laufzeitsystem durchgeführt, nachdem der Datensatz von der Datenbank zurückgegeben wird. Wenn die Lese-Richtung <code>ASCENDING</code> (aufsteigend) ist, müssen Sie den niedrigeren Wert als den Startwert und den höheren Wert als den Endwert angeben, da der Startwert den zuerst in der <code>READ</code>-Schleife zurückgegebenen Wert (und den Datensatz) darstellt. Wenn Sie aber einen rückwärtsgerichteten (<code>DESCENDING</code>) Lesevorgang starten, muss der höhere Wert im Startwert und der niedrigere Wert im Endwert erscheinen.</p>

	<p>Um das Ende des zu lesenden Wertebereichs zu bestimmen, liest Natural intern einen Datensatz über den Endwert hinaus ein. Wenn Sie die READ-Schleife verlassen haben, weil der Endwert erreicht war, denken Sie bitte daran, dass dieser letzte Datensatz wirklich nicht der letzte Datensatz innerhalb des erforderlichen Bereiches ist, sondern der erste Datensatz jenseits dieses Bereiches (außer in dem Fall, dass die Datei keinen weiteren Datensatz nach dem letzten Ergebnisdatensatz enthält).</p> <p>THRU / ENDING AT kann für alle Datenbanken benutzt werden, die die READ- oder HISTOGRAM-Statements unterstützen.</p>
TO	<p>Wenn das Schlüsselwort TO verwendet wird, werden sowohl der Startwert als auch der Endwert an die Datenbank übergeben, und Natural führt keine Prüfungen auf Wertebereiche hin aus. Wenn der Endwert überschritten wird, reagiert die Datenbank genauso wie wenn das Dateiende (End-of-File) erreicht wäre, und die Datenbank-Schleife wird verlassen. Da alle Bereichsprüfungen von der Datenbank vorgenommen werden, wird immer der niedrigere Wert (des Bereiches) im Startwert und der höhere Wert im Endwert angegeben, ungeachtet der Tatsache, ob Sie in aufsteigender (ASCENDING) oder absteigender (DESCENDING) Reihenfolge lesen.</p> <p>Die TO-Option steht nur zur Verfügung, wenn die zugrundeliegende Datenbank Adabas Version 3.1.1 auf UNIX, OpenVMS oder Windows, Adabas Version 7 (oder höher) auf Großrechnern, Tamino oder eine SQL-Datenbank ist.</p>

Anmerkungen zu Funktionsunterschieden zwischen THRU/ENDING AT und TO

Die folgende Liste beschreibt die Funktionsunterschiede zwischen der Benutzung der Option THRU/ENDING AT und der Option TO.

THRU/ENDING AT	TO
Wenn die READ-Schleife beendet wird, weil der Endwert erreicht worden ist, enthält der View den ersten Datensatz, der außerhalb des Bereiches ist (Out-of-Range).	Wenn die READ-Schleife beendet wird, weil der Endwert erreicht worden ist, enthält der View den letzten Datensatz des angegebenen Bereiches.
Wenn eine Endwert-Variable im Verlauf einer READ-Schleife geändert wird, wird beim nächsten gelesenen Datensatz der neue Wert für eine Endwerte-Prüfung benutzt.	Die Endwert-Variable wird erst beim READ-Schleifenstart verarbeitet. Alle weiteren Änderungen im Verlauf der READ-Schleife haben keine Auswirkung.
Ein falsch angegebener Bereich (z.B. READ .. ='B' THRU 'A') führt nicht zu einem Datenbank-Fehler, sondern dazu, dass einfach kein Datensatz zurückgegeben wird.	Ein falsch angegebener Bereich führt zu einem Datenbank-Fehler (z.B. Adabas RC=61), weil ein Wertebereich nicht in absteigender Reihenfolge angegeben werden darf.
Wenn ein READ .. DESCENDING mit Start- und Endwert benutzt wird, wird der Startwert zur Positionierung in der Datei benutzt, wohingegen der Endwert von Natural zum Abprüfen auf das Bereichsende (End-of-Range) hin benutzt wird. Deshalb ist der Startwert höher als der oder gleich dem Endwert.	Da beide Werte an die Datenbank übergeben werden, müssen sie in aufsteigender Reihenfolge erscheinen. Mit anderen Worten, der Startwert ist niedriger als der oder gleich dem Endwert, egal ob in aufsteigender oder in absteigender Reihenfolge gelesen wird.

THRU/ENDING AT	TO
Um auf einen Bereichsüberlauf hin abzuprüfen, muss der Deskriptorwert in dem zugrunde liegenden Datenbank-View erscheinen, d.h. er muss im Satzpuffer zurückgegeben werden.	Der Deskriptor ist für die zurückgegebenen Satzfelder nicht erforderlich.
Endwerte-Prüfungen auf Adabas-Mehrwertfelder (MU-Feld) oder Sub-/Super-/Hyperdeskriptoren hin ist nicht möglich und führt zum Syntaxfehler NAT0160 bei der Kompilierung des Programms.	Sie können einen Endwert für MU-Felder und Sub-/Super-/Hyperdeskriptoren angeben.
Kann für alle Datenbanken benutzt werden.	Kann nur für Adabas Version 3.1.1 auf Windows, UNIX oder OpenVMS, Adabas Version 7 (oder höher) auf Großrechnern, Tamino oder einer SQL-Datenbank benutzt werden.

WHERE-Klausel

```
WHERE logical-condition
```

Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (*logical-condition*) angeben. Diese wird ausgewertet, *nachdem* ein Wert gelesen wurde, aber *bevor* eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich **AT BREAK**-Verarbeitung) erfolgt.

Näheres zu logischen Bedingungen finden Sie unter *Logische Bedingungen* im *Leitfaden zur Programmierung*.

Ist über ein **LIMIT**-Statement oder eine Limit-Notation die Anzahl der zu lesenden Datensätze begrenzt, so werden bei einem **READ**-Statement, das eine WHERE-Klausel enthält, Datensätze, die aufgrund der WHERE-Bedingung nicht weiterverarbeitet werden, bei der Ermittlung des Limits nicht mitgezählt.

Bei READ verfügbare Systemvariablen

Die Natural-Systemvariablen *ISN und *COUNTER stehen mit dem **READ**-Statement zur Verfügung. Format/Länge dieser Systemvariablen ist P10.

Format und Länge können nicht geändert werden.

Die Systemvariablen werden folgendermaßen verwendet:

*ISN	Die Systemvariable *ISN enthält die Adabas-ISN des gerade verarbeiteten Datensatzes. Anmerkung: 1. Bei Tamino enthält *ISN die XML Objekt-ID. 2. Bei SQL-Datenbanken oder mit Entire System Server ist *ISN nicht verfügbar.
*COUNTER	Die Systemvariable *COUNTER enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen wurde.

Beispiele

- Beispiel 1 — READ-Statement
- Beispiel 2 — READ-Statement mit WITH REPOSITION
- Beispiel 3 — READ- und FIND-Statements miteinander kombiniert
- Beispiel 4 — READ-Statement mit DESCENDING-Option
- Beispiel 5 — READ-Statement mit VARIABLE-Option
- Beispiel 6 — READ-Statement mit DYNAMIC-Option
- Beispiel 7 — READ-Statement mit STARTING WITH ISN-Klausel

Beispiel 1 — READ-Statement

```

** Example 'REAEX1S': READ (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 3
*
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOY-VIEW IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOY-VIEW BY ISN STARTING FROM 1 ENDING AT 3
  DISPLAY          PERSONNEL-ID NAME *ISN *COUNTER

```

READ

```
END-READ
*
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOY-VIEW BY NAME
  DISPLAY      PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'''
READ EMPLOY-VIEW BY NAME STARTING FROM 'M'
  DISPLAY      PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
END
```

Ausgabe des Programms REAEX1S:

PERSONNEL ID	NAME	ISN	CNT

READ IN PHYSICAL SEQUENCE			
50005800	ADAM	1	1
50005600	MORENO	2	2
50005500	BLOND	3	3
READ IN ISN SEQUENCE			
50005800	ADAM	1	1
50005600	MORENO	2	2
50005500	BLOND	3	3
READ IN NAME SEQUENCE			
60008339	ABELLAN	478	1
30000231	ACHIESON	878	2
50005800	ADAM	1	3
READ IN NAME SEQUENCE STARTING FROM 'M'			
30008125	MACDONALD	923	1
20028700	MACKARNESS	765	2
40000045	MADSEN	508	3

Äquivalentes Reporting-Mode-Beispiel: [REAEX1R](#).

Beispiel 2 — READ-Statement mit WITH REPOSITION

```

DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR      (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
INPUT (IP=OFF AD=0) 'NAME:' NAME /
  'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
  'Press PF3 to stop'
IF *PF-KEY = 'PF3'
  THEN STOP
END-IF
IF #ATTR MODIFIED
  THEN ESCAPE TOP REPOSITION
END-IF
END-READ
...

```

```

DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR      (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
INPUT (IP=OFF AD=0) 'NAME:' NAME /
  'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
  'Press PF3 to stop'
IF *PF-KEY = 'PF3'
  THEN STOP
END-IF
IF #ATTR MODIFIED
  THEN RESET *COUNTER
END-IF
END-READ
...

```

Beispiel 3 — READ- und FIND-Statements miteinander kombiniert

Das folgende Beispiel verwendet zunächst ein READ-Statement, um Datensätze von der Datei EMPLOYEES (Mitarbeiter) in logischer Reihenfolge der Werte des Deskriptorfeldes NAME zu lesen. Dann werden mit einem FIND-Statement Datensätze von der Datei VEHICLES (Fahrzeuge) gelesen, wobei die Personalnummer (PERSONNEL-ID) der EMPLOYEES-Datei als Suchkriterium benutzt wird.

Der erzeugte Report zeigt den Namen und die Personalnummer aller von der EMPLOYEES-Datei gelesenen Personen sowie die Fabrikate (MAKE) der Autos (von der VEHICLES-Datei), die im Besitz dieser Personen sind. Besitzt eine Person mehrere Autos, werden entsprechend mehrere Zeilen pro Person ausgegeben.

```

** Example 'REAEX2': READ and FIND combination
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 10
*
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  SUSPEND IDENTICAL SUPPRESS
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      ENTER
    END-NOREC
  DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
    PERSONNEL-ID (RD.)
    FIRST-NAME (RD.)
    MAKE (FD.) (IS=OFF)

  END-FIND
END-READ
END

```

Ausgabe des Programms REAEX2:

PERSONNEL ID	FIRST-NAME	MAKE
20007500	VIRGINIA	CHRYSLER
20008400	MARSHA	CHRYSLER
		CHRYSLER
20021100	ROBERT	GENERAL MOTORS
20000800	LILLY	FORD
		MG
20001100	EDWARD	GENERAL MOTORS
20002000	MARTHA	GENERAL MOTORS
20003400	LAUREL	GENERAL MOTORS
30034045	KEVIN	DATSUN
30034233	GREGORY	FORD
11400319	MANFRED	

Beispiel 4 — READ-Statement mit DESCENDING-Option

```

** Example 'READSCND': READ (with DESCENDING SEQUENCE)
*****
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
END-DEFINE
*
READ (10) EMPL IN DESCENDING SEQUENCE BY NAME FROM 'ZZZ'
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
END-READ
END

```

Beispiel 5 — READ-Statement mit VARIABLE-Option

```

** Example 'REAVSEQ': READ (with VARIABLE SEQUENCE)
*****
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #DIR          (A1)
1 #STARTVALUE (A20)
END-DEFINE
*

```

```

SET KEY PF7 PF8
*
INPUT 'Select READ direction'
  // 'Press' 08T 'PF7' (I)                21T 'to read backward'
  /      08T 'PF8' (I) 'or' 'ENTER' (I) 21T 'to read forward'
*
IF *PF-KEY = 'PF7'
  MOVE 'D' TO #DIR
  MOVE 'ZZZ' TO #STARTVALUE
ELSE
  MOVE 'A' TO #DIR
  MOVE 'A' TO #STARTVALUE
END-IF
*
READ (10) EMPL IN VARIABLE #DIR SEQUENCE
          BY NAME FROM #STARTVALUE
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
END-READ
END

```

Beispiel 6 — READ-Statement mit DYNAMIC-Option

```

DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'> /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
2 NAME
...
END-DEFINE
...
READ #EMPVIEW IN DYNAMIC #DIRECTION SEQUENCE BY NAME = 'SMITH'
  INPUT (AD=0) NAME
    / 'Press PF7 to scroll in DESCENDING sequence'
    / 'Press PF8 to scroll in ASCENDING sequence'
  ..
  IF *PF-KEY = 'PF7' THEN MOVE 'D' TO #DIRECTION END-IF
  IF *PF-KEY = 'PF8' THEN MOVE 'A' TO #DIRECTION END-IF
END-READ
...

```

Beispiel 7 — READ-Statement mit STARTING WITH ISN-Klausel

```

** Example 'REASISND': READ (with STARTING WITH ISN)
*****
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #DIR      (A1)
1 #STARTVAL (A20)
1 #STARTISN (N8)
END-DEFINE
*
SET KEY PF3 PF7 PF8
*
MOVE 'ADKINSON' TO #STARTVAL
*
READ (9) EMPL BY NAME = #STARTVAL
  WRITE *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD) *COUNTER
  IF *COUNTER = 5 THEN
    MOVE NAME TO #STARTVAL
    MOVE *ISN TO #STARTISN
  END-IF
END-READ
*
#DIR := 'A'
*
REPEAT
  READ EMPL IN VARIABLE #DIR BY NAME = #STARTVAL
    STARTING WITH ISN = #STARTISN
  MOVE NAME TO #STARTVAL
  MOVE *ISN TO #STARTISN
  INPUT NO ERASE (IP=OFF AD=0)
  15/01 *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
  // 'Direction:' #DIR
  // 'Press PF3 to stop'
  / ' PF7 to go step back'
  / ' PF8 to go step forward'
  / ' ENTER to continue in that direction'
/*
IF *PF-KEY = 'PF7' AND #DIR = 'A'
  MOVE 'D' TO #DIR
  ESCAPE BOTTOM
END-IF
IF *PF-KEY = 'PF8' AND #DIR = 'D'
  MOVE 'A' TO #DIR
  ESCAPE BOTTOM
END-IF

```

READ

```
    IF *PF-KEY = 'PF3'  
        STOP  
    END-IF  
END-READ  
/*  
IF *COUNTER(0290) = 0  
    STOP  
END-IF  
END-REPEAT  
END
```

103

READ WORK FILE

▪ Funktion	692
▪ Feldlängen	697
▪ Verarbeitung dynamischer Variablen	697
▪ Syntax-Beschreibung	694
▪ Feldlängen	697
▪ Verarbeitung dynamischer Variablen	697
▪ Beispiel	698

Structured Mode-Syntax

```

READ WORK [FILE] work-file-number [ONCE]
  {
    RECORD operand1
    [AND] [SELECT] { [ OFFSET n ] ... operand2 } ... }
  [GIVING LENGTH operand3]
  [ AT [END] [OF] [FILE]
    statement ...
    END-ENDFILE ]
END-WORK
  
```

Reporting Mode-Syntax

```

READ WORK [FILE] work-file-number [ONCE]
  {
    RECORD {operand1 [FILLER nX]} ...
    [AND] [SELECT] { [ OFFSET n ] ... operand2 } ... }
  [GIVING LENGTH operand3]
  [ AT [END] [OF] [FILE] { statement
    DO statement ... DOEND } ]
[LOOP]
  
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `CLOSE WORK FILE` | `DEFINE WORK FILE` | `WRITE WORK FILE`

Gehört zur Funktionsgruppe: *Verarbeitung von Arbeitsdateien/PC-Dateien*

Funktion

Das Statement `READ WORK FILE` dient dazu, Daten von einer physisch-sequentiellen Nicht-Adabas-Arbeitsdatei zu lesen. Die Daten werden sequentiell von der Arbeitsdatei gelesen. Wie sie gelesen werden, ist unabhängig davon, wie Sie auf die Arbeitsdatei geschrieben wurden.

Das `READ WORK FILE`-Statement führt eine Verarbeitungsschleife aus, um alle Datensätze der Arbeitsdatei zu lesen. Innerhalb einer `READ WORK FILE`-Schleife können automatische Gruppenwechsel-Verarbeitungen durchgeführt werden.

Informationen zu Unicode- und Codepage-Support siehe *Work Files and Print Files on Windows, UNIX and OpenVMS Platforms* in der *Unicode and Code Page Support*-Dokumentation.



Anmerkungen:

1. Wenn bei Ausführung eines READ WORK FILE-Statements eine End-of-File-Bedingung auftritt, schließt Natural die Arbeitsdatei automatisch.
2. Bei Entire Connection: Beim Lesen von Entire Connection-Arbeitsdateien darf innerhalb der READ WORK FILE-Verarbeitungsschleife kein I/O-Statement stehen.

Wenn eine ASCII-Arbeitsdatei gelesen wird, kann es passieren, dass ein leerer Datensatz als letzter Datensatz nach dem letzten physischen Datensatz zurückgegeben wird. Dies geschieht, weil Natural keine individuellen Datensätze liest, sondern größere Blöcke der Arbeitsdatei, um die Performanz beim Datei-Zugriff zu optimieren.

Feldlängen

Die Länge der Felder in der **Operanden-Definitionstabelle** wird wie folgt bestimmt:

Format	Länge
A, B, I, F	Die Anzahl der Bytes im Eingabedatensatz entspricht der internen Längendefinition.
N	Die Anzahl der Bytes im Eingabedatensatz ergibt sich aus der Summe der internen Stellen vor und nach dem Komma (Dezimalpunkt). Komma und Vorzeichen belegen im Eingabedatensatz kein Byte.
P, D, T	Die Anzahl der Bytes im Eingabedatensatz ergibt sich aus der Summe der Stellen vor und nach dem Komma (Dezimalpunkt) plus einer Stelle für das Vorzeichen, geteilt durch 2 und aufgerundet.
L	1 Byte wird benutzt. Bei Feldern des Formats C werden 2 Bytes benutzt.

Beispiele für Feldlängen:

Felddefinition	Eingabedatensatz
#FIELD1 (A10)	10 Bytes
#FIELD2 (B15)	15 Bytes
#FIELD3 (N1.3)	4 Bytes
#FIELD4 (N0.7)	7 Bytes
#FIELD5 (P1.2)	2 Bytes
#FIELD6 (P6.0)	4 Bytes

Siehe auch *Format und Länge von Benutzervariablen im Leitfaden zur Programmierung*.

Verarbeitung dynamischer Variablen

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate											Referenzierung		
																erlaubt	Dynam. Definition	
<i>operand1</i>	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	ja	ja
<i>operand2</i>	S	A	G		A	U	N	P	I	F	B	D	T	L	C		ja	ja
<i>operand3</i>	S								I								ja	ja

Bei dem Arbeitsdateityp ENTIRECONNECTION darf *operand2* nicht im Format C sein.

Siehe auch [Feldlängen](#).

Syntax-Element-Beschreibung:

<i>work-file-number</i>	<p>Arbeitsdateinummer:</p> <p>Die für Natural definierte Nummer der Arbeitsdatei, die gelesen werden soll.</p> <p>Variabler Indexbereich:</p> <p>Wenn Sie ein Array von einer Arbeitsdatei lesen, können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel:</p> <pre>READ WORK FILE <i>work-file-number</i> #ARRAY (I:J)</pre>
ONCE	<p>ONCE-Option:</p> <p>Die ONCE-Option bewirkt, dass nur ein Datensatz gelesen und keine Verarbeitungsschleife initiiert wird (das schleifenbeendende Schlüsselwort END-WORK bzw. LOOP darf daher nicht zusammen mit ONCE verwendet werden). Wenn Sie ONCE verwenden, sollten Sie auch eine AT END OF FILE-Klausel verwenden.</p> <p>Wird ein READ WORK FILE-Statement mit ONCE-Option von einer benutzerinitiierten Verarbeitungsschleife gesteuert, kann es sein, dass auf der Arbeitsdatei eine End-of-File-Bedingung auftritt, bevor die Schleife beendet wird. Alle von der Arbeitsdatei gelesenen Felder enthalten dann immer noch die Werte des zuletzt gelesenen Datensatzes. Die Arbeitsdatei wird dann zum ersten Datensatz</p>

	zurückpositioniert, welcher dann bei der nächsten Ausführung des READ WORK FILE ONCE-Statements gelesen wird.				
RECORD <i>operand1</i> FILLER <i>nX</i>	<p>RECORD-Option:</p> <p>Wird RECORD angegeben, so werden alle Felder jedes gelesenen Datensatzes zur Verarbeitung zur Verfügung gestellt. Entsprechend dem Aufbau des Datensatzes muss eine Operandenliste (<i>operand1</i>) angegeben werden.</p> <p>Mit FILLER <i>nX</i> geben Sie an, dass beim Eingabedatensatz <i>n</i> Bytes übersprungen werden sollen. Der in der RECORD-Klausel definierte Datensatz muss in einem zusammenhängenden Speicherbereich gespeichert sein.</p> <p>Im Structured Mode ist FILLER nicht erlaubt. Im Structured Mode – oder wenn <i>operand1</i> in einem DEFINE DATA-Statement definiert ist – darf <i>operand1</i> nur einmal angegeben werden. In beiden Fällen darf FILLER nicht verwendet werden.</p> <p>Natural überprüft die Daten des Datensatzes nicht. Demzufolge ist diese Option die schnellste Art, Datensätze von einer sequentiellen Datei zu verarbeiten. Allerdings müssen Sie selbst darauf achten, dass der Aufbau des Datensatzes korrekt beschrieben ist, da es sonst durch nicht-numerische Daten in numerischen Feldern zu einem ungewollten Programmabbruch kommen kann. Bevor der Datensatz gelesen wird, füllt Natural die Operandenliste (<i>operand1</i>) mit Leerzeichen; demzufolge wird bei einer End-of-File-Bedingung ein leerer Bereich zurückgegeben. Kurze Datensätze werden mit Leerzeichen aufgefüllt.</p> <p>Siehe Übersicht für die Benutzung der RECORD-Option weiter unten.</p>				
SELECT	<p>SELECT-Option (Voreinstellung):</p> <p>Wird die SELECT-Option verwendet, so werden nur die in der Operandenliste (<i>operand2</i>) angegebenen Felder zur Verfügung gestellt. Die Position der Felder im Eingabedatensatz kann durch eine OFFSET- und/oder FILLER-Angabe spezifiziert werden.</p> <table border="1"> <tr> <td>OFFSET <i>n</i></td> <td>OFFSET 0 spezifiziert das erste Byte des Datensatzes.</td> </tr> <tr> <td>FILLER <i>nX</i></td> <td>Bedeutet, dass <i>n</i> Bytes des Eingabedatensatzes übersprungen werden.</td> </tr> </table> <p>Natural ordnet den einzelnen Feldern die ausgewählten Werte zu und überprüft, dass die vom Datensatz ausgewählten numerischen Felder entsprechend ihrer Definition auch wirklich gültige numerische Werte enthalten. Aufgrund dieser Prüfung dauert die Verarbeitung einer sequentiellen Datei mit der SELECT-Option etwas länger als mit der RECORD-Option.</p> <p>Wenn ein Datensatz nicht alle in der SELECT-Option angegebenen Felder füllt, gilt folgendes:</p> <ul style="list-style-type: none"> ■ Ein Feld, das nur teilweise gefüllt wurde, wird mit Leerzeichen bzw. Nullen aufgefüllt. ■ Ein Feld, das gar nicht gefüllt wurde, behält denselben Inhalt wie zuvor. 	OFFSET <i>n</i>	OFFSET 0 spezifiziert das erste Byte des Datensatzes.	FILLER <i>nX</i>	Bedeutet, dass <i>n</i> Bytes des Eingabedatensatzes übersprungen werden.
OFFSET <i>n</i>	OFFSET 0 spezifiziert das erste Byte des Datensatzes.				
FILLER <i>nX</i>	Bedeutet, dass <i>n</i> Bytes des Eingabedatensatzes übersprungen werden.				

	Beim Lesen des Datei-Typs CSV werden die Optionen <code>OFFSET</code> und <code>FILLER</code> ignoriert.
GIVING LENGTH <i>operand3</i>	<p>GIVING LENGTH-Klausel</p> <p>Mit dieser Klausel können Sie die tatsächliche Länge des gelesenen Datensatzes in Erfahrung bringen. Die Länge (Anzahl der Bytes) erhalten Sie in <i>operand3</i>.</p> <p><i>operand3</i> muss mit <code>Format/Länge I4</code> definiert sein.</p> <p>Wenn die Arbeitsdatei als <code>TYPE UNFORMATTED</code> definiert ist, verweist die zurückgegebene Länge auf die Anzahl der aus dem Byte-Strom gelesenen Bytes, einschließlich der beim <code>FILLER</code>-Operanden übersprungenen Bytes.</p> <p>Wenn die <code>GIVING LENGTH</code>-Klausel mit dem Arbeitsdateityp CSV benutzt wird, gibt der mit <code>GIVING LENGTH</code> angegebene Operand die Anzahl der Felder im Datensatz (und nicht die Länge des Datensatzes) an.</p>
AT END OF FILE	<p>AT END OF FILE-Klausel</p> <p>Diese Klausel kann nur zusammen mit der ONCE-Option verwendet werden. Wenn Sie die <code>ONCE</code>-Option verwenden, dann sollten Sie auch eine <code>AT END OF FILE</code>-Klausel angeben, um festzulegen, was beim Auftreten einer End-of-File-Bedingung geschehen soll. Wenn Sie die <code>ONCE</code>-Option nicht verwenden, wird eine End-of-File-Bedingung wie das normale Ende einer Verarbeitungsschleife behandelt.</p>
END-WORK	Das für Natural reservierte Wort <code>END-WORK</code> muss zum Beenden des <code>READ WORK FILE</code> -Statements benutzt werden.

Übersicht für die Benutzung der RECORD-Option

RECORD-Option wird benutzt mit	Zur Kompilierzeit zurückgewiesen	Zur Laufzeit zurückgewiesen	RECORD-Option wird ignoriert, Verarbeitung schaltet um in SELECT-Modus
Arbeitsdateityp <code>ENTIRE CONNECTION</code>		x	
Dynamische Variablen	x		
Arbeitsdateityp <code>CSV</code>			x
Arbeitsdateityp <code>PORTABLE</code>			x
Arbeitsdateitypen <code>ASCII</code> , <code>ASCII COMPRESSED</code> , <code>CSV</code> , <code>UNFORMATTED</code> , Codepage ist im Configuration Utility angegeben (Konvertierung ist erforderlich) oder zumindest ein Unicode-Feld ist angegeben (Operand mit <code>Format U</code> , Konvertierung ist erforderlich)			x

Feldlängen

Die Länge der Felder in der **Operanden-Definitionstabelle** wird wie folgt bestimmt:

Format	Länge
A, B, I, F	Die Anzahl der Bytes im Eingabedatensatz entspricht der internen Längendefinition.
N	Die Anzahl der Bytes im Eingabedatensatz ergibt sich aus der Summe der internen Stellen vor und nach dem Komma (Dezimalpunkt). Komma und Vorzeichen belegen im Eingabedatensatz kein Byte.
P, D, T	Die Anzahl der Bytes im Eingabedatensatz ergibt sich aus der Summe der Stellen vor und nach dem Komma (Dezimalpunkt) plus einer Stelle für das Vorzeichen, geteilt durch 2 und aufgerundet.
L	1 Byte wird benutzt. Bei Feldern des Formats C werden 2 Bytes benutzt.

Beispiele für Feldlängen:

Felddefinition	Eingabedatensatz
#FIELD1 (A10)	10 Bytes
#FIELD2 (B15)	15 Bytes
#FIELD3 (N1.3)	4 Bytes
#FIELD4 (N0.7)	7 Bytes
#FIELD5 (P1.2)	2 Bytes
#FIELD6 (P6.0)	4 Bytes

Siehe auch *Format und Länge von Benutzervariablen im Leitfaden zur Programmierung*.

Verarbeitung dynamischer Variablen

Arbeitsdateityp	Verarbeitung
ASCII ASCII-COMPRESSED SAG (binär)	Die Arbeitsdateitypen ASCII, ASCII-COMPRESSED und SAG (binär) können keine dynamischen Variablen verarbeiten und rufen einen Fehler hervor. Sie können jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 32766 Bytes verarbeiten.
ENTIRECONNECTION	Der Arbeitsdateityp ENTIRECONNECTION kann keine dynamische Variablen verarbeiten. Er kann jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 107341824 Bytes verarbeiten. Die RECORD-Option ist nicht erlaubt, wenn dynamische Variablen benutzt werden.

Arbeitsdateityp	Verarbeitung
PORTABLE UNFORMATTED	Große und dynamische Variablen können mit den beiden Arbeitsdateitypen PORTABLE und UNFORMATTED in Arbeitsdateien geschrieben oder aus Arbeitsdateien gelesen werden. Bei diesen Typen gibt es keine Größenbeschränkung für dynamische Variablen. Große Variablen dürfen jedoch eine maximale Feld-/Datensatzlänge von 32766 Bytes nicht überschreiten. Wenn eine dynamische Variable aus einer PORTABLE-Arbeitsdatei gelesen wird, so führt dies zu einer Größenänderung auf die gespeicherte Länge.
CSV	Die maximale Feld-/Datensatzlänge für dynamische und große Variablen ist 32766 Bytes. Dynamische Variables werden unterstützt. X-Arrays sind nicht erlaubt und resultieren in einer Fehlermeldung.

Beispiel

```

** Example 'RWFEX1': READ WORK FILE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
*
1 #RECORD
  2 #PERS-ID (A8)
  2 #NAME (A20)
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  WRITE WORK FILE 1
    PERSONNEL-ID NAME
END-FIND
*
* ...
*
READ WORK FILE 1 RECORD #RECORD
  DISPLAY NOTITLE #PERS-ID #NAME
END-WORK
*
END

```

Ausgabe des Programms RWFEX1:

#PERS-ID	#NAME
11100328	BERGHAUS
11100329	BARTHEL
11300313	AECKERLE
11300316	KANTE
11500304	KLUGE
11500308	DIETRICH
11500318	GASSNER
11500343	ROEHM
11600303	BERGER
11600320	BLAETTEL
11500336	JASPER
11100330	BUSH
11500328	EGGERT

104 REDEFINE

▪ Funktion	702
▪ Einschränkung	702
▪ Syntax-Beschreibung	702
▪ Beispiele	703

```
REDEFINE { operand1 ( { nX
                    { operand2 }... ) } ... }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Das Statement `REDEFINE` dient dazu, ein Feld zu redefinieren. Das Ergebnis der Neudefinition können eine oder mehrere Benutzervariablen sein.

Mit einem `REDEFINE`-Statement können Sie gleichzeitig mehrere Felder redefinieren.

Einschränkung

Dieses Statement gilt nur im Reporting Mode. Um ein Feld im Structured Mode zu redefinieren, verwenden Sie die `REDEFINE`-Klausel des `DEFINE DATA`-Statements.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G	A U N P I F B D T L C	ja	nein
<i>operand2</i>	S A G	A N P I F B D T L C	ja	ja

Syntax-Element-Beschreibung:

REDEFINE	<p>Methode der Redefinition:</p> <p>Die Byte-Positionen von <i>operand1</i> werden unabhängig vom Format von links nach rechts redefiniert.</p> <p>Das Format von <i>operand2</i> muss nicht mit dem von <i>operand1</i> identisch sein. Die Byte-Positionen des neudefinierten Feldes müssen zu den im Feld enthaltenen Daten passen; wird beispielsweise ein alphanumerisches Feld als numerisch redefiniert, und enthält es entsprechend der Format-Spezifikation keine numerischen Daten, so kann die Verwendung des Feldes einen Programmabbruch zur Folge haben.</p> <p>Weitere Redefinition:</p> <p>Ein mit einem REDEFINE-Statement neudefiniertes Feld kann mit einem weiteren REDEFINE-Statement nochmals redefiniert werden.</p>
<i>nX</i>	<p>Füllbyte-Notation:</p> <p>Mit der Notation <i>nX</i> können Sie in der redefinierten Variable oder dem nachgestellten Feld <i>n</i> Füllbytes definieren. Nachgestellte Füllbytes müssen nicht unbedingt angegeben werden.</p>

Beispiele

- [Beispiel 1 — REDEFINE-Statement](#)
- [Beispiel 2 — REDEFINE-Statement](#)
- [Beispiel 3 — REDEFINE-Statement](#)
- [Beispiel 4 — REDEFINE-Statement](#)

Beispiel 1 — REDEFINE-Statement

Die Benutzervariable #A (Format/Länge A10) enthält den Wert 123ABCDEFG.

```
REDEFINE #A (#A1(N3) #A2(A7))
```

#A1 erhält den Wert 123, #A2 den Wert ABCDEFG.

Beispiel 2 — REDEFINE-Statement

Die Benutzervariable #B (Format/Länge A10) enthält den hexadezimalen Wert 12345CC1C2C3C4C5C6C7.

```
REDEFINE #B (#B1(P4) #B2(A7))
```

#B1 erhält den hexadezimalen Wert 12345C, #B2 den hexadezimalen Wert C1C2C3C4C5C6C7.

```
REDEFINE #B (#BB1(B2)8X)) or REDEFINE #B(#BB1(B2))
```

#BB1 erhält den hexadezimalen Wert 1234.

Der Wert in #BB1 ist "1234" (im Hexadecimalformat).

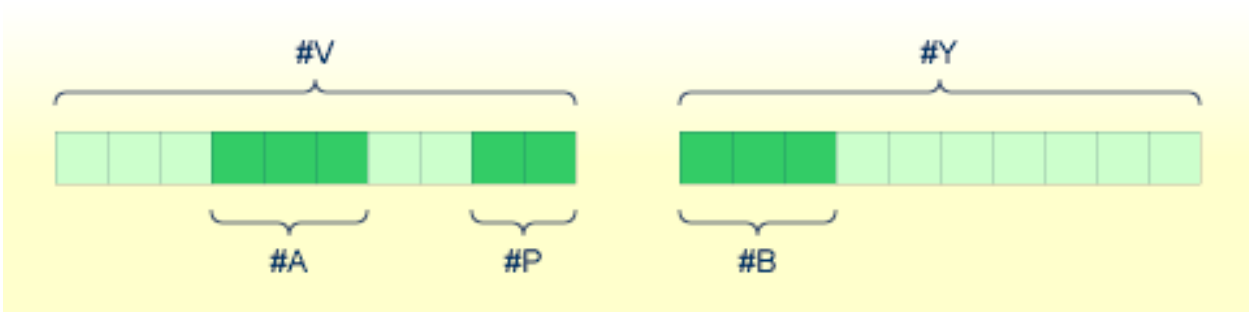


Anmerkung: Beim Format P (gepackt numerisch) muss die Anzahl der benötigten Dezimalstellen angegeben werden. Die Anzahl der Bytes, die eine gepackte Zahl benötigt, lässt sich wie folgt berechnen:

Anzahl der Bytes = (Anzahl der Dezimalstellen + 1) / 2, auf ganze Bytes aufgerundet.

Beispiel 3 — REDEFINE-Statement

```
COMPUTE #V (N8.2) = #Y (N10) = ...
REDEFINE #V (3X #A(N3) 2X #P (N2)) #Y (#B(N3) 7X)
```



Beispiel 4 — REDEFINE-Statement

In diesem Beispiel wird die Systemvariable *DATN, die die Form YYYYMMDD hat, redefiniert und das Ergebnis in der Reihenfolge Tag/Monat/Jahr (DAY/MONTH/YEAR) in drei getrennte Felder geschrieben:

```
MOVE *DATN TO #DATINT (N8)
REDEFINE #DATINT (#YEAR (N4) #MONTH (N2) #DAY (N2))
DISPLAY NOTITLE #DATINT #DAY #MONTH #YEAR
END
```

Ausgabe:

```
#DATINT  #DAY #MONTH #YEAR
-----  -
19950108   8   1     1995
```


105 REDUCE

▪ Funktion	708
▪ Syntax-Beschreibung	708

```
REDUCE { dynamic-clause } [GIVING operand5]
       { array-clause }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [EXPAND](#) | [RESIZE](#)

Gehört zur Funktionsgruppe: *Speicherverwaltungskontrolle für dynamische Variablen/X-Arrays*.

Funktion

Das Statement REDUCE dient zum Verringern

- der zugewiesenen Länge einer dynamischen Variable (*dynamic-clause*) oder
- der Anzahl der Ausprägungen von X-Arrays (*array-clause*).

Weitere Informationen entnehmen Sie den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- *Dynamische Variablen benutzen*
- *X-Arrays*
- *Speicherverwaltung von X-Gruppen-Arrays*

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>		S	A		A	U					B							nein	nein
<i>operand2</i>	C	S							I									nein	nein
<i>operand3</i>			A	G	A		N	P	I	F	B	D	T	L	C	G	O	ja	nein
<i>operand4</i>	C	S				U	N	P	I									nein	nein
<i>operand5</i>		S							I4									nein	ja

Syntax-Element-Beschreibung:

<i>dynamic-clause</i>	Das Statement REDUCE DYNAMIC VARIABLE dient dazu, die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variablen (<i>operand1</i>) auf die angegebene Länge (<i>operand2</i>) zu verringern. Weitere Informationen, siehe dynamic-clause weiter unten.
<i>operand1</i>	<i>operand1</i> ist die dynamische Variable, für die die zugewiesene Länge verringert werden soll.
<i>operand2</i>	<i>operand2</i> dient dazu, die verringerte Länge der dynamischen Variable anzugeben. Der angegebene Wert muss eine nicht negative, numerische Ganzzahl-Konstante oder eine Variable des Typs Integer4 (I4) sein.
<i>array-clause</i>	Mit dieser Klausel wird die Anzahl der Ausprägungen des X-Arrays (<i>operand3</i>) auf die mit (<i>dim[, dim[, dim]]</i>) angegebene Ober- und Untergrenze verringert. Weitere Informationen siehe array-clause weiter unten.
<i>operand3</i>	<i>operand3</i> ist das X-Array. Die Ausprägungen des X-Arrays können verringert werden. Die Index-Notation des Arrays ist optional. Als Index-Notation ist für jede Dimension nur die Stern-Notation (*) für den vollständigen Bereich zulässig.
<i>dim operand4</i>	Die Notation für die Ober- und Untergrenze (<i>operand4</i> oder Stern), auf die das X-Array verringert werden sollte, wird hier angegeben. Wenn der aktuelle Wert der Ober- oder Untergrenze verwendet werden sollte, kann ein Stern (*) anstatt <i>operand4</i> angegeben werden. Weitere Informationen siehe Dimension weiter unten.
GIVING <i>operand5</i>	Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die GIVING-Klausel angegeben wird, enthält <i>operand5</i> die Natural-Fehlernummer, wenn vorher ein Fehler aufgetreten ist, oder Null (0) bei Erfolg.

dynamic-clause

```
[SIZE OF] DYNAMIC [VARIABLE] operand 1 TO operand2
```

Das Statement REDUCE DYNAMIC VARIABLE dient dazu, die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variablen (*operand1*) auf die angegebene Länge (*operand2*) zu verringern. Liegt der zugewiesene Speicherplatz der dynamischen Variable oberhalb der angegebenen Länge (*operand2*), wird er sofort freigegeben, d.h. wenn das Statement ausgeführt wird.

Wenn die aktuell benutzte Länge (*LENGTH) der dynamischen Variablen größer als die gegebene Länge ist, wird *LENGTH auf die gegebene Größe gesetzt, und der Inhalt der Variable wird abgeschnitten (aber nicht geändert). Wenn die gegebene Länge die aktuell zugewiesene Länge der dynamischen Variable überschreitet, wird das Statement ignoriert.

array-clause

```
[OCCURRENCES OF] ARRAY operand3 TO { 0
                                         (dim[,dim[,dim]]) }
```

Mit dem Statement REDUCE ARRAY wird die Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit TO (*dim[,dim[,dim]]*) angegebene Ober- und Untergrenze verringert, wobei jedes *dim* eine mittels der im Folgenden beschriebenen Syntax definierte Dimension ist.

Wenn REDUCE TO 0 (Null) angegeben wird, werden alle Ausprägungen des X-Arrays freigegeben. Mit anderen Worten, das gesamte Array wird verringert.

Eine in einem REDUCE-Statement benutzte Ober- und Untergrenze muss genau mit der betreffenden, für das Array definierten Ober- und Untergrenze identisch sein.

Beispiel:

```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...

*/ reducing #a (1:10)

REDUCE ARRAY #a TO (1:10)          /* #a is reduced
REDUCE ARRAY #a TO (*:10)         /* to 10 occurrences.

*/ reducing #ga (1:10,1:20)

REDUCE ARRAY #g TO (1:10)         /* 1st dimension is set to (1:10)
REDUCE ARRAY #ga TO (*:*,1:20)   /* 1st dimension is dependent and
                                  /* therefore kept with (*:*)
                                  /* 2nd dimension is set to (1:20)

REDUCE ARRAY #a TO (5:10)        /* This is rejected because the lower index
                                  /* must be 1 or *
REDUCE ARRAY #a TO (#i:10)       /* This is rejected because the lower index
                                  /* must be 1 or *

REDUCE ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                                  /* because the dimension is dependent and
                                  /* must be specified with (*:*)
```

Weitere Informationen siehe:

- *Speicherverwaltung von X-Arrays*
- *Speicherverwaltung von X-Gruppen-Arrays*

Dimension

Jede in der Array-Klausel angegebene Dimension (*dim*) wird mittels der folgenden Syntax definiert:

$$\left\{ \begin{array}{c} * \\ \left\{ \begin{array}{c} * \\ \text{operand4} \end{array} \right\} : \left\{ \begin{array}{c} * \\ \text{operand4} \end{array} \right\} \end{array} \right\}$$

Die Notation für die Ober- und Untergrenze (*operand4* oder Stern), auf die das X-Array verringert werden sollte, wird hier angegeben. Wenn der aktuelle Wert der Ober- oder Untergrenze benutzt werden soll, kann ein Stern (*) anstelle von *operand4* angegeben werden. An Stelle von *: * können Sie auch einen einzelnen Stern angeben.

Die Anzahl der Dimensionen (*dim*) muss genau mit der definierten Anzahl der Dimensionen des X-Arrays (1, 2 oder 3) übereinstimmen.

Wenn Sie das REDUCE-Statement verwenden, ist es nur möglich, die Anzahl der Ausprägungen zu verringern. Wenn die erforderliche Anzahl größer ist als die aktuell zugewiesene Anzahl der Ausprägungen, wird dies einfach ignoriert.

106 REINPUT

▪ Funktion	714
▪ Syntax-Beschreibung	715
▪ Beispiele	721

REINPUT	[FULL] [(<i>statement-parameters</i>)]	{	USING HELP	}
	[<i>MARK-option</i>]		<i>WITH-TEXT-option</i>	
	[<i>ALARM-option</i>]			

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [DEFINE WINDOW](#) | [INPUT](#) | [SET WINDOW](#)

Gehört zur Funktionsgruppe: [Bildschirmgenerierung für interaktive Verarbeitung](#)

Funktion

Das Statement `REINPUT` dient dazu, zu einem `INPUT`-Statement zurückzukehren und dieses erneut auszuführen. Es wird in der Regel dazu benutzt, eine Fehlermeldung auszugeben, die dem Benutzer sagt, dass auf das `INPUT`-Statement hin ungültige Daten eingegeben wurden. Siehe [Beispiel 1](#).

Zwischen einem `INPUT`-Statement und dem dazugehörigen `REINPUT`-Statement werden keine `WRITE`- oder `DISPLAY`-Statements ausgeführt. Im Batch-Betrieb ist das `REINPUT`-Statement nicht gültig.

Wenn das `REINPUT`-Statement ausgeführt wird, setzt es den Programmstatus, was die Verarbeitung von Unterprogrammen, besonderen Bedingungen und Verarbeitungsschleifen anbelangt, wieder auf den Stand zurück, der galt, als das `INPUT`-Statement ausgeführt wurde (vorausgesetzt das `INPUT`-Statement ist nach wie vor aktiv). Wurde nach der Ausführung des `INPUT`-Statements eine Verarbeitungsschleife gestartet und das `REINPUT`-Statement befindet sich innerhalb dieser Schleife, so wird die Schleife abgebrochen und erst dann neu gestartet, wenn das `INPUT`-Statement aufgrund des `REINPUT`-Statements erneut ausgeführt worden ist.

Wird nach der ersten Ausführung des `INPUT`-Statements eine Hierarchie von Unterprogrammen aufgerufen und das `REINPUT`-Statement steht in einem dieser Unterprogramme, so kehrt Natural automatisch zu dem Programm zurück, das bei der Ausführung des `INPUT`-Statements aktiv war.

Steht ein `INPUT`-Statement innerhalb einer Verarbeitungsschleife, eines Unterprogramms oder eines nur unter bestimmten Bedingungen verarbeiteten Statement-Blocks, so kann ein `REINPUT`-Statement nicht ausgeführt werden, wenn der Status, unter dem das `INPUT`-Statement ausgeführt wurde, bereits beendet ist. Eine derartige Situation würde einen Programmabbruch und eine entsprechende Fehlermeldung zur Folge haben.



Anmerkung: Der `MODIFIED`-Status einer in dem betreffenden `INPUT`-Statement verwendeten Kontrollvariablen wird bei der Ausführung von einem `REINPUT`-Statement (ohne `FULL`-Option) nicht zurückgesetzt. Um zu überprüfen, ob einer Attributkontroll-Variable der Status `MODIFIED` (geändert) zugewiesen wurde, benutzen Sie die `MODIFIED`-Option.

Syntax-Beschreibung

REINPUT FULL	<p>Wenn Sie die Option <code>FULL</code> in einem <code>REINPUT</code>-Statement angeben, wird das entsprechende <code>INPUT</code>-Statement vollständig neu ausgeführt:</p> <ul style="list-style-type: none"> ■ Bei einem normalen <code>REINPUT</code>-Statement (ohne <code>FULL</code>-Option) werden Inhalte von Variablen, die zwischen <code>INPUT</code>- und <code>REINPUT</code>-Statement geändert wurden, nicht angezeigt; d.h. alle Variablen auf dem Schirm zeigen den Inhalt, den sie hatten, als das <code>INPUT</code>-Statement ursprünglich ausgeführt wurde. ■ Bei einem <code>REINPUT FULL</code>-Statement werden alle nach der ersten Ausführung des <code>INPUT</code>-Statements gemachten Änderungen sichtbar, wenn das <code>INPUT</code>-Statement erneut ausgeführt wird; d.h. alle Variablen auf dem Schirm haben den Inhalt, den sie zum Zeitpunkt der Ausführung des <code>REINPUT</code>-Statements hatten. <p>Anmerkung: Der Inhalt reiner Eingabefelder (<code>AD=A</code>) wird durch <code>REINPUT FULL</code> wieder gelöscht.</p> <p>Eine andere Eigenschaft des <code>REINPUT FULL</code>-Statements besteht darin, dass der Status der Kontrollvariable auf <code>NOT MODIFIED</code> (nicht geändert) zurückgesetzt wird. Dies erfolgt nicht mittels des normalen <code>REINPUT</code>-Statements. Um zu überprüfen, ob einer Attribut- Kontrollvariablen der Status <code>MODIFIED</code> (geändert) zugewiesen wurde, benutzen Sie die <code>MODIFIED</code>-Option.</p> <p>Siehe auch Beispiel 3 - REINPUT FULL mit MARK POSITION.</p>															
<i>statement-parameters</i>	<p>Mit einem <code>REINPUT</code>-Statement gesetzte Parameter gelten für alle Felder, die im Statement angegeben sind.</p> <p>Auf Feldebene gesetzte Parameter (siehe MARK-Option) haben für das betreffende Feld Gültigkeit vor auf Statement-Ebene gesetzten.</p> <table border="1" data-bbox="555 1516 1481 1816"> <thead> <tr> <th colspan="2" data-bbox="555 1516 1015 1579">Parameter, die mit <code>REINPUT</code>-Statement angegeben werden können:</th> <th data-bbox="1021 1516 1481 1558">Spezifikation</th> </tr> </thead> <tbody> <tr> <td colspan="2" data-bbox="555 1579 1015 1642"></td> <td data-bbox="1021 1579 1481 1642">S=auf Statement-Ebene</td> </tr> <tr> <td colspan="2" data-bbox="555 1642 1015 1684"></td> <td data-bbox="1021 1642 1481 1684">E=auf Element-Ebene</td> </tr> <tr> <td data-bbox="555 1684 776 1768">AD</td> <td data-bbox="782 1684 1015 1768">Attribute Definition *</td> <td data-bbox="1021 1684 1481 1768">SE</td> </tr> <tr> <td data-bbox="555 1768 776 1810">CD</td> <td data-bbox="782 1768 1015 1810">Color Definition</td> <td data-bbox="1021 1768 1481 1810">S</td> </tr> </tbody> </table> <p>* Wird <code>AD=P</code> auf Statement-Ebene gesetzt, so sind alle Felder geschützt, außer den in der MARK-Option angegebenen.</p>	Parameter, die mit <code>REINPUT</code> -Statement angegeben werden können:		Spezifikation			S=auf Statement-Ebene			E=auf Element-Ebene	AD	Attribute Definition *	SE	CD	Color Definition	S
Parameter, die mit <code>REINPUT</code> -Statement angegeben werden können:		Spezifikation														
		S=auf Statement-Ebene														
		E=auf Element-Ebene														
AD	Attribute Definition *	SE														
CD	Color Definition	S														

	Informationen zu den o.g. Session-Parametern finden Sie in der <i>Parameter-Referenz</i> .
USING HELP	<p>USING HELP bewirkt, dass die für die INPUT-Map definierte Helproutine aufgerufen wird.</p> <p>USING HELP in Verbindung mit der MARK-Option (siehe unten) bewirkt, dass die für das erste in der MARK-Option angegebene Feld definierte Helproutine aufgerufen wird. Ist für das Feld keine Helproutine definiert, wird die Helproutine für die Map aufgerufen.</p> <p>Beispiel:</p> <pre>REINPUT USING HELP MARK 3</pre> <p>In diesem Beispiel wird die für das dritte Feld der INPUT-Map definierte Helproutine aufgerufen.</p>
<i>WITH-TEXT-option</i>	Mit der Option WITH TEXT können Sie einen Text angeben, der in der Meldungszeile angezeigt werden soll. Siehe WITH TEXT-Option weiter unten.
<i>MARK-option</i>	Mit der MARK-Option können Sie ein bestimmtes Feld markieren, d.h. bei der Ausführung des REINPUT-Statements wird der Cursor in dieses Feld plaziert. Siehe MARK-Option weiter unten.
<i>ALARM-option</i>	Diese Option bewirkt, dass der Warnton des Terminals ausgelöst wird, wenn das REINPUT-Statement ausgeführt wird. Siehe ALARM-Option weiter unten.

WITH TEXT-Option

Mit der Option WITH TEXT können Sie einen Text angeben, der in der Meldungszeile angezeigt werden soll. Der Text ist in der Regel eine Meldung, die angibt, welche Maßnahme zum Abarbeiten des Bildschirms getroffen werden sollte, oder wie der Fehler korrigiert werden kann.

```
[WITH] [TEXT] { * operand1
                operand2 } [(attributes)] [,operand3]...7
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	N P I B*	ja	nein
<i>operand2</i>	C S	A U	ja	nein
<i>operand3</i>	C S	A U N P I F B D T L	ja	nein

* Format B von *operand1* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Meldungstext aus der Natural-Fehlermeldungsdatei:</p> <p>Als <i>operand1</i> geben Sie eine Natural-Fehlernummer an. Natural liest dann die entsprechende Fehlermeldung von der Natural- Fehlermeldungsdatei.</p> <p>Es können entweder benutzerdefinierte Meldungen oder Natural- Systemmeldungen gelesen werden.</p> <ul style="list-style-type: none"> ■ Wenn Sie einen positiven Wert von bis zu vier Stellen (z.B.: 0954) angeben, werden benutzerdefinierte Meldungen gelesen. ■ Wenn Sie einen negativen Wert von bis zu vier Stellen (z.B.: -0954) angeben, werden Natural-Systemmeldungen gelesen. <p>Siehe auch Beispiel 4 – WITH TEXT-Option. Natural-Meldungsdateien werden mit der SYSERR-Utility erstellt und gepflegt.</p>
<i>operand2</i>	<p>Meldungstext:</p> <p>Als <i>operand2</i> geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll.</p> <p>Siehe auch Beispiel 4 – WITH TEXT-Option.</p>
<i>attributes</i>	<p>Ausgabe-Attribute:</p> <p>Als <i>attributes</i> können Sie <i>operand1</i> oder <i>operand2</i> bestimmte Anzeige- und Farbattribute zuordnen. Diese Attribute und die Syntax, die benutzt werden kann, sind im Abschnitt Ausgabe-Attribute weiter unten beschrieben.</p>
<i>operand3</i>	<p>Dynamische Ergänzung im Meldungstext:</p> <p><i>operand3</i> kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden.</p> <p>Der angegebene Wert dient dazu, einen Teil einer mit <i>operand1</i> oder <i>operand2</i> angegebenen Meldung dynamisch zu generieren. Innerhalb der Fehlermeldung dient die Notation <i>:n</i> zur Referenzierung von <i>operand3</i>, wobei <i>n</i> die Ausprägung (1 – 7) von <i>operand3</i> darstellt.</p> <p>Siehe auch Beispiel 4 – WITH TEXT-Option.</p> <p>Anmerkung: Werden mehrere <i>operanden3</i> angegeben, müssen diese mit einem Komma voneinander getrennt werden. Falls das Komma als Dezimalzeichen verwendet wird (wie mit dem Session-Parameter DC definiert) und es sich bei <i>operand3</i> um numerische Konstanten handelt, setzen Sie Leerzeichen vor und nach dem Komma, damit es nicht als Dezimalkomma missinterpretiert wird. Alternativ können mehrere <i>operanden3</i> auch mit dem Eingabebegrenzungszeichen (Input Delimiter Character, wie mit dem Session-Parameter ID definiert) voneinander getrennt werden; dies geht jedoch nicht im Falle von ID=/ (Schrägstrich).</p> <p>Nicht signifikante Nullen oder Leerzeichen werden aus dem Feldwert entfernt, bevor er in einer Meldung angezeigt wird.</p>


Ausgabeattribute

attributes sind zur Text-Anzeige verwendete Ausgabeattribute. Sie können folgende *attributes* angeben:

```
{ AD=AD-value ... }
{ CD=CD-value ... } ...
```

Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt:

- *AD* - Attribute-Definition, Abschnitt Feldanzeige
- *CD* - Farbdefinition

 **Anmerkung:** Der Compiler akzeptiert mehr als einen Attributwert für ein Ausgabefeld. Beispielsweise können Sie angeben: *AD=BDI*. In einem solchen Fall gilt allerdings nur der letzte Wert. In dem vorliegenden Beispiel greift nur der Wert *I*, und das Ausgabefeld wird intensiviert dargestellt.

MARK-Option

Mit der *MARK*-Option können Sie ein bestimmtes Feld markieren, so dass bei der Ausführung des *REINPUT*-Statements der Cursor in dieses Feld plaziert wird. Sie können auch eine bestimmte Stelle innerhalb eines Feldes markieren. Außerdem können Sie Felder gegen Eingabe schützen sowie ihre Anzeige- und Farbattribute ändern.

```
MARK [POSITION operand4 [IN]] [FIELD] { { operand5 } [(attributes)] } ...
{ { *fieldname } }
```

Operanden-Definitionstabelle:

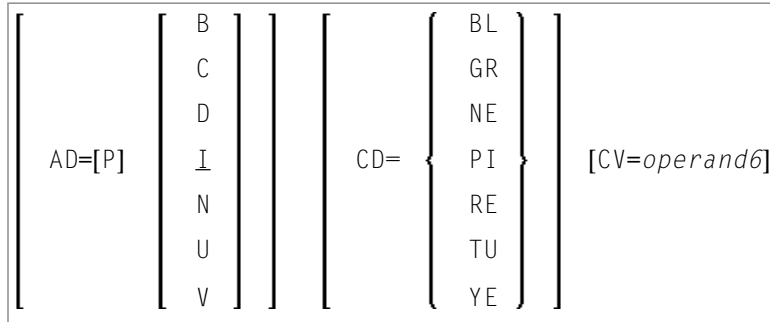
Operand	Mögliche Struktur			Mögliche Formate												Referenzierung erlaubt	Dynam. Definition	
<i>operand4</i>	C	S					N	P	I								ja	nein
<i>operand5</i>	C	S	A				N	P	I								ja	nein

Syntax-Element-Beschreibung:

<i>operand5</i>	<p>Das zu markierende Feld:</p> <p>Jedes mit einem INPUT-Statement angegebene Eingabefeld (AD=A oder AD=M) wird durchnummeriert (beginnend mit 1). Sie können als <i>operand5</i> die Nummer des Feldes angeben, in das der Cursor plaziert werden soll.</p> <p>Die Notation <i>*fieldname</i> wird verwendet, um den Cursor in ein Feld zu positionieren, und zwar mittels des (im INPUT-Statement verwendeten) Namens des Feldes.</p> <p>Ist das betreffende INPUT-Feld ein Array, kann zur Markierung einer oder mehrerer Ausprägungen des Arrays ein eindeutiger Index oder ein Indexbereich angegeben werden.</p> <pre>INPUT #ARRAY (A1/1:5) ... REINPUT (AD=P) 'TEXT' MARK *#ARRAY (2:3)</pre> <p>Ist <i>operand5</i> ebenfalls ein Array, so werden die Werte von <i>operand5</i> als Feldnummern für das INPUT-Array benutzt.</p> <pre>RESET #X(N2/1:2) INPUT #ARRAY REINPUT (AD=P) 'TEXT' MARK #X (1:2)</pre>
MARK POSITION	<p>POSITION-Option:</p> <p>Mit der MARK POSITION-Option können Sie den Cursor an eine bestimmte Stelle, die Sie mit <i>operand4</i> angeben, innerhalb eines Feldes plazieren.</p> <p>Siehe auch Beispiel 3 - REINPUT FULL mit MARK POSITION.</p>
<i>operand4</i>	<p>Cursor-Position: <i>operand4</i> gibt die Cursor-Position an, <i>operand4</i> darf keine Dezimalziffern enthalten.</p>
<i>attributes</i>	<p>Attribut-Zuweisungen: Siehe Attribut-Zuweisungen weiter unten.</p>

Attribut-Zuweisungen:


Sie können explizite Attribute verwenden, um die Darstellung und Farbe der Anzeige der WITH TEXT-Meldung und das Layout des MARK (welches durch das REINPUT-Statement positioniert wird) festzulegen.



Operanden-Definitionstabelle:

Operand	Possible Structure	Possible Formats	Referencing Permitted	Dynamic Definition
<i>operand6</i>	S	C	no	no

Mit dem Attribut AD=P können Sie ein Eingabefeld (AD=A oder AD=M) gegen Eingaben schützen.

 **Anmerkung:** Reine Ausgabefelder (AD=0) können nicht durch ein entsprechendes Attribut zu Eingabefeldern gemacht werden.

Informationen zu den Attributen AD, CD und CV finden Sie in der *Parameter Reference*-Dokumentation.

Die Attribute für die Felder WITH TEXT und MARK brauchen Sie nicht fest anzugeben, sondern Sie können sie dynamisch mittels einer Attributkontrollvariablen zuweisen, die in einer (CV=)-Klausel referenziert wird. Wenn sowohl eine AD- als auch eine CV-Option bei demselben Feld angegeben werden, dann werden die Attribute aus der AD-Option vollständig ignoriert, mit Ausnahme von der Option (AD=P), die wirksam bleibt.

Wird für dasselbe Feld eine CD- und eine CV-Option angegeben, dann wird die Farbe von der CV-Option übernommen. Falls die CV-Variable keine Farbe enthält, wird für dieses Feld die Farbe aus der CD-Option übernommen.

Wird AD=P auf Statement-Ebene gesetzt, so sind alle Felder geschützt außer den in der MARK-Option angegebenen.

ALARM-Option

```
[AND] [SOUND] ALARM
```

Diese Option bewirkt, dass der Warnton des Terminals ausgelöst wird, wenn das REINPUT-Statement ausgeführt wird. Voraussetzung ist, dass die verwendete Terminal-Hardware dies ermöglicht.

Beispiele

- Beispiel 1 — REINPUT-Statement
- Beispiel 2 — REINPUT mit Attribut-Zuweisung
- Beispiel 3 — REINPUT FULL mit MARK POSITION
- Beispiel 4 - mit TEXT-Option
- Beispiel 5 — REINPUT mit Attribut-Zuweisung mittels Kontrollvariable

Beispiel 1 — REINPUT-Statement

```
** Example 'REIEX1': REINPUT
*****
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARAM    (A1)
END-DEFINE
*
INPUT #FUNCTION #PARAM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARAM = 'X'
    REINPUT 'Funktion A with parameter X selected.'
    MARK *#PARAM
  WHEN #FUNCTION = 'C' THRU 'D'
    REINPUT 'Funktion C or D selected.'
  WHEN #FUNCTION = 'X'
    STOP
  WHEN NONE
    REINPUT 'Please enter a valid function.'
    MARK *#FUNCTION
END-DECIDE
*
END
```

Ausgabe des Programms REIEX1:

REINPUT

```
#FUNCTION A #PARM Y
```

Nach Drücken von EINGABE:

```
PLEASE ENTER A VALID FUNCTION  
#FUNCTION A #PARM Y
```

Beispiel 2 — REINPUT mit Attribut-Zuweisung

```
** Example 'REIEX2': REINPUT (with attributes)  
*****  
DEFINE DATA LOCAL  
1 #A (A20)  
1 #B (N7.2)  
1 #C (A5)  
1 #D (N3)  
END-DEFINE  
*  
INPUT (AD=A) #A #B #C #D  
*  
IF #A = ' ' OR #B = 0  
    REINPUT (AD=P) 'RETYPE VALUES'  
        MARK *#A (AD=I CD=RE) /* put cursor on first field  
            *#B (AD=U CD=PI) /* and change colours  
END-IF  
*  
END
```

Beispiel 3 — REINPUT FULL mit MARK POSITION

```
** Example 'REIEX3': REINPUT (with FULL and POSITION option)  
*****  
DEFINE DATA LOCAL  
1 #A (A20)  
1 #B (N7.2)  
1 #C (A5)  
1 #D (N3)  
END-DEFINE  
*  
INPUT (AD=M) #A #B #C #D  
*  
IF #A = ' '  
    COMPUTE #B = #B + #D  
    RESET #D  
END-IF  
*  
IF #A = SCAN 'TEST' OR = ' '
```

```

REINPUT FULL 'RETYPE VALUES' MARK POSITION 5 IN *#A
END-IF
*
END

```

Ausgabe des Programms REIEX3:

```

RETYPE VALUES
#A                #B          0.00 #C          #D          0

```

Beispiel 4 - mit TEXT-Option

```

** Example 'REIEX4': REINPUT (with TEXT option)
*****
DEFINE DATA LOCAL
01 #NAME   (A8)
01 #TEXT   (A20)
END-DEFINE
*
*
INPUT WITH TEXT 'Enter a program name.' 'Program name:' #NAME
*
IF #NAME = ' '
  REINPUT WITH TEXT 'Input missing. Enter a name.'
END-IF
*
IF #NAME NE MASK (A)
  MOVE 'Invalid input.' TO #TEXT
  REINPUT WITH TEXT ':1: Name must start with a letter.',#TEXT
ELSE
  /* Using Natural error message 7600 for demonstration
  COMPRESS *INIT-USER 'on' *DAT4I INTO #TEXT
  INPUT WITH TEXT *-7600,#NAME,#TEXT 'Input accepted.'
END-IF
END

```

Beispiel 5 — REINPUT mit Attribut-Zuweisung mittels Kontrollvariable

```

DEFINE DATA LOCAL
1 #HELLO (A5) INIT <'HELO'>
1 #VAR   (A20) INIT <'Enter "HELLO"'>
1 #CV   (C)
END-DEFINE
*
INPUT (IP=OFF) #HELLO (AD=M)
*
IF #HELLO NE 'HELLO' THEN

```

REINPUT

```
MOVE (AD=U CD=RE) TO #CV
REINPUT FULL WITH TEXT #VAR (CD=YE)
                MARK *#HELLO (CV=#CV)
END-IF
END
```

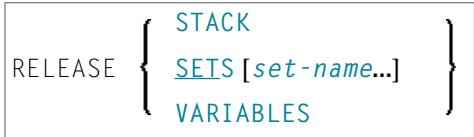

107 REJECT

Siehe Statement [ACCEPT/REJECT](#).

108

RELEASE

▪ Funktion	728
▪ Syntax-Beschreibung	728
▪ Beispiel	729



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [STACK](#) | [FIND with RETAIN option](#) | [DEFINE DATA GLOBAL](#)

Funktion

Das Statement RELEASE dient dazu

- den kompletten Inhalt des Natural-Stacks zu löschen,
- Sätze von ISNs freizugeben, die über ein FIND-Statement mit RETAIN-Klausel zurückgehalten wurden (gilt nur für Adabas-Datenbanken),
- globale und anwendungsunabhängige Variablen auf ihre Ausgangswerte zurückzusetzen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>set-name</i>	C S	A	nein	nein

Syntax-Element-Beschreibung:

RELEASE STACK	Mit RELEASE STACK löschen Sie alle gerade im Natural-Stack gelagerten Kommandos und Daten.
RELEASE SETS	<p>RELEASE SETS gilt nur für Adabas-Datenbanken.</p> <p>Wenn Sie nur RELEASE SETS ohne Angabe eines <i>set-name</i> angeben, werden alle ISNs freigegeben, die mit einem FIND-Statement, das eine RETAIN-Klausel enthält, gehalten wurden.</p>

RELEASE SETS <i>set-name</i>	Mit RELEASE SET <i>set-name</i> geben Sie eine bestimmte ISN frei, zum Beispiel: <pre>RELEASE SET 'CITY-SET' MOVE 'CITY-SET' TO #SET(A32) RELEASE SET #SET</pre>
RELEASE VARIABLES	Mit RELEASE VARIABLES werden alle in der aktuellen Global Data Area definierten Variablen auf ihre Ausgangswerte zurückgesetzt. Gleichzeitig werden alle anwendungsunabhängigen Variablen (AIVs) gelöscht, d.h. sie stehen dann nicht mehr zur Verfügung.

Beispiel

```
** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
  RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

Ausgabe des Programms RELEX1:

RELEASE

NAME	CITY	DATE OF BIRTH
RUBIN	NEW YORK	1945-10-27
WALLACE	NEW YORK	1945-08-04

109 REPEAT

▪ Funktion	732
▪ Syntax-Beschreibung	732
▪ Beispiele	733

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [FOR](#) | [ESCAPE](#)

Gehört zur Funktionsgruppe: *Schleifenverarbeitung*

Funktion

Mit dem Statement REPEAT können Sie eine Verarbeitungsschleife initiieren.

Syntax-Beschreibung

Zwei unterschiedliche Strukturen sind bei diesem Statement möglich:

- **Syntax 1** – Statements werden ein- oder mehrmals ausgeführt
- **Syntax 2** – Statements werden überhaupt nicht oder mehrmals ausgeführt

Wann die Bedingung ausgewertet wird, ist abhängig davon, ob Sie sie an den Anfang oder das Ende der logischen Bedingung stellen.

Weitere Informationen zu logischen Bedingungen, siehe den Abschnitt *Logische Bedingungen im Leitfaden zur Programmierung*).

Eine Erläuterung der in den Syntax-Diagrammen benutzten Symbole siehe *Syntax-Symbole*.

Syntax 1:

```
REPEAT
  statement ... [ { UNTIL } logical-condition ]
                 { WHILE }
END-REPEAT      (nur im Structured Mode)
[ LOOP ]        (nur im Reporting Mode)
```


Syntax 2:

```

REPEAT
  [ { UNTIL }      logical-condition ]  statement...
    [ WHILE ]
END-REPEAT      (nur im Structured Mode)
[ LOOP]         (nur im Reporting Mode)

```

Syntax-Element-Beschreibung:

UNTIL	Die Schleife wird so oft ausgeführt, bis die logische Bedingung erfüllt ist.
WHILE	Die Schleife wird solange ausgeführt, wie die logische Bedingung erfüllt ist.
<i>logical-condition</i>	Wenn Sie eine logische Bedingung angeben, bestimmt die Bedingung, wann die Ausführung der Schleife beendet werden soll. Wenn Sie keine logische Bedingung angeben, müssen Sie die Schleife mit einem ESCAPE- , STOP- oder TERMINATE- Statement beenden, das in der Schleife angegeben ist.
END-REPEAT	Das für Natural reservierte Wort END-REPEAT muss zum Beenden des REPEAT-Statements benutzt werden.

Beispiele

- [Beispiel 1 — REPEAT-Statement](#)
- [Beispiel 2 — REPEAT-Statement mit Optionen WHILE und UNTIL](#)

Beispiel 1 — REPEAT-Statement

```

** Example 'RPTX1S': REPEAT (structured mode)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
*
1 #PERS-NR (A8)
END-DEFINE
*
REPEAT
  INPUT 'ENTER A PERSONNEL NUMBER:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM

```

REPEAT

```
END-IF
/*
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PERS-NR
  IF NO RECORD FOUND
    REINPUT 'NO RECORD FOUND'
  END-NOREC
  DISPLAY NOTITLE NAME
END-FIND
END-REPEAT
*
END
```

Ausgabe des Programms RPTX1S:

```
ENTER A PERSONNEL NUMBER: 11500304
```

Nach Eingabe und Bestätigung der Personalnummer:

```
      NAME
-----
KLUGE
```

Äquivalentes Reporting-Mode-Beispiel: [RPTX1R](#).

Beispiel 2 — REPEAT-Statement mit Optionen WHILE und UNTIL

```
** Example 'RPTX2S': REPEAT (with WHILE and UNTIL option)
*****
DEFINE DATA LOCAL
1 #X (I1) INIT <0>
1 #Y (I1) INIT <0>
END-DEFINE
*
REPEAT WHILE #X <= 5
  ADD 1 TO #X
  WRITE NOTITLE '=' #X
END-REPEAT
*
SKIP 3
REPEAT
  ADD 1 TO #Y
  WRITE '=' #Y
  UNTIL #Y = 6
END-REPEAT
*
END
```

Ausgabe des Programms RPTX2S:

```
#X: 1  
#X: 2  
#X: 3  
#X: 4  
#X: 5  
#X: 6
```

```
#Y: 1  
#Y: 2  
#Y: 3  
#Y: 4  
#Y: 5  
#Y: 6
```

Äquivalentes Reporting-Mode-Beispiel: [RPTX2R](#).

110 REQUEST DOCUMENT

▪ Funktion	738
▪ Syntax-Beschreibung	739
▪ Kodierung von eingehenden/ausgehenden Daten	747
▪ Beispiele	748

```

REQUEST DOCUMENT FROM operand1
    WITH
    [ USER operand2
    [ PASSWORD operand3
    [ HEADER[[NAME] operand4 [VALUE] operand5]]... ]
    [ DATA { ALL operand6 [ ENCODED [[IN] CODEPAGE operand7 ] ]
    { [NAME] operand8 [VALUE] operand9 } ... } ]
    RETURN
    HEADER [ ALL operand10 ] [[NAME] operand11 [VALUE] operand12 ]...
    [ PAGE operand13 [ ENCODED [[FOR TYPES[S] operand14... ] [IN] CODEPAGE operand15 ] ] ]
    RESPONSE operand16
    [ GIVING operand17 ]

```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Gehört zur Funktionsgruppe: *Internet und XML*

Funktion

Mit dem Statement `REQUEST DOCUMENT` haben Sie die Möglichkeit, auf ein externes System zuzugreifen.

Siehe auch *Statements für Internet- und XML-Zugang* im *Leitfaden zur Programmierung*.

Informationen bezüglich Unicode-Support finden Sie im Abschnitt *Statements* in der *Unicode and Code Page Support*-Dokumentation.

Einschränkungen für Cookies

Unter dem HTTP-Protokoll setzt ein Server Cookies ein, um Status-Informationen zur Client-Workstation zu verwalten.

REQUEST DOCUMENT wird mit Optionseinstellungen für das Internet implementiert. Dies bedeutet, dass abhängig von den Sicherheitseinstellungen Cookies verwendet werden.

Wenn in den Einstellungen der Internet-Optionen *Disabled* (Sperrern) gesetzt ist, werden keine Cookies versandt, auch wenn ein Cookie-Header (*operand4/operand5*) versandt wird. Benutzen Sie für Server-Umgebungen nicht die Internet-Optionseinstellung *Prompt* (Eingabeaufforderung). Mit dieser Einstellung bleibt der Server hängen, weil kein Client auf die Eingabeaufforderung reagieren kann.

In Großrechner-Umgebungen werden Cookies nicht unterstützt und ignoriert.

Cookies werden automatisch vom Windows-API verarbeitet. Das heißt: wenn Cookies im Browser aktiviert wurden, werden alle eingehenden Cookies gespeichert und mit der nächsten Anfrage automatisch versendet.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate									Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	C	S			A											nein	ja
<i>operand2</i>	C	S			A											nein	ja
<i>operand3</i>	C	S			A											nein	ja
<i>operand4</i>	C	S			A											nein	ja
<i>operand5</i>	C	S			A	N	P	I	F	D	T	L				nein	ja
<i>operand6</i>	C	S			A	U	N	P	I	F	B	D	T	L		nein	ja
<i>operand7</i>	C	S			A											nein	ja
<i>operand8</i>	C	S			A											nein	ja
<i>operand9</i>	C	S			A	N	P	I	F	D	T	L				nein	ja
<i>operand10</i>		S			A											nein	ja
<i>operand11</i>	C	S			A											nein	ja
<i>operand12</i>		S			A	N	P	I	F	B	D	T	L			nein	ja
<i>operand13</i>		S			A	U				B						nein	ja
<i>operand14</i>	C	S			A											nein	ja

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition
<i>operand15</i>	C	S			A								nein	ja
<i>operand16</i>		S						I4					nein	ja
<i>operand17</i>		S						I4					nein	nein

Syntax-Element-Beschreibung:

DOCUMENT	Adresse des Dokuments:
FROM	<i>operand1</i> ist der URI, der zum Aufruf eines Dokuments benutzt wird.
<i>operand1</i>	Anmerkung: Die folgenden Informationen gelten nur, wenn <i>operand1</i> mit <code>http://</code> oder <code>https://</code> beginnt.
WITH	WITH-Klausel:
	Sie können diese Klausel benutzen, um optional Benutzer/Passwort, Header und Einzelheiten zu den Daten für die Anforderung anzugeben.
USER	Benutzer-Name:
<i>operand2</i>	<i>operand2</i> ist der Name des Benutzers, der für die Anforderung benutzt wird.
PASSWORD	Benutzer-Passwort:
<i>operand3</i>	<i>operand3</i> ist das Passwort des Benutzers, das für die Anforderung benutzt wird.
HEADER	HEADER-Klausel:
{[[NAME]	<i>operand4</i> und <i>operand5</i> können nur gemeinsam verwendet werden.
<i>operand4</i>	<ul style="list-style-type: none"> ■ <i>operand4</i> ist der Name einer mit dieser Anforderung versandten HEADER-Variable. ■ <i>operand5</i> ist der Wert einer mit dieser Anforderung versandten HEADER-Variable.
[VALUE]	
<i>operand5</i>]}...	
	HEADER-Name für <i>operand4</i>:
	HEADER-Namen dürfen nicht CR/LF (Carriage Return/Line Feed) oder Doppelpunkt (:) enthalten. Dies wird nicht vom Statement REQUEST DOCUMENT überprüft. Gültige HEADER-Namen entnehmen Sie den HTTP-Spezifikationen. Aus Gründen der Kompatibilität mit der Web-Schnittstelle können Header-Namen mit Unterstrich (_) anstatt Bindestrich (-) geschrieben werden. (Intern wird der Unterstrich durch einen Bindestrich ersetzt).
	HEADER-Wert für <i>operand5</i>:
	HEADER-Werte dürfen nicht CR/LF enthalten. Dies wird vom Statement REQUEST DOCUMENT nicht überprüft. Gültige Header-Werte und -Formate entnehmen Sie den HTTP-Spezifikationen.
	Allgemeine Informationen zu Headers:

	<p>Für eine HTTP-Anforderung sind einige Headers erforderlich, zum Beispiel: <i>Request-Method</i> oder <i>Content-Type</i>. Diese Headers werden automatisch erzeugt, und zwar abhängig von den mit dem Statement REQUEST DOCUMENT angegebenen Parametern.</p> <p>Siehe auch Automatisch erzeugte Headers.</p>								
DATA	<p>DATA-Klausel:</p> <p>Sie können entweder einen spezifischen DATA-Variablenamen und -wert angeben (siehe <i>operand8</i> und <i>operand9</i> weiter unten) oder das vollständige Dokument (siehe <i>DATA ALL-Klausel</i> weiter unten).</p>								
ALL operand6	<p><i>operand6</i> ist ein vollständiges, zu versendendes Dokument. Dieser Wert ist normalerweise erforderlich für die automatische HTTPAnforderungsmethode PUT (siehe Automatisch erzeugte Headers).</p> <p>Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i>, DATA ALL-Klausel.</p>								
[ENCODED [[IN] CODEPAGE operand7]	<p><i>operand6</i> wird von der voreingestellten (Default-)Codepage (Wert der Systemvariablen *CODEPAGE) in die in <i>operand7</i> angegebenen Codepage umkodiert.</p> <p>Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i>, DATA ALL-Klausel.</p>								
{[NAME] operand8 [VALUE] operand9}...	<p>DATA-Variablenname und -wert:</p> <p><i>operand8</i> und <i>operand9</i> können nur gemeinsam benutzt werden:</p> <ul style="list-style-type: none"> ■ <i>operand8</i> ist der Name einer mit dieser Anfrage zu sendenden DATA-Variablen. Dieser Wert ist erforderlich für die HTTP-Anforderungsmethode POST (URL-Kodierung erforderlich, insbesondere Und-Zeichen (&), Gleichheitszeichen (=), Prozentzeichen (%)). <p>Einschränkung:</p> <p>Wenn <i>operand8/operand9</i> angegeben wird und die Kommunikation standardmäßig http:// oder https:// ist, wird die Anfragemethode POST (siehe Automatisch erzeugte Headers) mit dem Inhaltstyp (<i>Content Type</i>) application/x-www-form-urlencoded benutzt.</p> <p>Im Verlauf der Anfrage werden <i>operand8/operand9</i> durch Gleichheitszeichen (=) und Und-Zeichen (&) voneinander getrennt. Deshalb dürfen die Operanden keine Und-Zeichen (&), Gleichheitszeichen (=) oder (wegen der URL-Kodierung) Prozentzeichen (%) enthalten. Diese Zeichen gelten als "unsicher" und müssen wie folgt kodiert werden:</p> <table border="1"> <thead> <tr> <th>Zeichen</th> <th>URL-Kodierungssyntax</th> </tr> </thead> <tbody> <tr> <td>%</td> <td>%25</td> </tr> <tr> <td>&</td> <td>%26</td> </tr> <tr> <td>=</td> <td>%3D</td> </tr> </tbody> </table> <p>Siehe auch Allgemeine Anmerkung zur URL-Kodierung.</p>	Zeichen	URL-Kodierungssyntax	%	%25	&	%26	=	%3D
Zeichen	URL-Kodierungssyntax								
%	%25								
&	%26								
=	%3D								
RETURN	<p>RETURN-Klausel:</p> <p>Diese Klausel kann benutzt werden, um die HEADER- und/ oder PAGE-Rückgabeinformationen anzugeben.</p>								

<p>HEADER [ALL <i>operand10]</i></p>	<p>RETURN HEADER ALL-Klausel:</p> <p>Wenn diese Klausel angegeben wird, enthält <i>operand10</i> alle mit der HTTP-Rückmeldung angegebenen Header-Werte.</p> <p>Die erste Zeile enthält die Status-Informationen, und alle folgenden Zeilen enthalten die Headers als Paare mit Namen und Werten. Die Namen enden immer auf einen Doppelpunkt (:), und die Werte enden mit einem Zeilenvorschub (LF). (Intern werden alle CR/LF auf Zeilenvorschub, d.h. LF, umgesetzt.)</p>
<p>HEADER [[NAME] [VALUE] <i>operand11]</i> <i>operand12]</i>...</p>	<p>RETURN HEADER NAME/VALUE-Klausel:</p> <p>Wenn diese Klausel angegeben wird, werden nur spezifische HEADER-Informationen zurückgegeben.</p> <p><i>operand11</i> und <i>operand12</i> können nur in Verbindung miteinander benutzt werden:</p> <ul style="list-style-type: none"> ■ <i>operand11</i> ist der Name eines in dieser Anfrage erhaltenen Headers. Die HEADER-Angabe ist für HTTP erforderlich. ■ <i>operand12</i> ist der Wert eines in dieser Anfrage erhaltenen Headers. Die HEADER-Angabe ist für HTTP erforderlich. <p>Rückgabe des Header-Namens für <i>operand11</i>:</p> <p>Aus Gründen der Kompatibilität mit dem Natural Web Interface können Header-Namen mit Unterstrich (_) anstatt Bindestrich (-) geschrieben werden. Intern wird der Unterstrich durch einen Bindestrich ersetzt.</p> <p>Wenn <i>operand11</i> eine Leerzeichenkette ist, werden die Status-Informationen zurückgegeben:</p> <pre>HTTP/1.0 200 OK</pre>
<p>RETURN PAGE</p>	<p>RETURN PAGE-Klausel:</p> <p>Sie können die PAGE-Klausel benutzen, wenn Sie möchten, dass die eingehenden Daten in einer spezifischen Codepage kodiert werden.</p> <p>Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i>, RETURN PAGE-Klausel weiter unten.</p>
<p>PAGE <i>operand13]</i></p>	<p><i>operand13</i> ist das für diese Anfrage zurückgegebene Dokument.</p> <p>Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i>, RETURN PAGE-Klausel weiter unten.</p>
<p>[ENCODED [[FOR TYPE[S] <i>operand14...</i> [IN] CODEPAGE <i>operand15]</i></p>	<p><i>operand14</i> ist die Liste der Mime-Typen, für die eine Kodierung des zurückgegebenen Dokuments in <i>operand13</i> ausgeführt wird.</p> <p>Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i>, RETURN PAGE-Klausel weiter unten.</p> <p><i>operand15</i> ist die Codepage, die erforderlichenfalls für die Kodierung von <i>operand13</i> benutzt wird.</p> <p>Wenn der Wert von <i>operand15</i> leer ist, wird keine Konvertierung vorgenommen. <i>operand13</i> ist dann mit der Standard-Codepage kodiert (Profilparameter CP im Configuration Utility).</p>

	Siehe <i>Kodierung von eingehenden/ausgehenden Daten</i> , RETURN PAGE-Klausel weiter unten.
RESPONSE <i>operand16</i>	<p>RESPONSE-Klausel:</p> <p>Geben Sie die RESPONSE-Klausel an, wenn Sie möchten, dass die Response-Codenummer der Anforderung angezeigt wird.</p> <p><i>operand16</i> ist die Response-Codenummer der Anforderung, zum Beispiel: 200 (Anforderung erledigt).</p> <p>Siehe auch Übersicht über Response-Nummern für HTTP/HTTPs-Anfragen.</p>
GIVING <i>operand17</i>	<p>GIVING-Klausel:</p> <p><i>operand17</i> enthält den Natural-Fehler, wenn die Anforderung nicht ausgeführt werden konnte.</p>

Automatisch erzeugte Header (operand4/5)

Request-Method

Die folgenden Werte werden für *operand5* unterstützt: HEAD, POST, GET und PUT.

Die folgende Tabelle zeigt die automatische Berechnung der *request-method* in Abhängigkeit von den vorgegebenen Operanden:

Operand	Request-Method			
	HEAD	POST	GET	PUT
WITH HEADER <i>(operand4/operand5)</i>	optional	optional	optional	optional
WITH DATA <i>(operand7/operand8)</i>	nicht angegeben	angegeben	nicht angegeben	nur bei Option ALL <i>(operand6)</i>
RETURN HEADER <i>(operand10 bis operand12)</i>	angegeben	angegeben	optional	optional
RETURN PAGE <i>(operand13)</i>	nicht angegeben	angegeben	angegeben	optional

Content-Type

Wenn die *request-method* POST ist, muss ein Header des Typs *content-type* mit der HTTP-Anfrage angegeben werden. Wenn kein *content-type* explizit gesetzt wird, ist der automatisch generierte Wert von *operand5* wie folgt:

```
application/x-www-form-urlencoded
```



Anmerkung: Es ist möglich, die automatisch erzeugten Headers zu überschreiben. Natural überprüft sie nicht auf Fehler. Es können unerwartete Fehler auftreten.

Allgemeiner Hinweis zur URL-Kodierung

Wenn Sie POST-Daten mit dem Inhaltstyp `application/x-www-form-urlencoded` versenden, müssen bestimmte Zeichen mittels URL-Kodierungen dargestellt werden, was bedeutet, dass das Zeichen durch einen hexadezimalen Zeichencode (`%hexadecimal-character-code`) ersetzt wird. Die vollständigen Einzelheiten, wann und warum die URL-Kodierung erforderlich ist, sind in RFC 1630, RFC 1738 und RFC 1808 erläutert. Sie finden hier einige grundsätzliche Informationen. Alle Nicht-ASCII-Zeichen (d.h. gültige ISO 8859/1-Zeichen, die nicht auch ASCII-Zeichen sind) müssen URL-kodiert sein, z.B. die Datei `köln.html` würde in einer URL als `k%F6ln.html` erscheinen.

Einige Zeichen gelten als „unsicher“, wenn Web-Seiten per Email angefordert werden.

Diese Zeichen lauten:

Zeichen	URL-Kodierungssyntax
Tab-Zeichen	%09
Leerzeichen	%20
[%5B
\	%5C
]	%5D
^	%5E
`	%60
{	%7B
	%7C
}	%7D
~	%7E

Wenn Sie URLs schreiben, sollten Sie diese Zeichen URL-kodieren.

Einige Zeichen haben spezielle Bedeutungen in URLs, wie z.B. der Doppelpunkt (:), der das URL-Schema vom Rest des URLs abtrennt, der doppelte Schrägstrich (//), der angibt, dass der URL der Common Internet Scheme-Syntax entspricht, und das Prozentzeichen (%). Wenn diese Zeichen

als Teile von Dateinamen erscheinen, müssen sie generell URL-kodiert werden, um sie von ihrer Sonderbedeutung in URLs zu unterscheiden (dies ist eine Vereinfachung, die vollständigen Informationen finden Sie in den RFCs).

Diese Zeichen sind:

Zeichen	URL-Kodierungssyntax
"	%22
#	%23
%	%25
&	%26
+	%2B
,	%2C
/	%2F
:	%3A
<	%3C
=	%3D
>	%3E
?	%3F
@	%40

Übersicht über Response-Nummern für HTTP-Anforderungen

Status	Wert	Rückmeldung
STATUS CONTINUE	100	OK — Fortfahren mit der Anforderung.
STATUS SWITCH_PROTOCOLS	101	Server hat die Protokolle im Upgrade-Header geändert.
STATUS OK	200	Anforderung erledigt.
STATUS CREATED	201	Objekt erstellt, Grund = neuer URL.
STATUS ACCEPTED	202	Asynchrone Beendigung (TBS).
STATUS PARTIAL	203	Teilweise Beendigung.
STATUS NO_CONTENT	204	Keine Infos zurückzugeben.
STATUS RESET_CONTENT	205	Anforderung abgeschlossen, aber Inhalt zurückgesetzt.
STATUS PARTIAL_CONTENT	206	Teilweises GET vollendet.
STATUS AMBIGUOUS	300	Server konnte keine Entscheidung über Rückmeldung fällen.
STATUS MOVED	301	Objekt permanent übertragen.
STATUS REDIRECT	302	Objekt zeitweise übertragen.
STATUS REDIRECT_METHOD	303	Umleiten ohne neue Zugriffsmethode.
STATUS NOT_MODIFIED	304	Wenn geändert — seit wann nicht geändert.

Status	Wert	Rückmeldung
STATUS USE_PROXY	305	Umleiten zu Proxy, Adress-Header gibt zu benutzende Proxy an.
STATUS REDIRECT_KEEP_VERB	307	HTTP/1.1: dasselbe Verb beibehalten.
STATUS BAD_REQUEST	400	Ungültige Syntax.
STATUS DENIED	401	Zugriff verweigert.
STATUS PAYMENT_REQ	402	Zahlung erforderlich.
STATUS FORBIDDEN	403	Anforderung nicht erlaubt.
STATUS NOT_FOUND	404	Objekt nicht gefunden.
STATUS BAD_METHOD	405	Methode ist nicht zulässig.
STATUS NONE_ACCEPTABLE	406	Keine Rückmeldung für gefundenen Client annehmbar.
STATUS PROXY_AUTH_REQ	407	Proxy-Echtheitsprüfung erforderlich.
STATUS REQUEST_TIMEOUT	408	Server-Zeitüberschreitung – warten auf Anforderung.
STATUS CONFLICT	409	Benutzer sollte mit mehr Informationen neu starten.
STATUS GONE	410	Die Ressource steht nicht mehr zur Verfügung.
STATUS LENGTH_REQUIRED	411	Der Server verweigerte die Annahme der Anforderung ohne Länge.
STATUS PRECOND_FAILED	412	In Anfrage angegebene Vorbedingung unzulässig
STATUS REQUEST_TOO_LARGE	413	Anforderungselement war zu groß.
STATUS URL_TOO_LONG	414	Anforderungs-URL zu lang.
STATUS UNSUPPORTED_MEDIA	415	Nicht unterstützter Medien-Typ.
STATUS SERVER_ERROR	500	Interner Server-Fehler.
STATUS NOT_SUPPORTED	501	„Required“ nicht unterstützt.
STATUS BAD_GATEWAY	502	Fehler-Rückmeldung vom Gateway.
STATUS SERVICE_UNAVAIL	503	Zeitweise überlastet.
STATUS GATEWAY_TIMEOUT	504	Zeitüberschreitung – warten auf Gateway.
STATUS VERSION_NOT_SUP	505	HTTP-Version nicht unterstützt.

Response 301 - 303 (Umleitung)

Umleitung bedeutet, dass der angeforderte URL umgezogen ist. Als Response (Rückmeldung) wird der Rückgabe-Header mit dem Namen `LOCATION` (Adresse) angezeigt. Dieser Header enthält den URL, wohin die angeforderte Seite umgezogen ist. Eine neue `REQUEST DOCUMENT`-Anfrage kann benutzt werden, um die umgezogene Seite zu suchen.

HTTP-Browser leiten automatisch zur neuen URL um, aber das Statement `REQUEST DOCUMENT` nimmt die Umleitung nicht automatisch vor.

Response 401 (Verweigert)

Die Rückmeldung `Access Denied` (Zugriff verweigert) bedeutet, dass die angeforderte Seite nur aufgerufen werden kann, wenn mit der Anfrage eine gültige Benutzer-ID und ein gültiges Passwort

angegeben werden. Als Rückmeldung wird der Rückgabe-Header mit dem Namen WWW-AUTHENTICATE mit dem für diese Anfrage erforderlichen Bereich ausgegeben.

HTTP-Browser zeigen normalerweise einen Dialog mit Benutzer-ID und Passwort an, aber beim Statement REQUEST DOCUMENT wird kein Dialog angezeigt.

Kodierung von eingehenden/ausgehenden Daten

Bei der Datenübertragung mit dem Statement REQUEST DOCUMENT kommt es normalerweise nicht zu Konvertierungen von Codepages. Wenn Sie möchten, dass die ausgehenden und/oder eingehenden Daten in einer bestimmten Codepage kodiert werden, können sie die Klausel DATA ALL und/oder die Klausel RETURN PAGE verwenden, um dies anzugeben.

DATA ALL-Klausel

Zur Kodierung der ausgehenden Daten wird die Klausel DATA ALL benutzt:

```
ALL operand6 [ENCODED [[IN] CODEPAGE operand7]]
```

Syntax-Beschreibung:

ALL <i>operand6</i>	<i>operand6</i> ist ein vollständiges Dokument, das versandt werden soll. Dieser Wert wird normalerweise für die automatische HTTP-Anforderungsmethode PUT benötigt (siehe Automatisch erzeugte Headers).
[ENCODED [[IN] CODEPAGE <i>operand7]</i>	<i>operand6</i> wird von der voreingestellten (Default-)Codepage (Wert der Systemvariablen *CODEPAGE) in die in <i>operand7</i> angegebenen Codepage umgesetzt.

RETURN PAGE-Klausel

Zur Kodierung von eingehenden Daten wird die Klausel RETURN PAGE verwendet:

```
[PAGE operand13 [ENCODED [[FOR TYPE[S] operand14...] [IN] CODEPAGE operand15]]]
```

Als Rückmeldung für eine HTTP-/HTTPS-Anfrage können eingehende Daten binäre Daten (zum Beispiel image/gif) oder Zeichen-Daten (zum Beispiel text/html) enthalten. Zusammen mit der Rückmeldung erhält das Statement REQUEST DOCUMENT einen Parameter, der die Art des Inhalts des angeforderten Dokuments angibt (Mime-Typ). Dieser Parameter kann Informationen über die Codepage enthalten, in der das Dokument kodiert ist.

Diese Klausel bietet eine automatische Umsetzung in die voreingestellte (Default-)Codepage (Wert der Systemvariablen *CODEPAGE) der Natural-Session.

Syntax-Beschreibung:

RETURN PAGE <i>operand13</i>	Es erfolgt keine Kodierung der zurückgegebenen Seite; das bedeutet, die Seite bleibt so kodiert, wie sie vom HTTP-Server geliefert wird.
RETURN PAGE <i>operand13</i> ENCODED	Wenn der zurückgegebene Mime-Typ eine Kodierung enthält, dann wird <i>operand13</i> von dieser Codepage in die voreingestellte (Default-)Codepage (A/B) oder (U) umkodiert. Siehe Anmerkung unten.
RETURN PAGE <i>operand13</i> ENCODED [IN] CODEPAGE <i>operand15</i>	Wenn der der zurückgegebene Mime-Typ keine Kodierung enthält, wird <i>operand13</i> von der mit <i>operand15</i> definierten Codepage in die voreingestellte (Default-)Codepage (Wert der Systemvariablen *CODEPAGE) (A/B) oder (U) umkodiert.
RETURN PAGE <i>operand13</i> [ENCODED [[FOR TYPE[S] <i>operand14...</i>] [IN] CODEPAGE <i>operand15]]</i>	Wenn der zurückgegebene Mime-Typ keine Codierung enthält, wird eine zusätzliche Prüfung durchgeführt, ob der zurückgegebene Mime-Typ mit einem der in <i>operand14</i> gelieferten Typen übereinstimmt. Wenn eine Übereinstimmung vorliegt, wird <i>operand13</i> von der mit <i>operand15</i> definierten Codepage in die voreingestellte (Default-)Codepage (A/B) oder (U) umkodiert.



Anmerkung: „Zurückgegebener Mime-Typ enthält eine Kodierung“ bedeutet, dass der HTTP-Server einen Content-Type-Header mit einer `charset=`-Klausel zurückliefert, zum Beispiel: `charset=ISO-8859-1`.

Beispiele

- [Beispiel 1 — Allgemeine Anforderung](#)
- [Beispiel 2 — Einfache Get-Anforderung \(keine Daten\)](#)
- [Beispiel 3 — Einfache Head-Anforderung \(keine zurückgelieferte Seite\)](#)
- [Beispiel 4 — Einfache Post-Anforderung \(Voreinstellung\)](#)
- [Beispiel 5 — Einfache Put-Anforderung \(mit allen Daten\)](#)



Anmerkung: Es gibt einen Beispiel-Dialog `V5-RDOC` für dieses Statement in der Beispiel-Library `SYSEXV`.

Beispiel 1 — Allgemeine Anforderung

```

REQUEST DOCUMENT FROM "http://bo1sap1:5555/invoke/sap.demo/handle_RFC_XML_POST"
WITH
  USER #User PASSWORD #Password
  DATA
  NAME 'XMLData'          VALUE #Queryxml
  NAME 'repServerName' VALUE 'NT2'
RETURN
  PAGE #Resultxml
RESPONSE #rc

```

Beispiel 2 — Einfache Get-Anforderung (keine Daten)

```

REQUEST DOCUMENT FROM "http://pcnatweb:8080"
RETURN
  PAGE #Resultxml
RESPONSE #rc

```

Beispiel 3 — Einfache Head-Anforderung (keine zurückgelieferte Seite)

```

REQUEST DOCUMENT FROM "http://pcnatweb"
RESPONSE #rc

```

Beispiel 4 — Einfache Post-Anforderung (Voreinstellung)

```

REQUEST DOCUMENT FROM "http://pcnatweb/cgi-bin/nwwcgi.exe/sysweb/nat-env"
WITH
  DATA
  NAME 'XMLData'          VALUE #Queryxml
  NAME 'repServerName' VALUE 'NT2'
RETURN
  PAGE #Resultxml
RESPONSE #rc

```

Beispiel 5 — Einfache Put-Anforderung (mit allen Daten)

```

REQUEST DOCUMENT FROM "http://pcnatweb/test.txt"
WITH
  DATA ALL      #document
RETURN
  PAGE #Resultxml
RESPONSE #rc

```


111 RESET

▪ Funktion	752
▪ Syntax-Beschreibung	753
▪ Beispiel	754

```
RESET [INITIAL] operand1 ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `ADD` | `COMPRESS` | `COMPUTE` | `DIVIDE` | `EXAMINE` | `MOVE` | `MOVE ALL` | `MULTIPLY` | `SEPARATE` | `SUBTRACT`

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das `RESET`-Statement wird benutzt, um den Wert eines Feldes zurückzusetzen.

- Mit dem Statement `RESET` (ohne `INITIAL`) können Sie in Abhängigkeit von seinem Format den Inhalt jedes angegebenen Feldes auf seinen **Standard-Ausgangswert** zurücksetzen.
- Mit `RESET INITIAL` können Sie jedes Feld auf einen im `DEFINE DATA`-Statement definierten Ausgangswert zurücksetzen.

Bei einem ohne `INIT`-Klausel im `DEFINE DATA`-Statement deklarierten Feld hat `RESET INITIAL` die gleiche Auswirkung wie `RESET` (ohne `INITIAL`).



Anmerkungen:

1. Ein mit einer `CONSTANT`-Klausel im `DEFINE DATA`-Statement deklariertes Feld kann in einem `RESET`-Statement nicht referenziert werden, da sein Inhalt nicht geändert werden kann.
2. Im Reporting Mode kann das `RESET`-Statement auch verwendet werden, um eine Variable zu definieren, vorausgesetzt dass das Programm kein `DEFINE DATA LOCAL`-Statement enthält.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G M	A U N P I F B D T L C G O	ja	ja

Syntax-Element-Beschreibung:

RESET <i>operand1</i>	<p>Zurücksetzen auf Nullwert:</p> <p>RESET (ohne INITIAL) setzt jedes angegebene Feld (<i>operand1</i>) auf seinen Standard-Ausgangswert.</p> <p>Wenn <i>operand1</i> eine dynamische Variable ist, wird sie auf einen Nullwert zurückgesetzt, und zwar mit der Länge, die die Variable zum Zeitpunkt der Ausführung des RESET-Statements hat. Die aktuelle Länge einer dynamischen Variable kann mittels der Systemvariable *LENGTH ermittelt werden.</p> <p>Allgemeine Informationen über dynamische Variablen finden Sie in dem Abschnitt <i>Dynamische und große Variablen benutzen</i>.</p>
RESET INITIAL <i>operand1</i>	<p>Zurücksetzen auf Ausgangswert:</p> <p>Mit RESET INITIAL werden die angegebenen Felder (<i>operand1</i>) auf die für sie im DEFINE DATA-Statement definierten Ausgangswerte zurückgesetzt.</p> <ul style="list-style-type: none"> ■ Wenn für ein Feld kein Ausgangswert definiert ist, wird es abhängig von seinem Format auf einen Standard-Ausgangswert (siehe unten) zurückgesetzt. ■ Wenn eine dynamische Variable benutzt wird, wird die Systemvariable *LENGTH auf Null gesetzt, wenn kein Ausgangswert definiert ist. ■ Wenn Sie RESET INITIAL auf ein Array anwenden, müssen Sie es auf das gesamte Array (wie im DEFINE DATA-Statement definiert) anwenden; RESET INITIAL für einzelne Array-Ausprägungen ist nicht möglich. ■ Wird ein X-array verwendet, wird die Systemvariable *OCCURRENCE auf Null gesetzt. ■ Ein RESET INITIAL von aus einer Redefinition hervorgehenden Feldern ist ebenfalls nicht möglich. ■ RESET INITIAL wird für eine dynamische Variable benutzt. ■ Auf Datenbankfelder ist RESET INITIAL nicht anwendbar.

Beispiel

```

** Example 'RSTEX1': RESET (with/without INITIAL)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 #BINARY (B4) INIT <1>
1 #INTEGER (I4) INIT <5>
1 #NUMERIC (N2) INIT <25>
END-DEFINE
*
LIMIT 1
READ EMPLOY-VIEW
  /*
  WRITE NOTITLE 'VALUES BEFORE RESET STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
  RESET NAME #BINARY #INTEGER #NUMERIC
  /*
  WRITE /// 'VALUES AFTER RESET STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
  RESET INITIAL #BINARY #INTEGER #NUMERIC
  /*
  WRITE /// 'VALUES AFTER RESET INITIAL STATEMENT:'
  WRITE / '=' NAME '=' #BINARY '=' #INTEGER '=' #NUMERIC
  /*
END-READ
END

```

Ausgabe des Programms RSTEX1:

```

VALUES BEFORE RESET STATEMENT:
NAME: ADAM                #BINARY: 00000001 #INTEGER:          5 #NUMERIC:
 25

VALUES AFTER RESET STATEMENT:
NAME:                     #BINARY: 00000000 #INTEGER:          0 #NUMERIC:
 0

```

VALUES AFTER RESET INITIAL STATEMENT:

NAME: #BINARY: 00000001 #INTEGER: 5 #NUMERIC:
25

112

RESIZE

▪ Funktion	758
▪ Syntax-Beschreibung	758

```
RESIZE { dynamic-clause
        array-clause } [GIVING operand5]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [EXPAND](#) | [REDUCE](#)

Gehört zur Funktionsgruppe: [Speicherverwaltungskontrolle für dynamische Variablen/X-Arrays](#).

Funktion

Das Statement RESIZE dient dazu, Folgendes anzupassen:

- die zugewiesene Länge einer dynamischen Variable (*dynamic-clause*) oder
- die Anzahl der Ausprägungen von X-Arrays (*array-clause*).

Weitere Informationen entnehmen Sie den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- *Dynamische Variablen benutzen*
- *Hauptspeicherplatz für eine dynamische Variable zuweisen/freigeben*
- *X-Arrays*
- *Speicherverwaltung von X-Gruppen-Arrays*

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur		Mögliche Formate												Referenzierung erlaubt	Dynam. Definition		
<i>operand1</i>	S	A	A	U						B							nein	nein
<i>operand2</i>	C	S							I								nein	nein
<i>operand3</i>			A	G	A	N	P	I	F	B	D	T	L	C	G	O	ja	nein
<i>operand4</i>	C	S				N	P	I									nein	nein
<i>operand5</i>		S							I4								nein	ja

Syntax-Element-Beschreibung:

<i>dynamic-clause</i>	Mit dem Statement <code>RESIZE DYNAMIC</code> können Sie die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variable (<i>operand1</i>) an den mit <i>operand2</i> angegebenen Wert anpassen. Weitere Informationen, siehe DYNAMIC-Klausel weiter unten.
<i>operand1</i>	<i>operand1</i> ist die dynamische Variable, für die die zugewiesene Länge angepasst werden soll.
<i>operand2</i>	<i>operand2</i> dient dazu, die neue Länge der dynamischen Variable anzugeben. Der angegebene Wert muss eine nicht negative, numerische Ganzzahl-Konstante oder eine Variable des Typs Integer4 (I4) sein.
<i>array-clause</i>	Mit dem Statement <code>RESIZE ARRAY</code> wird die Anzahl der Ausprägungen des X-Arrays (<i>operand3</i>) auf die mit (<i>dim[, dim[, dim]]</i>) angegebene Ober- und Untergrenze angepasst. Weitere Informationen siehe Array-Klausel weiter unten.
<i>operand3</i>	<i>operand3</i> ist das X-Array. Die Ausprägungen des X-Arrays können erweitert oder verringert werden. Die Index-Notation des Arrays ist optional. Als Index-Notation ist für jede Dimension nur die Stern-Notation (*) für den vollständigen Bereich zulässig.
dim <i>operand4</i>	Die Notation für die Ober- und Untergrenze (<i>operand4</i> oder Stern-Notation), auf die das X-Array erweitert werden soll, wird hier angegeben. Wenn der aktuelle Wert der Ober- oder Untergrenze verwendet werden soll, kann ein Stern (*) anstatt <i>operand4</i> angegeben werden. Weitere Informationen, siehe Dimension weiter unten.
GIVING <i>operand5</i>	Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die GIVING-Klausel angegeben wird, enthält <i>operand5</i> die Natural-Fehlernummer, wenn vorher ein Fehler aufgetreten ist, oder Null (0) bei Erfolg.

DYNAMIC-Klausel

```
[SIZE OF] DYNAMIC [VARIABLE] operand1 TO operand2
```

Mit dem Statement `RESIZE DYNAMIC` können Sie die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variable (*operand1*) an den mit *operand2* angegebenen Wert anpassen.

Wenn Sie das `RESIZE`-Statement benutzen, wird die Anzahl der Ausprägungen an die erforderlichen Werte angepasst, ungeachtet der Tatsache, ob die Anzahl der Ausprägungen erhöht oder verringert werden muss.

ARRAY-Klausel

```
[AND RESET] [OCCURRENCES OF] ARRAY operand3 TO (dim[,dim[,dim]])
```

Mit dem Statement RESIZE ARRAY wird die Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit TO (*dim*[,*dim*[,*dim*]]) angegebene Ober- und Untergrenze angepasst, wobei jedes *dim* eine mittels der im Folgenden beschriebenen Syntax definierte Dimension ist.

Die RESET-Option setzt alle Ausprägungen des größenmäßig angepassten X-Arrays auf ihren standardmäßigen Nullwert zurück. Als Voreinstellung (keine RESET-Option) werden die Direktwerte beibehalten, und die größenmäßig angepassten (neuen) Ausprägungen werden zurückgesetzt.

Eine in einem RESIZE-Statement benutzte Ober- und Untergrenze muss genau mit der betreffenden, für das Array definierten Ober- und Untergrenze identisch sein.

Beispiel:

```
DEFINE DATA LOCAL
1 #a(I4/1:*)
1 #g(1:*)
  2 #ga(I4/1:*)

1 #i(i4)
END-DEFINE
...

/* resizing #a (1:10)
RESIZE ARRAY #a TO (1:10)          /* #a is resized to
RESIZE ARRAY #a TO (*:10)         /* 10 occurrences.

/* resizing #ga (1:10,1:20)
RESIZE ARRAY #g TO (1:10)          /* 1st dimension is set to (1:10)
RESIZE ARRAY #ga TO (*:*,1:20)    /* 1st dimension is dependent and
                                  /* therefore kept with (*:*)
                                  /* 2nd dimension is set to (1:20)

RESIZE ARRAY #a TO (5:10)          /* This is rejected because the lower index
                                  /* must be 1 or *
RESIZE ARRAY #a TO (#i:10)        /* This is rejected because the lower index
                                  /* must be 1 or *

RESIZE ARRAY #ga TO (1:10,1:20)    /* (1:10) for the 1st dimension is rejected
                                  /* because the dimension is dependent and
                                  /* must be specified with (*:*)
```

Weitere Informationen siehe:

- *Speicherverwaltung von X-Arrays*

■ *Speicherverwaltung von X-Gruppen-Arrays*

Dimension

Jede in der Array-Klausel angegebene Dimension (*dim*) wird mittels der folgenden Syntax definiert:

$$\left\{ \begin{array}{c} \textit{operand4} \\ * \end{array} \right\} : \left\{ \begin{array}{c} \textit{operand4} \\ * \end{array} \right\}$$

Die Notation für die Ober- und Untergrenze (*operand4* oder Stern-Notation), auf die das X-Array erweitert werden soll, wird hier angegeben. Wenn der aktuelle Wert der Ober- oder Untergrenze benutzt werden soll, kann ein Stern (*) anstatt von *operand4* angegeben werden. An Stelle von *: * können Sie auch einen einzelnen Stern angeben.

Die Anzahl der Dimensionen (*dim*) muss genau mit der definierten Anzahl der Dimensionen des X-Arrays (1, 2 oder 3) übereinstimmen.

Wenn die Anzahl der Ausprägungen für eine angegebene Dimension kleiner als die Anzahl der aktuell zugewiesenen Ausprägungen ist, wird die Anzahl der Ausprägungen für die entsprechende Dimension nicht aktualisiert.

113

RETRY

▪ Funktion	764
▪ Einschränkung	764
▪ Beispiel	764

RETRY

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ACCEPT/REJECT](#) | [AT BREAK](#) | [AT START OF DATA](#) | [AT END OF DATA](#) | [BACKOUT TRANSACTION](#) | [BEFORE BREAK PROCESSING](#) | [DELETE](#) | [END TRANSACTION](#) | [FIND](#) | [GET](#) | [GET SAME](#) | [GET TRANSACTION DATA](#) | [HISTOGRAM](#) | [LIMIT](#) | [PASSW](#) | [PERFORM BREAK PROCESSING](#) | [READ](#) | [STORE](#) | [UPDATE](#)

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement `RETRY` wird in einem `ON ERROR`-Statement-Block (siehe [ON ERROR](#)-Statement) verwendet. Es dient dazu, erneut zu versuchen, auf einen Datensatz zuzugreifen, auf den bereits ein anderer Benutzer zugegriffen hat und der sich daher im Hold-Status befindet.

Befindet sich ein Datensatz im Hold für einen anderen Benutzer, gibt Natural die Fehlermeldung 3145 aus. Siehe auch Session-Parameter `WH` (Wait for Record in Hold Status).

Das `RETRY`-Statement muss in dem Objekt stehen, das die Fehlermeldung 3145 verursacht. Weitere Informationen zur Record-Hold-Logik finden Sie im Abschnitt *Datensatz-Kontrolle während einer Transaktion (Hold-Logik)* im Leitfaden zur Programmierung.

Einschränkung

Dieses Statement kann nur für den Zugriff auf Adabas-Datenbanken verwendet werden.

Beispiel

```
** Example 'RTYEX1': RETRY
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
*
1 #RETRY (A1) INIT <' '>
END-DEFINE
```



```
*
FIND EMPLOY-VIEW WITH NAME = 'ALDEN'
/*
DELETE
END TRANSACTION
/*
ON ERROR
  IF *ERROR-NR = 3145
    INPUT NO ERASE 10/1
      'RECORD IS IN HOLD' /
      'DO YOU WISH TO RETRY?' /
      #RETRY '(Y)ES OR (N)O?'
    IF #RETRY = 'Y'
      RETRY
    ELSE
      STOP
    END-IF
  END-IF
END-ERROR
/*
AT END OF DATA
  WRITE NOTITLE *NUMBER 'RECORDS DELETED'
END-ENDDATA
END-FIND
*
END
```


114 RUN

▪ Funktion	768
▪ Syntax-Beschreibung	768
▪ Dynamische Sourcecode-Generierung und -Ausführung	769
▪ Beispiel	770

```
RUN [REPEAT] operand1 [ operand2 [(parameter)]] ... 40
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Gehört zur Funktionsgruppe: [Aufrufen von Programmen und Unterprogrammen](#)

Funktion

Das RUN-Statement dient dazu, ein Natural-Source-Programm aus der Natural-Systemdatei zu lesen und es dann auszuführen.

Für Natural Remote Procedure Call (RPC): Siehe *Notes on Natural Statements on the Server* (in der *Natural Remote Procedure Call (RPC)*-Dokumentation).

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C	S		A	ja	nein
<i>operand2</i>	C	S	A	G A U N P I F B D T L G	ja	nein

Syntax-Element-Beschreibung:

REPEAT	<p>Mit RUN REPEAT wird ein Programm vollständig ausgeführt, ohne dass der Benutzer zwischendurch auf etwaigen (durch INPUT-Statements ausgegebenen) Ausgabeschirmen durch eine Eingabe reagieren muss.</p> <p>Diese Option kann eingesetzt werden, wenn ein Programm mehrere Schirme mit Informationen ausgibt, bei denen es nicht erforderlich ist, dass der Benutzer auf jeden ausgegebenen Schirm reagiert.</p>
<i>operand1</i>	<p>Programmname:</p> <p>Der Name des auszuführenden Programms (<i>operand1</i>) kann als alphanumerische Konstante oder als Inhalt einer alphanumerischen Variablen angegeben werden. Wird eine Variable verwendet, so muss sie 8 Stellen lang sein.</p>

	<p>Das Programm kann entweder in der aktuellen Library oder in einer Steplib (Standard-Steplib ist SYSTEM) gespeichert sein. Wird es dort nicht gefunden, gibt Natural eine Fehlermeldung aus.</p> <p>Das ausgeführte Programm wird in den Arbeitsbereich des Programm-Editors gelesen und überschreibt dabei den vorherigen Inhalt des Arbeitsbereichs.</p>
<i>operand2</i>	<p>Parameter:</p> <p>Mit dem RUN-Statement können Parameter an das Programm, das ausgeführt werden soll, übergeben werden.</p> <p>Die Parameter können mit beliebigem Format definiert werden; sie werden automatisch in Formate umgesetzt, die zu den entsprechenden INPUT-Feldern passen. Alle angegebenen Parameter werden oben auf dem Natural-Stack abgelegt. Parameter können von einem INPUT-Statement gelesen werden. Das erste INPUT-Statement fügt alle Parameter in die im INPUT-Statement angegebenen Felder ein. Bei numerischen Feldern muss der Vorzeichen-Parameter SG auf ON gesetzt werden.</p> <p>Werden mehr Parameter übergeben als INPUT-Felder vorhanden sind, so werden überschüssige Parameter ignoriert. Die Anzahl der Parameter kann über die Systemvariable *DATA ermittelt werden.</p> <p>Anmerkung: Wenn <i>operand2</i> eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.</p>
<i>parameter</i>	<p>Wenn <i>operand2</i> eine Datumsvariable ist, können Sie den Session- Parameter DF als <i>parameter</i> für diese Variable angeben.</p>

Dynamische Sourcecode-Generierung und -Ausführung

Das RUN-Statement kann dazu verwendet werden, ein Programm dynamisch zu kompilieren und auszuführen, dessen Source ganz oder teilweise dynamisch erstellt wird.

Dynamische Sourcecode-Generierung erfolgt, indem man Sourcetext in globalen Variablen unterbringt, und diese Variablen dadurch referenziert, dass man im Sourcecode als erstes Zeichen im Variablennamen ein Pluszeichen (+) jeweils durch ein Und-Zeichen (&) ersetzt.

Wird das Programm mit RUN aufgerufen, so wird der Inhalt der globalen Variablen als Sourcecode interpretiert. Eine globale Variable mit Index darf nicht in einem mit RUN ausgeführten Programm verwendet werden.

Eine globale Variable darf keinen Kommentar und kein INCLUDE-Statement enthalten.

Beispiel

Programm mit RUN-Statement:

```
** Example 'RUNEX1': RUN (with dynamic source program creation)
*****
DEFINE DATA
GLOBAL
  USING RUNEXGDA
LOCAL
1 #NAME (A20)
1 #CITY (A20)
END-DEFINE
*
INPUT 'Please specify the search values:' //
  'Name:' #NAME /
  'City:' #CITY
*
RESET +CRITERIA      /* defined in GDA 'RUNEXGDA'
*
IF #NAME = ' ' AND #CITY = ' '
  REINPUT 'Enter at least 1 value'
END-IF
*
IF #NAME NE ' '
  COMPRESS 'NAME' ' '='' #NAME '''' INTO +CRITERIA LEAVING NO
END-IF
IF #CITY NE ' '
  IF +CRITERIA NE ' '
    COMPRESS +CRITERIA 'AND' INTO +CRITERIA
  END-IF
  COMPRESS +CRITERIA ' CITY =' ' ' #CITY '''' INTO +CRITERIA LEAVING NO
END-IF
*
RUN 'RUNEXFND'
*
END
```

Programm RUNEXFND, das per RUN-Statement ausgeführt wird:

```

** Example 'RUNEXFND': RUN (program executed with RUN in RUNEX1)
*****
DEFINE DATA
GLOBAL
  USING RUNEXGDA
LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
END-DEFINE
*
* &CRITERIA filled with "NAME = 'xxxxx' AND CITY = 'xxxx'"
*
FIND NUMBER EMPLOY-VIEW WITH &CRITERIA
  RETAIN AS 'EMP-SET'
DISPLAY *NUMBER
*
END

```

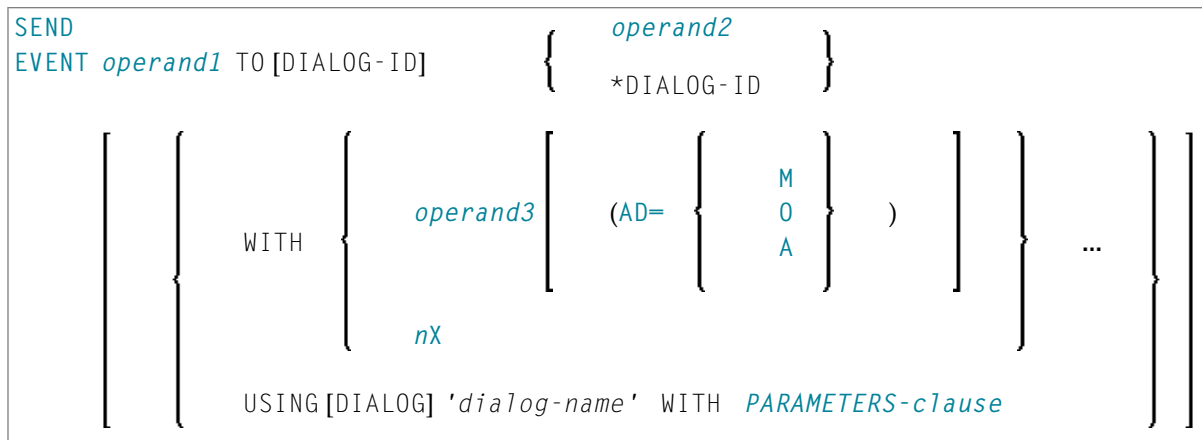
Global Data Area RUNEXGDA:

Global	RUNEXGDA	Library	SYSEXSYN	DBID	10	FNR	32
Command							> +
I T L	Name		F Length	Miscellaneous			
All	----->						
	1 +CRITERIA		A	80			

115

SEND EVENT

- Funktion 774
- Syntax-Beschreibung 774
- Weitere Informationen und Beispiele 776



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [OPEN DIALOG](#) | [CLOSE DIALOG](#) | [PROCESS GUI](#)

Gehört zur Funktionsgruppe: [Ereignisgesteuerte Programmierung](#)

Funktion

Das SEND EVENT-Statement dient dazu, Benutzerereignisse (Events) in einer Natural-Anwendung auszulösen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	ja	nein
<i>operand2</i>	S	I	ja	nein
<i>operand3</i>	C S A	A N P I F B D T L C G O	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	Operanden:	
	<i>operand1</i> ist der Name des zu sendenden Ereignisses.	
<i>operand2</i>	<i>operand2</i> ist der Bezeichner (Identifier) des Dialoges, der das Benutzerereignis empfängt. <i>operand2</i> muss mit Format/Länge I4 definiert werden.	
<i>operand3</i>	Übergabe von Parametern an den Dialog:	
	Es ist möglich, Parameter an den Dialog zu übergeben.	
	Als <i>operand3</i> geben Sie die Parameter an, die an den Dialog übergeben werden sollen.	
	Mit der <i>PARAMETERS-Klausel</i> können Parameter selektiv übergeben werden. Siehe PARAMETERS-Klausel unten.	
AD=	Definition von Attributen:	
	Wenn <i>operand3</i> eine Variable ist, können Sie sie folgendermaßen kennzeichnen:	
	AD=O	Nicht änderbar, siehe Session-Parameter AD=O.
	AD=M	Änderbar, siehe Session-Parameter AD=M. Dies ist die Standardeinstellung.
	AD=A	Nur Eingabe, siehe Session-Parameter AD=A.
	<i>operand3</i> kann nicht explizit angegeben werden, wenn <i>operand3</i> eine Konstante ist. AD=O gilt immer für Konstanten.	
<i>nX</i>	Angabe zu überspringender Parameter:	
	Mit der Notation <i>nX</i> können Sie angeben, dass die nächsten <i>n</i> Parameter übersprungen werden (zum Beispiel 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen); dies bedeutet, dass für die nächsten <i>n</i> Parameter keine Werte an den Dialog übergeben werden.	
	Ein zu überspringender Parameter muss im DEFINE DATA PARAMETER -Statement des Dialogs mit dem Schlüsselwort OPTIONAL definiert werden. OPTIONAL bedeutet, dass ein Wert von dem aufrufenden Objekt an einen solchen Parameter übergeben werden kann - aber nicht muss.	

PARAMETERS-Klausel

```
PARAMETERS {parameter-name=operand3} ...
END-PARAMETERS
```



Anmerkung: Sie können Sie *PARAMETERS-Klausel* nur benutzen, wenn der angegebene Zieldialog (*dialog-name*) katalogisiert ist.

dialog-name ist der Name des Dialogs, der das Benutzerereignis empfängt.



Anmerkung: Es führt zu einem Laufzeitfehler, wenn der Wert eines Parameters, der mit AD=0 markiert und „By Reference“ übergeben wird, im Dialog geändert wird.

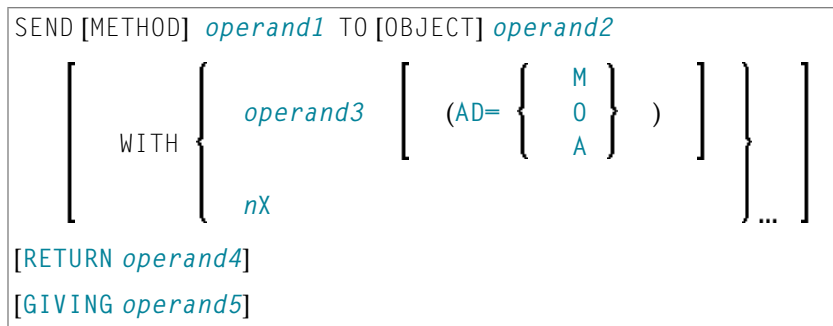
Weitere Informationen und Beispiele

Siehe *Event-Driven Programming Techniques* im Leitfaden zur Programmierung.

116

SEND METHOD

▪ Funktion	778
▪ Syntax-Beschreibung	778
▪ Beispiel	781



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `CREATE OBJECT` | `DEFINE CLASS` | `INTERFACE` | `METHOD` | `PROPERTY`

Gehört zur Funktionsgruppe: *Komponentenbasierte Programmierung*

Funktion

Das `SEND METHOD`-Statement dient dazu, eine bestimmte Methode eines Objekts aufzurufen. Informationen zur komponentenbasierten Programmierung, siehe *NaturalX im Leitfaden zur Programmierung*.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition			
<i>operand1</i>	C	S				A													ja	nein
<i>operand2</i>		S																O	nein	nein
<i>operand3</i>	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
<i>operand4</i>		S	A			A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
<i>operand5</i>		S			N					I									ja	nein

Format C und G kann nur an Methoden lokaler Klassen übergeben werden. Weitere Informationen siehe Abschnitt *Local Classes*.

Syntax-Element-Beschreibung:

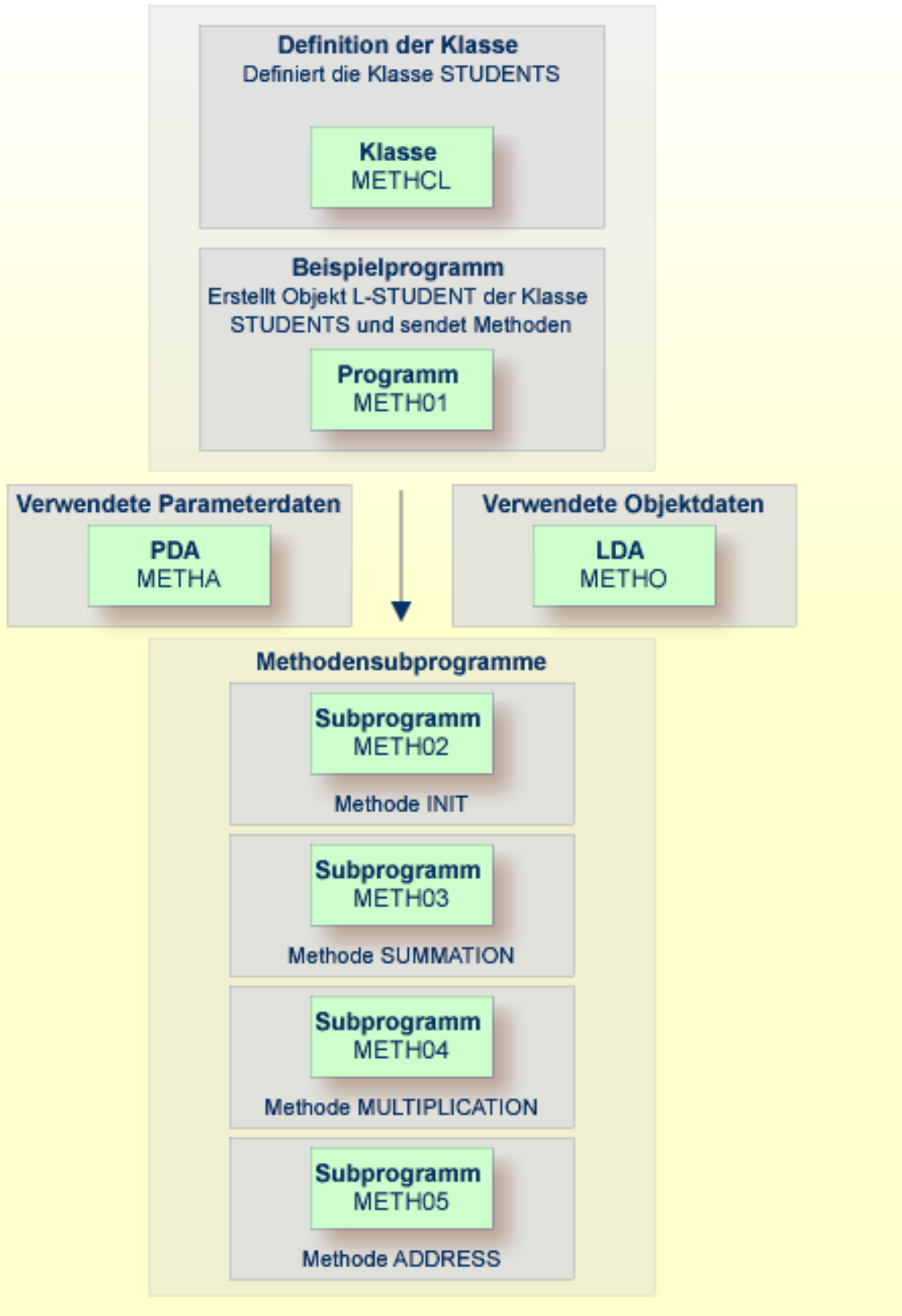
<i>operand1</i>	<p>Method-Name:</p> <p><i>operand1</i> ist der Name einer Methode, die vom in <i>operand2</i> angegebenen Objekt unterstützt wird.</p> <p>Da die Methoden-Namen in unterschiedlichen Interfaces einer Klasse identisch sein können, kann der Methoden-Name in <i>operand1</i> auch mit dem Interface-Namen versehen werden, um Mehrdeutigkeiten zu vermeiden.</p> <p>Im folgenden Beispiel hat das Objekt #03 ein Interface <i>Iterate</i> mit der Methode <i>Start</i>. Es gelten die folgenden Statements:</p> <pre style="background-color: #f0f0f0; padding: 5px;">* Specifying only the method name. SEND 'Start' TO #03 * Qualifying the method name with the interface name. SEND 'Iterate.Start' TO #03</pre> <p>Wenn kein Interface-Name angegeben wird, sucht Natural den Methoden-Namen in allen Interfaces der Klasse. Wenn der Methoden-Name in mehr als einem Interface gefunden wurde, tritt ein Laufzeitfehler auf.</p>
<i>operand2</i>	<p>Object-Handle:</p> <p>Die Handle des Objekts, an das der Aufruf der Methode gesendet werden soll.</p> <p><i>operand2</i> muss als Objekt-Handle (HANDLE OF OBJECT) definiert werden. Das Objekt muss bereits vorhanden sein.</p> <p>Um eine Methode des aktuellen Objekts innerhalb einer Methode aufzurufen, verwenden Sie die Systemvariable *THIS-OBJECT.</p>
<i>operand3</i>	<p>Methodenspezifische Parameter:</p> <p>Als <i>operand3</i> können Sie Parameter angeben, die methodenspezifisch sind.</p> <p>Im folgenden Beispiel hat das Objekt #03 die Methode <i>PositionTo</i> mit dem Parameter <i>Pos</i>. Die Methode wird wie folgt aufgerufen:</p> <pre style="background-color: #f0f0f0; padding: 5px;">SEND 'PositionTo' TO #03 WITH Pos</pre> <p>Methoden können optionale Parameter haben. Optionale Parameter brauchen nicht angegeben zu werden, wenn die Methode aufgerufen wird. Um einen optionalen Parameter wegzulassen, verwenden Sie den Platzhalter 1X. Um <i>n</i> optionale Parameter wegzulassen, verwenden Sie den Platzhalter <i>nX</i>.</p>

	<p>Im folgenden Beispiel hat die Methode <code>SetAddress</code> des Objekts #04 die Parameter <code>FirstName</code> (Vorname), <code>MiddleInitial</code> (Mittlere Initiale), <code>LastName</code> (Nachname), <code>Street</code> (Straße) und <code>City</code> (Stadt), wobei <code>MiddleInitial</code>, <code>Street</code> und <code>City</code> optional sind. Es gelten die folgenden Statements:</p> <pre> * Specifying all parameters. SEND 'SetAddress' TO #04 WITH FirstName MiddleInitial LastName Street City * Omitting one optional parameter. SEND 'SetAddress' TO #04 WITH FirstName 1X LastName Street City * Omitting all optional parameters. SEND 'SetAddress' TO #04 WITH FirstName 1X LastName 2X </pre> <p>Wenn ein Pflichtparameter weggelassen wird, führt dies zu einem Laufzeitfehler.</p>						
<p>AD=</p>	<p>Attribut-Definition: Wenn <i>operand3</i> eine Variable ist, können Sie sie wie folgt kennzeichnen:</p> <table border="1" data-bbox="279 751 1380 982"> <tr> <td data-bbox="279 751 821 831">AD=O</td> <td data-bbox="821 751 1380 831">Nicht modifizierbar, siehe Session-Parameter AD=O.</td> </tr> <tr> <td data-bbox="279 831 821 940">AD=M</td> <td data-bbox="821 831 1380 940">Modifizierbar, siehe Session-Parameter AD=M. Dies ist die Voreinstellung.</td> </tr> <tr> <td data-bbox="279 940 821 982">AD=A</td> <td data-bbox="821 940 1380 982">Nur für Eingabe, siehe Session-Parameter AD=A.</td> </tr> </table> <p>Wenn <i>operand3</i> eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.</p>	AD=O	Nicht modifizierbar, siehe Session-Parameter AD=O.	AD=M	Modifizierbar, siehe Session-Parameter AD=M. Dies ist die Voreinstellung.	AD=A	Nur für Eingabe, siehe Session-Parameter AD=A.
AD=O	Nicht modifizierbar, siehe Session-Parameter AD=O.						
AD=M	Modifizierbar, siehe Session-Parameter AD=M. Dies ist die Voreinstellung.						
AD=A	Nur für Eingabe, siehe Session-Parameter AD=A.						
<p><i>nX</i></p>	<p>Zu überspringende Parameter:</p> <p>Mit der Notation <i>nX</i> können Sie angeben, dass die nächsten <i>n</i> Parameter übersprungen werden sollen (zum Beispiel 1X, um den nächsten Parameter zu überspringen, oder 3X, um die nächsten drei Parameter zu überspringen). Dies bedeutet, dass für die nächsten <i>n</i> Parameter an die Methode keine Werte übergeben werden.</p> <p>Bei einer in Natural implementierten Methode muss ein zu überspringender Parameter mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Subprogramms der Methode definiert sein. OPTIONAL bedeutet, dass ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann, aber nicht unbedingt muss.</p>						
<p>RETURN <i>operand4</i></p>	<p>RETURN-Klausel:</p> <p>Wenn die RETURN-Klausel weggelassen wird und die Methode einen Rückgabewert hat, wird der Rückgabewert nicht berücksichtigt.</p> <p>Wenn die RETURN-Klausel angegeben wird, enthält <i>operand4</i> den Rückgabewert der Methode. Wenn die Ausführung der Methode ohne Erfolg abgebrochen wird, wird <i>operand4</i> auf seinen ursprünglichen Wert zurückgesetzt.</p> <p>Anmerkung: Bei in Natural geschriebenen Klassen wird der Rückgabewert einer Methode durch Eingabe eines zusätzlichen Parameters in der Parameter Data Area der Methode und durch Kennzeichnung mit BY VALUE RESULT definiert. Weitere Informationen siehe Abschnitt PARAMETER-Klausel. Deshalb enthält die Parameter Data Area einer Methode, die in Natural</p>						

	geschrieben ist, und die einen Rückgabewert hat, neben den Methoden-Parametern immer ein zusätzliches Feld. Dies ist zu berücksichtigen, wenn Sie eine Methode einer in Natural geschriebenen Klasse aufrufen und die Parameter Data Area der Methode im SEND-Statement verwenden möchten.
GIVING <i>operand5</i>	<p>GIVING-Klausel:</p> <p>Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt.</p> <p>Wenn die GIVING-Klausel angegeben wird, enthält <i>operand5</i> die Natural-Meldungsnummer, wenn ein Fehler aufgetreten ist, oder Null (0), wenn kein Fehler aufgetreten ist.</p>

Beispiel

Das folgende Diagramm gibt eine Übersicht über die Programmierobjekte, die in diesem Beispiel benutzt werden. Der entsprechende Sourcecode und die Programm-Ausgabe sind im Folgenden veranschaulicht



Programm METH01: CTREATE OBJECT und SEND METHOD mit einer Klasse und mehreren Methoden:

```

** Example 'METH01': CREATE OBJECT and SEND METHOD
**                      using a class and several methods (see METH*)
*****
DEFINE DATA
LOCAL
  USING METHA
LOCAL
1 L-STUDENT HANDLE OF OBJECT
1 #NAME      (A20)
1 #STREET    (A20)
1 #CITY      (A20)
1 #SUM       (I4)
1 #MULTI     (I4)
END-DEFINE
*
CREATE OBJECT L-STUDENT OF CLASS 'STUDENTS' /* see METHCL for class
*
L-STUDENT.<> := 'John Smith'
*
SEND METHOD 'INIT' TO L-STUDENT             /* see METHCL
  WITH #VAR1 #VAR2 #VAR3 #VAR4
*
SEND METHOD 'SUMMATION' TO L-STUDENT        /* see METHCL
  WITH #VAR1 #VAR2 #VAR3 #VAR4
*
SEND METHOD 'MULTIPLICATION' TO L-STUDENT  /* see METHCL
  WITH #VAR1 #VAR2 #VAR3 #VAR4
*
#NAME := L-STUDENT.<>
#SUM  := L-STUDENT.<>
#MULTI := L-STUDENT.<>
*
SEND METHOD 'ADDRESS' TO L-STUDENT         /* see METHCL
*
#STREET := L-STUDENT.<>
#CITY   := L-STUDENT.<>
*
*
WRITE 'Name   :' #NAME
WRITE 'Street:' #STREET
WRITE 'City   :' #CITY
WRITE ' '
WRITE 'The summation of      ' #VAR1 #VAR2 #VAR3 #VAR4
WRITE 'is' #SUM
WRITE 'The multiplication of' #VAR1 #VAR2 #VAR3 #VAR4
WRITE 'is' #MULTI
*
END

```

Vom Programm METH01 benutzte Klassen-Definition METHCL:

```
** Example 'METHCL': DEFINE CLASS (used by METH01)
*****
* Defining class STUDENTS for METH01
*
DEFINE CLASS STUDENTS
  OBJECT
    USING METH0          /* Object data for class STUDENTS
  /*
  INTERFACE STUDENT-ARITHMETICS
    PROPERTY FULL-NAME
      IS NAME
    END-PROPERTY
    PROPERTY SUM
    END-PROPERTY
    PROPERTY MULTI
    END-PROPERTY
*
  METHOD INIT
    IS METH02
    PARAMETER USING METHA
  END-METHOD
  METHOD SUMMATION
    IS METH03
    PARAMETER USING METHA
  END-METHOD
  METHOD MULTIPLICATION
    IS METH04
    PARAMETER USING METHA
  END-METHOD
END-INTERFACE
*
INTERFACE STUDENT-ADDRESS
  PROPERTY STUDENT-NAME
    IS NAME
  END-PROPERTY
  PROPERTY STREET
  END-PROPERTY
  PROPERTY CITY
  END-PROPERTY
*
  METHOD ADDRESS
    IS METH05
  END-METHOD
END-INTERFACE
END-CLASS
END
```

Local Data Area METHO (Objektdaten), die von der Klasse METHCL und den Subprogrammen METH02, METH03, METH04 und METH05 benutzt wird:

Local Command	METHO	Library SYSEXSYN	DBID	10 FNR	32
I T L	Name	F Length	Miscellaneous		
All	----->	----->	----->		
1	NAME	A 20			
1	STREET	A 30			
1	CITY	A 20			
1	SUM	I 4			
1	MULTI	I 4			

Parameter Data Area METHA, die vom Programm METH01, der Klasse METHCL und den Subprogrammen METH02, METH03 und METH04 benutzt wird:

Parameter Command	METHA	Library SYSEXSYN	DBID	10 FNR	32
I T L	Name	F Length	Miscellaneous		
All	----->	----->	----->		
1	#VAR1	I 4			
1	#VAR2	I 4			
1	#VAR3	I 4			
1	#VAR4	I 4			

Subprogramm METH02 - vom Programm METH01 verwendete Methode INIT:

```

** Example 'METH02': Method INIT (used by METH01)
*****
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method INIT of class STUDENTS
*
#VAR1 := 1
#VAR2 := 2
#VAR3 := 3
#VAR4 := 4
*
END

```

Subprogramm METH03 - vom Programm METH01 verwendete Methode SUMMATION:

```
** Example 'METH03': Method SUMMATION (used by METH01)
*****
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method SUMMATION of class STUDENTS
*
COMPUTE SUM = #VAR1 + #VAR2 + #VAR3 + #VAR4
END
```

Subprogramm METH04 - vom Programm METH01 verwendete Methode MULTIPLICATION:

```
** Example 'METH04': Method MULTIPLICATION (used by METH01)
*****
DEFINE DATA
PARAMETER
  USING METHA
OBJECT
  USING METHO
END-DEFINE
*
* Method MULTIPLICATION of class STUDENTS
*
COMPUTE MULTI = #VAR1 * #VAR2 * #VAR3 * #VAR4
END
```

Subprogramm METH05 - vom Programm METH01 verwendete Methode ADDRESS:

```
** Example 'METH05': Method ADDRESS (used by METH01)
*****
DEFINE DATA
  OBJECT USING METHO
END-DEFINE
*
* Method ADDRESS of class STUDENTS
*
IF NAME = 'John Smith'
  STREET := 'Oxford street'
  CITY   := 'London'
END-IF
END
```

Ausgabe des Programms METH01:

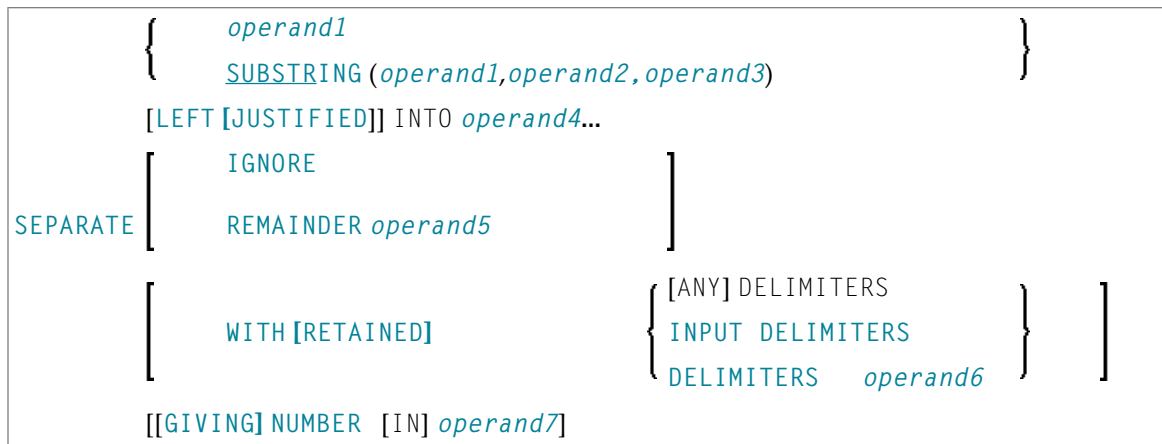
Page 1 05-01-17 15:59:04

Name : John Smith
Street: Oxford street
City : London

The summation of	1	2	3	4
is	10			
The multiplication of	1	2	3	4
is	24			

117 SEPARATE

▪ Funktion	790
▪ Syntax-Beschreibung	790
▪ Beispiele	794



Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: COMPRESS | COMPUTE | EXAMINE | MOVE | MOVE ALL | RESET

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Das Statement SEPARATE dient dazu, den Inhalt eines alphanumerischen oder binären Operanden auf zwei oder mehr alphanumerische oder binäre Operanden (oder auf mehrere Ausprägungen eines alphanumerischen oder binären Arrays) zu verteilen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate								Referenzierung erlaubt	Dynam. Definition	
operand1	C	S			A	U				B				ja	nein
operand2	C	S					N	P	I	B*				ja	nein
operand3	C	S					N	P	I	B*				ja	nein
operand4		S	A	G	A	U				B				ja	ja
operand5		S			A	U				B				ja	ja
operand6	C	S			A	U				B				ja	nein

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand7</i>	S	N P I	ja	ja

* Format B von *operand2* und *operand3* können nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Ausgangsoperand:</p> <p><i>operand1</i> ist die alphanumerische/binäre Konstante oder Variable, deren Inhalt aufgeteilt werden soll.</p> <p>Nachgestellte Leerzeichen in <i>operand1</i> werden entfernt, bevor der Wert verarbeitet wird (auch wenn das Leerzeichen als Begrenzungszeichen verwendet wird; vgl. DELIMITERS-Option).</p>
SUBSTRING	<p>SUBSTRING-Option:</p> <p>Normalerweise wird der ganze Inhalt des Feldes aufgeteilt, und zwar vom Anfang des Feldes an.</p> <p>Die SUBSTRING-Option ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes aufzuteilen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (<i>operand1</i>) zunächst die erste Stelle (<i>operand2</i>) und dann die Länge (<i>operand3</i>) des Feldteils an, der aufgeteilt werden soll. Wenn z.B. ein Feld #A den Wert CONTRAPTION enthält, würde SUBSTRING(#A, 5, 3) den Wert RAP enthalten.</p> <p>Normalerweise wird der ganze Inhalt des Feldes aufgeteilt, und zwar vom Anfang des Feldes an.</p> <p>Anmerkung: Wenn Sie <i>operand2</i> weglassen, wird ab Anfang des Feldes (Position 1) aufgeteilt. Wenn Sie <i>operand3</i> weglassen, wird ab der angegebenen Stelle (<i>operand2</i>) bis zum Ende des Feldes aufgeteilt.</p>
LEFT JUSTIFIED	<p>LEFT JUSTIFIED-Option:</p> <p>Diese Option bewirkt, dass den aufgeteilten Feldwertteilen vorangestellte Leerzeichen aus den Zieloperanden entfernt werden.</p>

<i>operand4</i>	<p>Zieloperand:</p> <p><i>operand4</i> enthält die Zieloperanden, die die Teile des Ausgangsoperanden aufnehmen sollen. Wird als Zieloperand ein Array verwendet, wird es Ausprägung für Ausprägung mit den übertragenen Feldwertteilen gefüllt.</p> <p>Die Anzahl der Zieloperanden entspricht der Anzahl der Begrenzungszeichen (einschließlich nachgestellter Begrenzungszeichen) in <i>operand1</i>, plus 1.</p> <p>Ist <i>operand4</i> eine dynamische Variable, kann deren Länge mit der SEPARATE-Operation geändert werden. Die aktuelle Länge einer dynamischen Variable kann mittels der Systemvariable *LENGTH ermittelt werden.</p> <p>Allgemeine Informationen zu dynamischen Variablen finden Sie im Abschnitt <i>Dynamische und große Variablen benutzen im Leitfaden zur Programmierung</i>.</p>	
IGNORE / REMAINDER <i>operand5</i>	<p>IGNORE/REMAINDER-Option:</p> <p>Wenn Sie nicht genug Zieloperanden angeben, um alle Feldwertteile aufzunehmen, erhalten Sie eine entsprechende Fehlermeldung.</p> <p>Um dies zu vermeiden, haben Sie zwei Möglichkeiten:</p> <ul style="list-style-type: none"> ■ Wenn Sie IGNORE angeben, ignoriert Natural es, falls nicht genügend Zieloperanden zur Aufnahme des Ausgangswertes vorhanden sind. ■ Wenn Sie REMAINDER <i>operand5</i> angeben, wird der Teil des Ausgangswertes, für den keine Zieloperanden mehr zur Verfügung stehen, in <i>operand5</i> gestellt. Den Inhalt von <i>operand5</i> können Sie dann weiter verarbeiten, zum Beispiel in einem weiteren SEPARATE-Statement. <p>Siehe auch Beispiel 3.</p>	
DELIMITERS	<p>DELIMITERS-Option:</p> <p>Siehe Delimiters-Option weiter unten.</p>	

WITH RETAINED DELIMITERS	<p>WITH RETAINED DELIMITERS-Option:</p> <p>Normalerweise werden die Begrenzungszeichen selbst nicht mit in die Zieloperanden übertragen.</p> <p>Wenn Sie allerdings <code>RETAINED</code> angeben, wird jedes Begrenzungszeichen (d.h. entweder das mit dem Session-Parameter <code>ID</code> festgelegte Standard-Begrenzungszeichen und Leerzeichen oder die mit <code>operand6</code> angegebenen Zeichen) ebenfalls in einen Zieloperanden übertragen.</p> <p>Beispiel:</p> <p>Das folgende <code>SEPARATE</code>-Statement überträgt 150 nach <code>#B</code>, das Plus-Zeichen (+) nach <code>#C</code> und 30 nach <code>#D</code>:</p> <pre>... MOVE '150+30' TO #A SEPARATE #A INTO #B #C #D WITH RETAINED DELIMITER '+' ...</pre> <p>Siehe auch Beispiel 3.</p>	
GIVING NUMBER <i>operand7</i>	<p>GIVING NUMBER-Option</p> <p>Diese Option bewirkt, dass die Anzahl der Zieloperanden, die mit einem Wert gefüllt wurden (einschließlich der mit Leerzeichen gefüllten), in <i>operand7</i> ausgegeben wird. Die Anzahl, die Sie erhalten, errechnet sich aus der Anzahl der Delimiterzeichen plus 1.</p> <p>Wenn Sie die <code>IGNORE</code>-Option verwenden, enthält <i>operand7</i> maximal die Anzahl der Zieloperanden (<i>operand4</i>).</p> <p>Wenn Sie die <code>REMAINDER</code>-Option verwenden, enthält <i>operand7</i> maximal die Anzahl der Zieloperanden (<i>operand4</i>) plus <i>operand5</i>.</p>	

DELIMITERS-Option:

WITH [RETAINED]	{ [ANY] DELIMITERS INPUT DELIMITERS DELIMITERS <i>operand6</i> }
------------------------	---

Begrenzungszeichen innerhalb von *operand1* bestimmen die Stellen, an denen der Wert geteilt werden soll.

- Falls Sie die `DELIMITERS`-Option nicht angeben (oder `WITH ANY DELIMITER` angeben), wird jedes Leerzeichen sowie jedes Zeichen, das weder ein Buchstabe noch eine Ziffer ist, als Begrenzungszeichen interpretiert.

- `WITH INPUT DELIMITERS` bedeutet, dass das mit dem Session-Parameter `ID` definierte Zeichen sowie das Leerzeichen als Begrenzungszeichen gelten.
- `WITH DELIMITERS operand6` bedeutet, dass jedes der angegebenen Zeichen (`operand6`) als Begrenzungszeichen interpretiert wird. Wenn `operand6` nachgestellte Leerzeichen enthält, werden diese ignoriert.

Beispiele

- [Beispiel 1 — Verschiedene Beispiele für den Gebrauch des SEPARATE-Statements](#)
- [Beispiel 2 — SEPARATE-Statement bei einem Array](#)
- [Beispiel 3 — Gebrauch der Optionen REMAINDER/RETAINED](#)

Beispiel 1 — Verschiedene Beispiele für den Gebrauch des SEPARATE-Statements

```

** Example 'SEPEX1': SEPARATE
*****
DEFINE DATA LOCAL
1 #TEXT1   (A6) INIT <'AAABBB'>
1 #TEXT2   (A7) INIT <'AAA BBB'>
1 #TEXT3   (A7) INIT <'AAA-BBB'>
1 #TEXT4   (A7) INIT <'A.B/C,D'>
1 #FIELD1A (A6)
1 #FIELD1B (A6)
1 #FIELD2A (A3)
1 #FIELD2B (A3)
1 #FIELD3A (A3)
1 #FIELD3B (A3)
1 #FIELD4A (A3)
1 #FIELD4B (A3)
1 #FIELD4C (A3)
1 #FIELD4D (A3)
1 #NBT     (N1)
1 #DEL     (A5)
END-DEFINE
*
WRITE NOTITLE 'EXAMPLE A (SOURCE HAS NO BLANKS)'
SEPARATE #TEXT1 INTO #FIELD1A #FIELD1B GIVING NUMBER #NBT
WRITE      / '=' #TEXT1 5X '=' #FIELD1A 4X '=' #FIELD1B 4X '=' #NBT
*
WRITE NOTITLE /// 'EXAMPLE B (SOURCE HAS EMBEDDED BLANK)'
SEPARATE #TEXT2 INTO #FIELD2A #FIELD2B GIVING NUMBER #NBT
WRITE      / '=' #TEXT2 4X '=' #FIELD2A 7X '=' #FIELD2B 7X '=' #NBT
*
WRITE NOTITLE /// 'EXAMPLE C (USING DELIMITER ''-'')'
SEPARATE #TEXT3 INTO #FIELD3A #FIELD3B WITH DELIMITER '-'
WRITE      /      '=' #TEXT3 4X '=' #FIELD3A 7X '=' #FIELD3B

```

```

*
MOVE ',/' TO #DEL
WRITE NOTITLE /// 'EXAMPLE D USING DELIMITER' '=' #DEL
*
SEPARATE #TEXT4 INTO #FIELD4A #FIELD4B
                #FIELD4C #FIELD4D WITH DELIMITER #DEL
WRITE      /      '=' #TEXT4 4X '=' #FIELD4A 7X '=' #FIELD4B
           /
           19X '=' #FIELD4C 7X '=' #FIELD4D
*
END

```

Ausgabe des Programms SEPEX1:

```

EXAMPLE A (SOURCE HAS NO BLANKS)

#TEXT1: AAABBB      #FIELD1A: AAABBB      #FIELD1B:           #NBT:  1

EXAMPLE B (SOURCE HAS EMBEDDED BLANK)

#TEXT2: AAA BBB     #FIELD2A: AAA          #FIELD2B: BBB         #NBT:  2

EXAMPLE C (USING DELIMITER '-')

#TEXT3: AAA-BBB     #FIELD3A: AAA          #FIELD3B: BBB

EXAMPLE D USING DELIMITER #DEL: ,/

#TEXT4: A.B/C,D     #FIELD4A: A.B         #FIELD4B: C
                   #FIELD4C: D          #FIELD4D:

```

Beispiel 2 — SEPARATE-Statement bei einem Array

```

** Example 'SEPEX2': SEPARATE (using array variable)
*****
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VALUE1, VALUE2,VALUE3'>
1 #FIELD      (A20/1:5)
1 #NUMBER     (N2)
END-DEFINE
*
SEPARATE #INPUT-LINE LEFT JUSTIFIED INTO #FIELD (1:5)
                GIVING NUMBER IN #NUMBER
*

```

SEPARATE

```
WRITE NOTITLE #INPUT-LINE //
              #FIELD (1) /
              #FIELD (2) /
              #FIELD (3) /
              #FIELD (4) /
              #FIELD (5) /
              #NUMBER
*
END
```

Ausgabe des Programms SEPEX2:

```
VALUE1,    VALUE2,VALUE3

VALUE1
VALUE2
VALUE3

3
```

Beispiel 3 — Gebrauch der Optionen REMAINDER/RETAINED

```
** Example 'SEPEX3': SEPARATE (with REMAINDER, RETAIN option)
*****
DEFINE DATA LOCAL
1 #INPUT-LINE (A60) INIT <'VAL1,  VAL2, VAL3,VAL4'>
1 #FIELD      (A10/1:4)
1 #REM        (A30)
END-DEFINE
*
WRITE TITLE LEFT 'INP:' #INPUT-LINE /
            '#FIELD (1)' 13T '#FIELD (2)' 25T '#FIELD (3)'
            37T '#FIELD (4)' 49T 'REMAINDER'
            /  '-----' 13T '-----' 25T '-----'
            37T '-----' 49T '-----'
*
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
            REMAINDER #REM WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:2)
            IGNORE WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
SEPARATE #INPUT-LINE INTO #FIELD (1:4) IGNORE
            WITH RETAINED DELIMITERS ','
```



```

WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
RESET #FIELD(*) #REM
*
SEPARATE SUBSTRING(#INPUT-LINE,1,50) INTO #FIELD (1:4)
      IGNORE WITH DELIMITERS ','
WRITE #FIELD(1) 13T #FIELD(2) 25T #FIELD(3) 37T #FIELD(4) 49T #REM
*
END

```

Ausgabe des Programms SEPEX3:

```

INP: VAL1,  VAL2, VAL3,VAL4
#FIELD (1) #FIELD (2) #FIELD (3) #FIELD (4) REMAINDER
-----
VAL1          VAL2                      VAL3,VAL4
VAL1          VAL2
VAL1          ,          VAL2          ,
VAL1          ,  VAL2          VAL3          VAL4

```


118 SET CONTROL

▪ Funktion	800
▪ Syntax-Beschreibung	800
▪ Beispiele	800

```
SET CONTROL operand1 ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Mit dem Statement SET CONTROL können Sie ein Terminalkommando von einem Programm aus ausführen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Auszuführende Terminalkommandos:</p> <p>Das Terminalkommando (<i>operand1</i>) wird standardmäßig ohne das Kontrollzeichen (%) angegeben und kann als Textkonstante oder als Inhalt einer alphanumerischen Variablen angegeben werden.</p> <p>Informationen zu den einzelnen Terminalkommandos finden Sie in der <i>Terminalkommandos</i>-Dokumentation.</p>
-----------------	--

Beispiele

- [Beispiel 1 — Umschalten auf Kleinschreibung](#)

- Beispiel 2 — Hardcopy-Ausgabeziel aktivieren

Beispiel 1 — Umschalten auf Kleinschreibung

```
...  
SET CONTROL 'L'  
...
```

Schaltet die automatische Umsetzung von Klein- in Großbuchstaben aus (entspricht dem Terminalkommando %L).

Beispiel 2 — Hardcopy-Ausgabeziel aktivieren

```
...  
SET CONTROL 'HDEST'  
...
```

Erzeugt bei der logischen Destination `DEST` eine Hardcopy-Ausgabe (entspricht dem Terminalkommando %H*destination*).

119 SET GLOBALS

▪ Funktion	804
▪ Parameter	804
▪ Beispiel	805

```
SET GLOBALS parameter=value ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Mit dem Statement `SET GLOBALS` können Sie Werte für Session-Parameter setzen.

Die Auswertung der Parameter erfolgt je nach Parameter entweder bei der Kompilierung oder bei der Ausführung des Programms, das das `SET GLOBALS`-Statement enthält. Dies ist abhängig von den einzelnen Parametern.

Die mit `SET GLOBALS` gesetzten Parameterwerte gelten für die ganze Natural-Session, sofern sie nicht durch ein weiteres `SET GLOBALS`-Statement (bzw. Systemkommando `GLOBALS`) geändert werden.

Mit dem Statement `SET GLOBALS` können Sie dieselben Parameter verwenden wie beim Systemkommando `GLOBALS`. Sie können sowohl das Statement als auch das Systemkommando in derselben Natural-Session benutzt werden.

Mit einem `GLOBALS`-Kommando angegebene Parameterwerte bleiben gültig, bis sie von einem neuen `GLOBALS`-Kommando oder `SET GLOBALS`-Statement geändert werden, die Session beendet wird oder Sie sich in einer anderen Library anmelden.

Parameter

Wenn Sie mehrere Parameter angeben, müssen Sie diese durch ein oder mehrere Leerzeichen voneinander trennen. Die Reihenfolge der Parameter ist beliebig. Siehe [Beispiel](#).

Mit dem SET GLOBALS-Statement spezifizierbare Parameter:	Auswertung
	R = zur Laufzeit
	C = bei der Kompilierung
CF - Character for Terminal Commands	R
CPCVERR - Code Page Conversion Error	R
DC - Character for Decimal Point Notation	R
DC - Character for Decimal Point Notation	R
DFOUT - Date Format for Output	R

Mit dem SET GLOBALS-Statement spezifizierbare Parameter:	Auswertung
	R = zur Laufzeit
	C = bei der Kompilierung
DFSTACK - Date Format for Stack	R
DFTITLE - Date Format in Default Page Title	R
DU - Dump Generation	R
EJ - Page Eject	R
FCDP - Filler Character for Dynamically Protected Fields	R
FS - Format Specification	R
IA - INPUT Assign Character	R
ID - INPUT Delimiter Character	R
IM - INPUT Mode	R
LE - Limit Error Processing	C
LS - Line Size	C
LT - Limit of Records Read	R
NC - Use of Natural System Commands	R
OPF - Overwriting of Protected Fields by Helproutines	R
PM - Print Mode	C
PS - Page Size	RC
REINP - Internal REINPUT for Invalid Data	R
SA - Sound Terminal Alarm	R
SF - Spacing Factor	C
WH - Wait for Record in Hold Status	R
ZD - Zero Division Check	R
ZP - Zero Printing	R

Informationen über die einzelnen Parameter finden Sie in der *Parameter-Referenz*.

Beispiel

In dem folgenden Beispiel wird das SET GLOBALS-Statement dazu benutzt, die maximale Anzahl der Zeichen pro Zeile auf 74 zu setzen und die Anzahl der Datensätze der Datenbank, die in Verarbeitungsschleifen in einem Natural- Programm gelesen werden können, auf 5000 zu begrenzen.

SET GLOBALS

```
SET GLOBALS LS=74 LT=5000  
...
```

120 SET KEY

▪ Funktion	808
▪ Syntax-Beschreibung	808
▪ Tasten programm-sensitiv machen und deaktivieren	809
▪ Kommandos/Programme einer Taste zuweisen	811
▪ Eingabedaten einer Taste zuweisen (DATA)	811
▪ Tastenfunktion vorübergehend deaktivieren	812
▪ Helproutine zuweisen (HELP)	813
▪ Dynamische Funktionszuweisung (DYNAMIC)	813
▪ GUI-Element-Zuweisung deaktivieren (DISABLED)	814
▪ SET KEY-Statements auf verschiedenen Programmebenen	814
▪ Namen zuweisen	816
▪ Beispiel	817

Dieses Kapitel behandelt folgende Themen:

Funktion

Das SET KEY-Statement dient dazu, den folgenden Tasten-Arten Funktionen zuzuweisen:

- Videoterminal PA-Tasten (Programmabrufstasten)
- PF-Tasten (Programmfunktionstasten)
- CLEAR- bzw. LÖSCH-Taste.

Wird ein SET KEY-Statement ausgeführt, erhält Natural während der Programmausführung die Kontrolle über diese Tasten, und zwar unter Verwendung der Werte, die den Tasten zugewiesen sind.

Über die Natural-Systemvariable *PF-KEY kann ermittelt werden, welche Taste zuletzt gedrückt wurde.



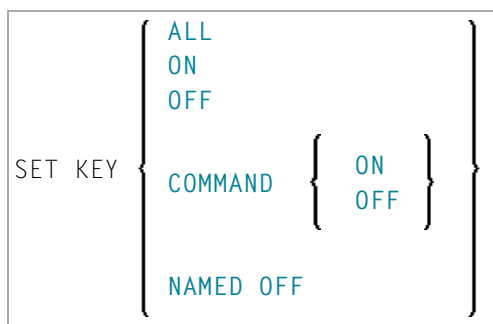
Anmerkung: Wird eine Taste gedrückt, der keine Funktion zugewiesen ist, wird der Benutzer entweder aufgefordert, eine andere Taste zu drücken, oder der Wert ENTR wird in die Natural-Systemvariable *PF-KEY gestellt, d.h. Natural reagiert, als ob die ENTER- bzw. FREIG-Taste gedrückt worden wäre (je nachdem, wie Ihr Natural-Administrator den Profilparameter IKEY gesetzt hat).

Syntax-Beschreibung

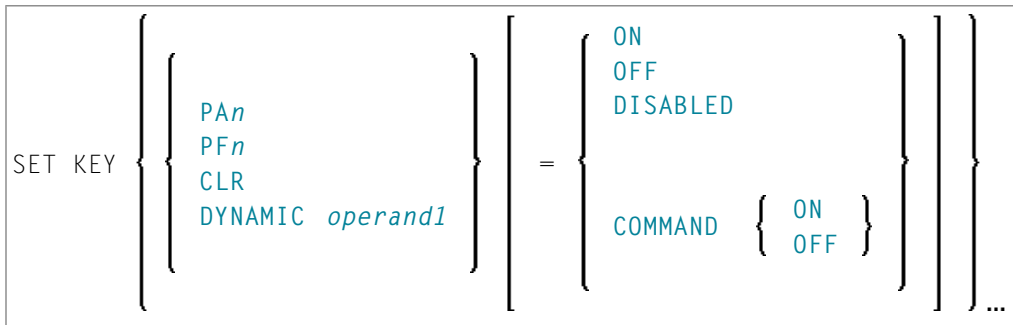
Mehrere Strukturen sind bei diesem Statement möglich.

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

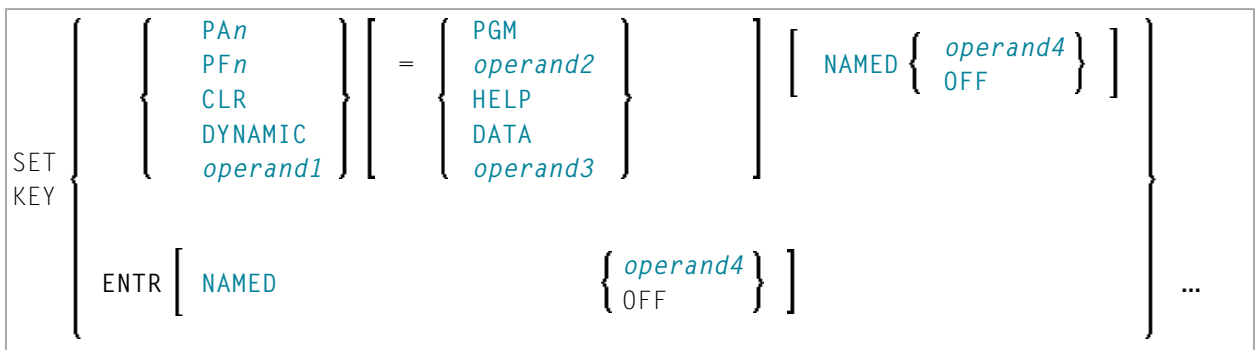
Syntax 1 – für alle Tasten



Syntax 2 – für einzelne Tasten



Syntax 3 – für einzelne Tasten




Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S	A	ja	nein
<i>operand2</i>	C S	A U	ja	nein
<i>operand3</i>	C S	A U	ja	nein
<i>operand4</i>	C S	A U	ja	nein

Tasten programm-sensitiv machen und deaktivieren

Wenn eine Taste programm-sensitiv gemacht ist, kann Sie vom gerade aktiven Programm abgefragt werden. Das Drücken einer programm-sensitiven Taste hat die gleiche Wirkung wie das Drücken der FREIG-Taste. Alle auf dem Bildschirm eingegebenen Daten werden an das Programm übergeben.

 **Anmerkung:** Beim Drücken einer programm-sensitiven PA- oder CLEAR- bzw. LÖSCH-Taste werden keine Daten vom Bildschirm an das Programm übergeben.

Die Programm-Sensitivität ist nur während der Ausführung des aktuellen Programms wirksam. Vgl. Abschnitt *SET KEY-Statements auf verschiedenen Programmebenen*.

Beispiele:

SET KEY ALL	Dieses Statement bewirkt, dass alle Tasten programm-sensitiv gemacht werden. Alle Funktionszuweisungen zu Tasten werden damit überschrieben.
SET KEY PF2 SET KEY PF2=PGM	Jedes dieser beiden Statements bewirkt, dass PF2 programm-sensitiv gemacht wird.
SET KEY OFF	Dieses Statement deaktiviert alle Funktionstastenbelegungen. Die Natural-Systemvariable *PF-KEY enthält ENTR, nachdem SET KEY OFF ausgeführt worden ist.
SET KEY ON	Dieses Statement reaktiviert die Funktionen aller Tasten, denen eine Funktion zugewiesen war, und macht alle Tasten, die vor der Deaktivierung programm-sensitiv waren, wieder programm-sensitiv.
SET KEY PF2=OFF	Dieses Statement deaktiviert PF2. Nach Ausführung von SET KEY PF2=OFF enthält die Natural-Systemvariable *PF-KEY ENTR, wenn sie vorher PF2 enthalten hatte.
SET KEY PF2=ON	Dieses Statement reaktiviert die Funktion, die PF2 zugewiesen wurde, bevor die Taste deaktiviert oder programm-sensitiv gemacht wurde. War PF2 keine Funktion zugewiesen, wird die Taste wieder programm-sensitiv gemacht.

Programm-Sensitivität einer Taste und Inhalt von *PF-KEY

Das folgende Beispiel zeigt die Beziehung zwischen der Programm-Sensitivität einer Taste und dem Inhalt der Systemvariablen *PF-KEY.

Gehen wir davon aus, dass PF2 mittels SET KEY PF2=PGM programm-sensitiv gemacht worden ist und dass ein INPUT-Statement im Anschluss daran ausgeführt wird. Die folgende Tabelle zeigt, wie Aktionen durch den Benutzer und ausgeführte Natural-Statements den Inhalt von *PF-KEY beeinflussen.

Reihenfolge	Aktion durch Benutzer / Natural-Statement ausgeführt	Inhalt der Systemvariablen *PF-KEY
1	Benutzer drückt PF2.	PF2
2	SET KEY OFF	ENTR
3	SET KEY ON	PF2
4	SET KEY PF2=OFF	ENTR
5	SET KEY PF2=ON	PF2
6	SET KEY PF3=OFF	PF2

Kommandos/Programme einer Taste zuweisen

Sie können einer Taste ein Kommando oder einen Programmnamen zuweisen. Wenn die Taste gedrückt wird, wird das aktuelle Programm unterbrochen und das der Taste zugewiesene Kommando/Programm über den Natural-Stack aufgerufen. Wenn Sie einer Taste ein Kommando/Programm zuweisen, können Sie auch Parameter an das Kommando/Programm übergeben (siehe drittes Beispiel unten).

Sie können einer Taste auch ein Terminalkommando zuweisen. Wenn die Taste gedrückt wird, wird das ihr zugewiesene Terminalkommando ausgeführt.

Wenn *operand2* als Konstante angegeben wird, muss diese in Apostrophen stehen.

Beispiele:

SET KEY PF4='SAVE'	Das Kommando <code>SAVE</code> wird PF4 zugewiesen.
SET KEY PF4=#XYZ	Der in der Variablen <code>#XYZ</code> enthaltene Wert wird PF4 zugewiesen.
SET KEY PF6='LIST MAP *'	Das Kommando <code>LIST</code> , einschließlich der <code>LIST</code> -Kommando-Parameter <code>MAP</code> und <code>*</code> wird PF6 zugewiesen.
SET KEY PF2='%%'	Das Terminalkommando <code>%%</code> wird PF2 zugewiesen.

Die Zuweisungen sind solange wirksam, bis sie mit einem anderen `SET KEY`-Statement überschrieben werden, der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird. Vgl. Abschnitt [SET KEY-Statements auf verschiedenen Programmebenen](#).



Anmerkung: Bevor ein über eine Taste aufgerufenes Programm ausgeführt wird, führt Natural intern ein `BACKOUT TRANSACTION`-Statement aus.

Eingabedaten einer Taste zuweisen (DATA)

Sie können einer Taste eine Datenkette (*operand3*) zuweisen. Wenn die Taste gedrückt wird, werden die Daten in das Eingabefeld übertragen, in dem der Cursor gerade steht, und werden dann an das ausführende Programm übergeben (als ob `FREIG` gedrückt worden wäre).

Wenn *operand3* als Konstante angegeben wird, muss diese in Apostrophen stehen.

Beispiel:

```
SET KEY PF12=DATA 'YES'
```

Für eine DATA-Zuweisung gilt dasselbe wie für eine Kommando-Zuweisung: sie ist solange wirksam, bis sie mit einem anderen SET KEY-Statement überschrieben wird, der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird. Vgl. Abschnitt [SET KEY-Statements auf verschiedenen Programmebenen](#).

Tastenfunktion vorübergehend deaktivieren

Mit COMMAND OFF können Sie eine Funktion (Kommando, Programm oder Daten), die einer Taste zugewiesen ist, vorübergehend außer Kraft setzen. Wenn die Taste vor der Zuweisung der Funktion programm-sensitiv war, macht COMMAND OFF sie wieder programm-sensitiv.

Mit einem anschließenden COMMAND ON können Sie die zugewiesene Funktion später wieder aktivieren.

Beispiele:

SET KEY PF4=COMMAND OFF	Die der Taste PF4 zugewiesene Funktion wird vorübergehend deaktiviert; war PF4 vor der Zuweisung der Funktion programm-sensitiv, wird die Taste jetzt wieder programm-sensitiv.
SET KEY PF4=COMMAND ON	Die der Taste PF4 zugewiesene Funktion wird wieder reaktiviert.
SET KEY COMMAND OFF	Die allen Tasten zugewiesenen Funktionen werden vorübergehend deaktiviert; waren Tasten vor der Zuweisung der jeweiligen Funktion programm-sensitiv, werden sie jetzt wieder programm-sensitiv.
SET KEY COMMAND ON	Die allen Tasten zugewiesenen Funktionen werden wieder reaktiviert.

Mit SET KEY PFnn='' können Sie die einer Taste zugewiesene Funktion löschen und gleichzeitig die Programm-Sensitivität der Taste deaktivieren.

Helproutine zuweisen (HELP)

Sie können einer Taste `HELP` zuweisen. Wenn die Taste gedrückt wird, wird die Helproutine aufgerufen, die dem Feld, in dem der Cursor gerade steht, zugewiesen ist.

Der Effekt ist derselbe wie beim Aufrufen von Hilfe durch Eingabe des Hilfe-Zeichens in das Feld. (Das Hilfe-Zeichen — standardmäßig ein Fragezeichen (?) — wird vom Natural-Administrator mit dem Natural-Profilparameter `HI` bestimmt.)

Beispiel:

```
SET KEY PF1=HELP
```

Für die `HELP`-Zuweisung gilt dasselbe wie für Programm-Sensitivität: sie gilt nur für die Ausführung des aktuellen Programms. Vgl. Abschnitt [SET KEY-Statements auf verschiedenen Programmebenen](#).

Dynamische Funktionszuweisung (DYNAMIC)

Anstatt im `SET KEY`-Statement eine bestimmte Taste anzugeben, können Sie die Option `DYNAMIC` mit Angabe einer Variablen (*operand1*) verwenden und dieser Variablen im Programm einen Wert (`PFn`, `PAn`, `CLR`) zuweisen. Dadurch haben Sie die Möglichkeit, eine Funktion anzugeben und es von der Programmlogik abhängig zu machen, welcher Taste diese Funktion zugewiesen wird.

Beispiel:

```
...
IF ...
  MOVE 'PF4' TO #KEY
ELSE
  MOVE 'PF7' TO #KEY
END-IF
...
SET KEY DYNAMIC #KEY = 'SAVE'
...
```

GUI-Element-Zuweisung deaktivieren (DISABLED)

Elemente graphischer Benutzeroberflächen (GUI) wie z.B. Drucktasten, Menüs und Bitmaps sind als PF-Tasten implementiert. Mit der Option `DISABLED` können Sie das einer PF-Taste zugewiesene GUI-Element deaktivieren. Das betreffende GUI-Element wird dann grau dargestellt und kann nicht ausgewählt werden.

Mit `SET KEY PFnn=ON` können Sie `SET KEY PFnn=DISABLED` wieder rückgängig machen.

Beispiel:

<code>SET KEY PF10=DISABLED</code>	Deaktiviert das PF10 zugewiesene GUI-Element.
------------------------------------	---

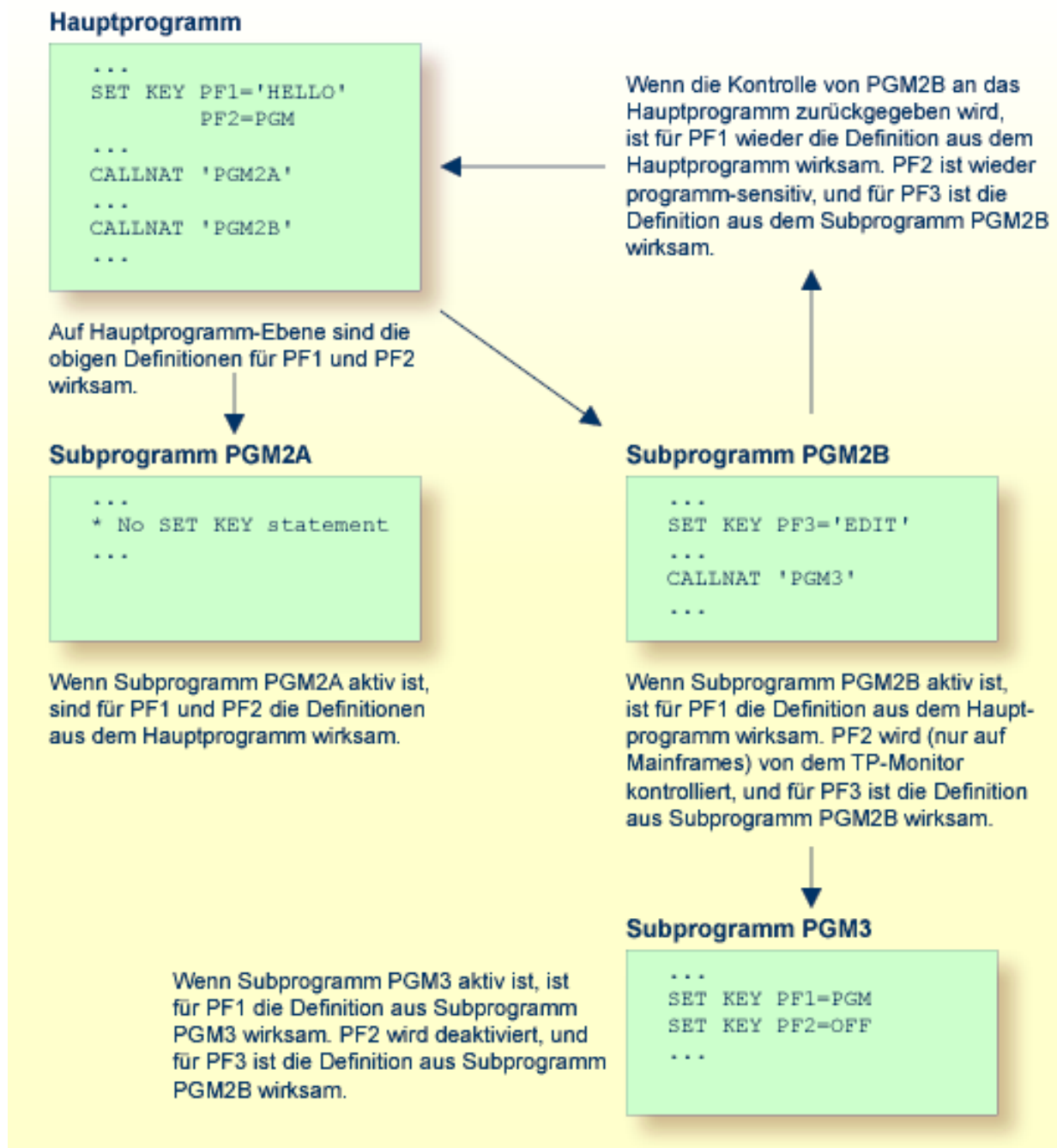
Die `DISABLED`-Option kann nur in Processing Rules verwendet werden.

SET KEY-Statements auf verschiedenen Programmebenen

Wenn eine Anwendung `SET KEY`-Statements auf verschiedenen Ebenen enthält, gilt folgendes:

- Wenn Tasten programm-sensitiv gemacht werden, gilt die Programm-Sensitivität auch für alle (aufgerufenen) Programme (bzw. Unterprogramme) auf untergeordneten Ebenen, es sei denn, diese Programme enthalten eigene `SET KEY`-Statements. Wenn die Kontrolle an ein übergeordnetes Programm zurückgegeben wird, gelten wieder die auf dieser übergeordneten Ebene gemachten `SET KEY`-Zuweisungen.
- Für Tasten, die als `HELP`-Tasten definiert sind, gilt das gleiche wie für programm-sensitive Tasten.
- Wenn einer Taste eine Funktion (Programm, Kommando, Terminalkommando oder Daten) zugewiesen wird, gilt diese Zuweisung für alle über- und untergeordneten Ebenen — ganz gleich, auf welcher Ebene die Zuweisung erfolgt —, und zwar solange, bis der Taste eine andere Funktion zugewiesen wird oder sie programm-sensitiv gemacht wird, oder bis der Benutzer in eine andere Anwendung wechselt oder die Natural-Session beendet wird.

Beispiel für SET KEY-Statements auf verschiedenen Programmebenen



Namen zuweisen

Mit der NAMED-Klausel können Sie einer Taste einen Namen (*operand4*) zuweisen. Dieser Name wird dann in der PF-Tastenleiste auf dem Bildschirm angezeigt, was es dem Benutzer ermöglicht, die den Tasten zugewiesenen Funktionen zu erkennen:

```

      ?  Help
      .  Exit
-----
Code ...: ?  Library ...: *_____
          Object ...: *_____
          DBID .....: 0__  FILENR ...: 0__

Command ==>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Exit  Last      Flip                                Canc
    
```

Die Anzeige der PF-Tastenleiste wird mit dem Session-Parameter KD aktiviert.

Der Name, den Sie einer Taste zuweisen, darf bis zu 10 Stellen lang sein. Im normalen tabellarischen PF-Tastenleistenformat werden jeweils nur die ersten 5 Stellen angezeigt.

Wenn Sie *operand4* als Konstante angeben, muss diese in Apostrophen stehen (siehe Beispiele weiter unten).

Sie können einer Taste keinen Namen geben, ohne ihr eine Funktion zuzuweisen oder sie programm-sensitiv zu machen. Der ENTER-Taste können Sie allerdings nur einen Namen (z.B. EINGABE) zuweisen, aber keine Funktion.

Mit NAMED OFF löschen Sie den Namen einer programm-sensitiven Taste.

Beispiele:

SET KEY ENTR NAMED 'EXEC'	Der ENTER-Taste wird der Name EXEC zugewiesen.
SET KEY PF3 NAMED 'EXIT'	PF3 wird programm-sensitiv gemacht und erhält den Namen EXIT.
SET KEY PF3 NAMED OFF	PF3 wird programm-sensitiv gemacht, und der PF3 zugewiesene Name wird gelöscht.
SET KEY NAMED OFF	Alle programm-sensitiven Tasten zugewiesenen Namen werden gelöscht.
SET KEY PF4='AP1' NAMED 'APPL1'	PF4 werden das Programm AP1 und der Name APPL1 zugewiesen.

Wenn Sie normales tabellarisches PF-Tastenleistenformat verwenden, gilt folgendes:

- Wenn Sie einer Taste ein Kommando/Programm zuweisen und die NAMED-Klausel weglassen, wird der Name dieses Kommandos/Programms in der PF-Tastenleiste angezeigt; ist der Name länger als 5 Stellen, wird stattdessen CMND angezeigt.
- Wenn Sie einer Taste Eingabedaten zuweisen und die NAMED-Klausel weglassen, wird als Name DATA in der PF-Tastenleiste angezeigt.
- Wenn Sie einer PF-Taste (per NAMED-Klausel) einen Namen im Unicode-Format zuweisen, kann es sein, dass der Name nicht richtig unter den entsprechenden Überschriften positioniert wird. Dieses Problem kann allerdings nur bei Verwendung des *Natural Web I/O Interface* und nur bei Zeichen vom Typ „wide“ auftreten. In diesem Fall wird empfohlen, das sequenzielle PF-Tastenformat zu verwenden .

Wenn Sie sequentielles PF-Tastenleistenformat verwenden, werden nur die Tasten in der PF-Tastenleiste angezeigt, denen sie Namen zugewiesen haben; d.h. wenn Sie einer Taste ein Kommando/Programm/Daten zuweisen und die NAMED-Klausel weglassen, erscheint die Taste nicht in der PF-Tastenleiste.

Siehe auch *Verarbeitung aufgrund der Namen von Funktionstasten im Leitfaden zur Programmierung*.

Beispiel

```

** Example 'SKYEX1': SET KEY
*****
DEFINE DATA LOCAL
1 #PF4 (A56)
END-DEFINE
*
MOVE 'LIST VIEW' TO #PF4
*
SET KEY PF1 PF2
SET KEY PF3 = 'MENU'
          PF4 = #PF4
          PF5 = 'LIST VIEW EMPLOYEES' NAMED 'Emp1'
*
FORMAT KD=ON
INPUT ////
10X 'The following function keys are assigned:' //
10X 'PF1: Funktion for PF1      ' /
10X 'PF2: Funktion for PF2      ' /
10X 'PF3: Return to MENU program' /
10X 'PF4: LIST VIEW             ' /
10X 'PF5: LIST VIEW EMPLOYEES   ' ///
*
IF *PF-KEY = 'PF1'

```

SET KEY

```
WRITE 'Funktion for PF1 executed.'  
END-IF  
IF *PF-KEY = 'PF2'  
  WRITE 'Funktion for PF2 executed.'  
END-IF  
*  
END
```

Ausgabe des Programms SKYEX1:

The following function keys are assigned:

```
PF1: Funktion for PF1  
PF2: Funktion for PF2  
PF3: Return to MENU program  
PF4: LIST VIEW  
PF5: LIST VIEW EMPLOYEES
```

121 SET TIME

▪ Funktion	820
▪ Beispiel	820

```
{ SET TIME }  
{ SETTIME }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Das Statement SET TIME (oder SETTIME) wird in Verbindung mit der Natural-Systemvariablen *TIMD verwendet und dient dazu, die für die Ausführung eines bestimmten Programnteils benötigte Zeit zu messen.

Das SET TIME-Statement wird an einer bestimmten Stelle im Programm platziert, und die Systemvariable *TIMD gibt dann an, wieviel Zeit seit der Ausführung des SET TIME-Statements verstrichen ist.

Die Systemvariable *TIMD muss das SET TIME-Statement ausdrücklich referenzieren, entweder durch Angabe der Sourcecode-Zeilenummer oder mittels eines Statement-Labels.

Beispiel

```
** Example 'STIEX1': SETTIME  
*****  
DEFINE DATA LOCAL  
1 EMPLOY-VIEW VIEW OF EMPLOYEES  
  2 NAME  
END-DEFINE  
*  
ST. SETTIME  
WRITE 10X 'START TIME:' *TIME  
*  
READ (100) EMPLOY-VIEW BY NAME  
END-READ  
*  
WRITE NOTITLE 10X 'END TIME: ' *TIME  
WRITE          10X 'ELAPSED TIME TO READ 100 RECORDS'  
                '(HH:II:SS.T) :' *TIMD (ST.) (EM=99:99:99'.'9)  
*  
END
```

Ausgabe des Programms STIEX1:

START TIME: 16:39:07.6
END TIME: 16:39:07.7
ELAPSED TIME TO READ 100 RECORDS (HH:MM:SS.T) : 00:00:00.1

122 SET WINDOW

▪ Funktion	824
▪ Syntax-Beschreibung	824
▪ Beispiel	825

```
SET WINDOW { 'window-name'
             OFF }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: `DEFINE WINDOW` | `INPUT WINDOW='window-name'` | `REINPUT`

Gehört zur Funktionsgruppe: *Bildschirmgenerierung für interaktive Verarbeitung*

Funktion

Das Statement `SET WINDOW` dient dazu, ein Bildschirmfenster (“Window”) zu aktivieren und zu deaktivieren.

Jedes `SET WINDOW 'window-name'`- oder `INPUT WINDOW='window-name'`-Statement deaktiviert das gerade aktive Fenster und aktiviert das im Statement angegebene Fenster. Dies bedeutet, dass jeweils nur ein Fenster zur Zeit aktiv sein kann.



Anmerkung: Wenn Sie mit `SET WINDOW` ein Fenster aktivieren, das mit `SIZE AUTO` definiert ist, bestimmen die Daten, die auf dem Schirm sind, *bevor* das Fenster aktiviert wird, die Größe des Fensters.

Syntax-Beschreibung

SET WINDOW 'window-name'	Mit <code>SET WINDOW 'window-name'</code> aktivieren Sie das angegebene Fenster; d.h. alle nachfolgenden Statements beziehen sich auf dieses Fenster, bis es entweder deaktiviert oder ein anderes Fenster aktiviert wird. Das angegebene Fenster muss in einem <code>DEFINE WINDOW</code> -Statement definiert worden sein.
SET WINDOW OFF	Mit <code>SET WINDOW OFF</code> deaktivieren Sie das gerade aktive Fenster.

Beispiel

Siehe `DEFINE WINDOW`-Statement.

123 SKIP

▪ Funktion	828
▪ Syntax-Beschreibung	828
▪ Beispiel	829

```
SKIP [(rep)] operand1 [LINES]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt **Syntax-Symbole**.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TITLE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellung von Ausgabe-Reports*

Funktion

Das Statement SKIP dient dazu, in einem Ausgabe-Report eine oder mehrere Leerzeilen zu generieren.

Siehe auch *Seitenüberschriften, Seitenvorschübe und Leerzeilen im Leitfaden zur Programmierung*.

Verarbeitung

Wenn bei der Ausführung eines SKIP-Statements die in den Report einzufügenden Leerzeilen nicht mehr auf die aktuelle Ausgabeseite passen, werden die überschüssigen Leerzeilen ignoriert (außer in einem AT TOP OF PAGE-Statement).

Ein SKIP-Statement wird nur ausgeführt, falls vorher auf der Seite bereits etwas ausgegeben wurde (eine über ein AT TOP OF PAGE-Statement erzeugte Ausgabe wird hierbei nicht berücksichtigt).

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand1	C S	N P I	ja	nein

Syntax-Element-Beschreibung:

<i>(rep)</i>	<p>Report-Spezifikation::</p> <p>Mit der Notation (<i>rep</i>) kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement SKIP beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls nichts anderes angegeben wird, bezieht sich das SKIP-Statement auf den ersten ausgegebenen Report (Report 0).</p> <p>Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern können, finden Sie im Abschnitt <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
<i>operand1</i>	<p>Anzahl der Leerzeilen:</p> <p><i>operand1</i> ist die Anzahl der zu generierenden Leerzeilen (1 – 250). Die Anzahl kann als numerische Konstante oder als Inhalt einer numerischen Variablen angegeben werden.</p> <p>Ist <i>operand1</i> größer als die für den Report definierte Seitenlänge, so löst das SKIP-Statement eine <i>Newpage</i>-Bedingung aus.</p>

Beispiel

```

** Example 'SKPEX1': SKIP
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 COUNTRY
  2 NAME
END-DEFINE
*
LIMIT 7
READ EMPL-VIEW BY CITY STARTING FROM 'W'
  AT BREAK OF CITY
    SKIP 2
  END-BREAK
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
/*
END-READ
END

```

Ausgabe des Programms SKPEX1:

CITY	COUNTRY	NAME
WASHINGTON	USA	REINSTEDT PERRY
WEITERSTADT	D	BUNGERT UNGER DECKER
WEST BRIDGFORD	UK	ENTWHISTLE
WEST MIFFLIN	USA	WATSON

124 SORT

▪ Funktion	832
▪ Einschränkungen	833
▪ Syntax-Beschreibung	833
▪ Phasen der SORT-Verarbeitung	836
▪ Beispiel	837

Structured Mode-Syntax

```

END-ALL
[AND]
SORT      [ THEM          ] [BY] { operand1 [ ASCENDING  ] } ... 10
          [ RECORDS      ]
          USING-clause
          [GIVE-clause]
          statement ...
END-SORT

```

* Wenn ein Statement-Label angegeben wird, muss es vor dem Schlüsselwort SORT, aber *nach* END-ALL (und AND) stehen.

Reporting Mode-Syntax

```

SORT [ THEM          ] [BY] { operand1 [ ASCENDING  ] } ... 10
     [ RECORDS      ]
     [USING-clause]
     [GIVE-clause]
     statement ...
[LOOP]

```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandtes Statement: [FIND mit SORTED BY-Option](#)

Gehört zur Funktionsgruppe: [Schleifenverarbeitung](#)

Funktion

Das Statement SORT dient dazu, eine Sortieroperation durchzuführen und die Datensätze aus allen Verarbeitungsschleifen, die zum Zeitpunkt der SORT-Ausführung aktiv sind, zu sortieren.

Einschränkungen

- Das SORT-Statement muss im selben Objekt stehen wie die Verarbeitungsschleifen, deren Datensätze es sortiert.
- Geschachtelte SORT-Statements sind nicht erlaubt.
- Die Gesamtlänge eines zu sortierenden Datensatzes darf 10240 Bytes nicht überschreiten.
- Die Anzahl der Sortierkriterien darf 10 nicht überschreiten.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S	A N P I F B D T	nein	nein

Syntax-Element-Beschreibung:

END-ALL	<p>Im Structured Mode müssen Sie vor dem SORT-Statement das Statement END-ALL angeben; damit werden alle noch aktiven Verarbeitungsschleifen beendet. Das SORT-Statement initiiert seinerseits eine neue Verarbeitungsschleife, welche mit END-SORT geschlossen werden muss.</p> <p>Anmerkung: Im Reporting Mode beendet das SORT-Statement alle noch aktiven Verarbeitungsschleifen und initiiert eine neue Verarbeitungsschleife.</p>
<i>operand1</i>	<p>Sortierkriterien:</p> <p><i>operand1</i> sind die Felder/Variablen, die als Sortierkriterium dienen. 1 bis 10 Felder dürfen angegeben werden. Hierbei kann es sich um Datenbankfelder (Deskriptoren oder Nicht-Deskriptoren) und/oder Benutzervariablen handeln. Multiple Felder oder Felder aus einer Periodengruppe können ebenfalls verwendet werden; eine Feldgruppe oder ein Array kann nicht verwendet werden.</p>
ASCENDING	<p>Sortierreihenfolge:</p> <p>Wird nichts anderes angegeben, so gilt ASCENDING, d.h. die Werte werden in aufsteigender Reihenfolge sortiert. Möchten Sie die Werte in absteigender Reihenfolge sortiert haben, geben Sie das Schlüsselwort DESCENDING an.</p> <p>Das Schlüsselwort ASCENDING bzw. DESCENDING kann für jedes Sortierfeld getrennt angegeben werden.</p>
DESCENDING	
USING	USING-Klausel:

	Siehe <i>USING-Klausel</i> weiter unten.
GIVE	GIVE-Klausel: Siehe <i>GIVE-Klausel</i> weiter unten.
END-SORT	Das für Natural reservierte Wort END-SORT muss zum Beenden des SORT-Statements benutzt werden.

USING-Klausel

In der USING-Klausel geben Sie die Felder an, die in den Sortier-Zwischenspeicher geschrieben werden sollen. Die USING-Klausel ist im Structured Mode unbedingt erforderlich, im Reporting Mode nicht; allerdings wird dringend empfohlen, sie auch im Reporting Mode zu verwenden, um Speicherplatz zu sparen.

```
{ USING {operand2}... }
  USING KEYS }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand2</i>	S A	A N P I F B D T L C	nein	nein

Syntax-Element-Beschreibung:

USING <i>operand2</i>	Mit <i>USING operand2</i> können Sie weitere Felder angeben, die — zusätzlich zu den (als <i>operand1</i> angegebenen) Sortierfeldern — in den Sortier-Zwischenspeicher geschrieben werden sollen.
USING KEYS	Wenn Sie <i>USING KEYS</i> angeben, werden nur die als <i>operand1</i> angegebenen Sortierfelder in den Sortier-Zwischenspeicher geschrieben.

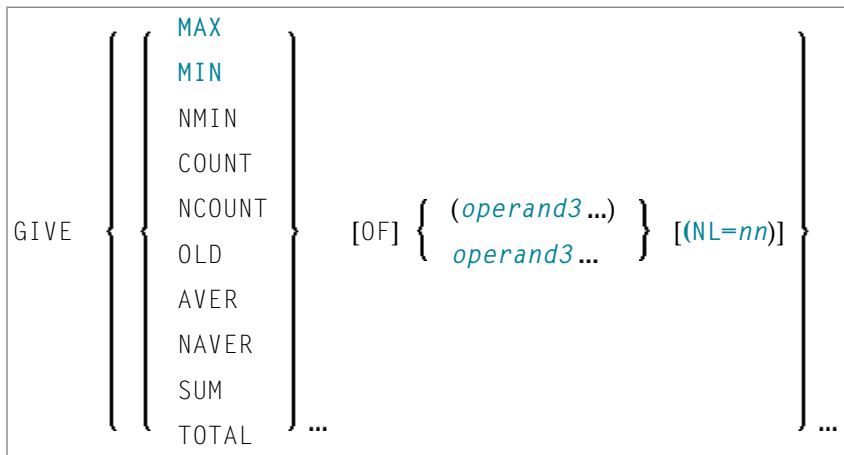
Im Reporting Mode:

Verwenden Sie keine USING-Klausel, so werden alle Datenbankfelder aus vor dem SORT-Statement initiierten Verarbeitungsschleifen sowie alle vor dem SORT-Statement definierten Benutzervariablen in den Sortier-Zwischenspeicher geschrieben.

Wird nach Ausführung des SORT-Statements ein Feld referenziert, das nicht in den Sortier-Zwischenspeicher geschrieben wurde, so ist der Wert des Feldes der, den es vor der SORT-Operation hatte.

GIVE-Klausel

Die GIVE-Klausel dient dazu, Natural-Systemfunktionen (MAX, MIN usw.) anzugeben, die in der ersten Phase des Sortiervorgangs ausgewertet werden und dann in der dritten Phase referenziert werden können (siehe Abschnitt *Phasen der SORT-Verarbeitung*). Wird nach dem SORT-Statement eine Systemfunktion referenziert, muss ihrem Namen ein Stern vorangestellt werden; Beispiel: *AVER(SALARY).



Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand3	S A	*	ja	nein

* je nach Funktion

Syntax-Element-Beschreibung:

MAX MIN NMIN COUNT NCOUNT OLD AVER NAVER SUM TOTAL	Näheres zu den einzelnen Systemfunktionen finden Sie in der <i>Systemfunktionen</i> -Dokumentation.
operand3	operand3 ist der Feldname.
(NL=nn)	Diese Option gilt nur für AVER, NAVER, SUM und TOTAL; für alle anderen Systemfunktionen wird sie ignoriert. Diese Option kann dazu verwendet werden, einen arithmetischen Überlauf bei der Auswertung von Systemfunktionen zu vermeiden; sie ist unter <i>Arithmetischer Überlauf bei AVER, NAVER, SUM oder TOTAL</i> in der <i>Systemfunktionen</i> -Dokumentation beschrieben.

Phasen der SORT-Verarbeitung

Ein Programm, das ein `SORT`-Statement enthält, wird in drei Phasen ausgeführt:

1. Phase – Auswählen der zu sortierenden Datensätze

Die Statements vor dem `SORT`-Statement werden ausgeführt. Die in der `USING`-Klausel angegebenen Daten werden in den Sortier-Zwischenspeicher geschrieben.

Im Reporting Mode dürfen Variablen, die nach dem Sortieren als Akkumulatoren verwendet werden, nicht vor dem `SORT`-Statement definiert werden.

Im Structured Mode dürfen sie nicht in der `USING`-Klausel angegeben werden.

In den Sortier-Zwischenspeicher geschriebene Felder können als Akkumulatoren nicht verwendet werden, weil sie in der dritten Phase mit jedem einzelnen Datensatz zurückgeschrieben werden. Folglich hätte die Akkumulationsfunktion nicht das gewünschte Ergebnis, da das Feld bei jedem Datensatz mit dem Wert des jeweiligen Datensatzes überschrieben würde.

Die Anzahl der in den Sortier-Zwischenspeicher geschriebenen Datensätze ergibt sich aus der Anzahl der Verarbeitungsschleifen und der Anzahl der verarbeiteten Datensätze pro Schleife. Jedesmal wenn das `SORT`-Statement in einer Verarbeitungsschleife ausgeführt wird, wird im internen Sortier-Zwischenspeicher ein Datensatz angelegt.

Bei geschachtelten Schleifen wird ein Datensatz nur in den Sortier-Zwischenspeicher geschrieben, wenn die innere Schleife ausgeführt wird. Sollen im folgenden Beispiel Datensätze in den Sortier-Zwischenspeicher geschrieben werden, auch wenn in der inneren (`FIND`-)Schleife keine gefunden werden, so muss das `FIND`-Statement eine `IF NO RECORDS FOUND`-Klausel enthalten.

```
READ ...
...
  FIND ...
...
END-ALL
SORT ...
  DISPLAY ...
END-SORT
...
```

2. Phase – Sortieren der Datensätze

Die Datensätze werden sortiert.

3. Phase — Weiterverarbeitung der sortierten Datensätze

Die Datensätze aus dem Sortier-Zwischenspeicher werden in der angegebenen Sortierfolge mit den auf das SORT-Statement folgenden Statements weiterverarbeitet. Werden Datenbankfelder nach dem SORT-Statement referenziert, so muss dies über ein Statement-Label oder durch Angabe der entsprechenden Sourcecode- Zeilennummer erfolgen.

Beispiel

- Beispiel 1 — SORT
- Beispiel 2 — SORT

Beispiel 1 — SORT

```

** Example 'SRTEX1S': SORT (structured mode)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 SALARY      (1:2)
  2 PERSONNEL-ID
  2 CURR-CODE  (1:2)
*
1 #AVG          (P11)
1 #TOTAL-TOTAL (P11)
1 #TOTAL-SALARY (P11)
1 #AVER-PERCENT (N3.2)
END-DEFINE
*
LIMIT 3
FIND EMPL-VIEW WITH CITY = 'BOSTON'
  COMPUTE #TOTAL-SALARY = SALARY (1) + SALARY (2)
  ACCEPT IF #TOTAL-SALARY GT 0
/*
END-ALL
AND
SORT BY PERSONNEL-ID USING #TOTAL-SALARY SALARY(*) CURR-CODE(1)
  GIVE AVER(#TOTAL-SALARY)
/*
AT START OF DATA
  WRITE NOTITLE '*' (40)
    'AVG CUMULATIVE SALARY:' *AVER (#TOTAL-SALARY) /
  MOVE *AVER (#TOTAL-SALARY) TO #AVG
END-START
COMPUTE ROUNDED #AVER-PERCENT = #TOTAL-SALARY / #AVG * 100
ADD #TOTAL-SALARY TO #TOTAL-TOTAL

```

```

/*
DISPLAY NOTITLE PERSONNEL-ID SALARY (1) SALARY (2)
      #TOTAL-SALARY CURR-CODE (1)
      'PERCENT/OF/AVER' #AVER-PERCENT
AT END OF DATA
  WRITE / '*' (40) 'TOTAL SALARIES PAID: ' #TOTAL-TOTAL
END-ENDDATA
END-SORT
*
END

```

Ausgabe des Programms SRTEX1S:

PERSONNEL ID	ANNUAL SALARY	ANNUAL SALARY	#TOTAL-SALARY	CURRENCY CODE	PERCENT OF AVER
***** AVG CUMULATIVE SALARY:					41900
20007000	16000	15200	31200	USD	74.00
20019200	18000	17100	35100	USD	83.00
20020000	30500	28900	59400	USD	141.00
***** TOTAL SALARIES PAID:					125700

Das obige Beispiel wird wie folgt verarbeitet:

Phase 1:

- Von der EMPLOYEES-Datei werden Datensätze mit CITY = BOSTON gelesen.
- Die ersten beiden Ausprägungen des SALARY-Feldes werden in der Variablen #TOTAL-SALARY addiert.
- Es werden nur Datensätze weiterverarbeitet, bei denen der Wert von #TOTAL-SALARY größer als 0 ist.
- Die Sätze werden in den Sortier-Zwischenspeicher geschrieben. Die Datenbank-Arrays SALARY (die ersten beiden Ausprägungen) und CURR-CODE (erste Ausprägung), das Datenbankfeld PERSONNEL-ID und die Benutzervariable #TOTAL-SALARY werden in den Zwischenspeicher geschrieben.
- Der Durchschnittswert von #TOTAL-SALARY wird errechnet.

Phase 2:

- Die Datensätze werden sortiert.

Phase 3:

- Der sortierte Zwischenspeicherinhalt wird gelesen.
- Bei der Ausführung des AT START OF DATA-Blocks wird der Durchschnittswert von #TOTAL-SALARY angezeigt.
- Der Wert von #TOTAL-SALARY wird zu #TOTAL-TOTAL hinzuaddiert; es werden die Felder PERSONNEL-ID, SALARY (1), SALARY (2), #AVER-PERCENT und #TOTAL-SALARY angezeigt.
- Bei der Ausführung des AT END OF DATA-Blocks wird die Variable #TOTAL-TOTAL ausgegeben.

Äquivalentes Reporting-Mode-Beispiel: [SRTEX1R](#).

Beispiel 2 — SORT

```

** Example 'SRTEX2': SORT
*****
DEFINE DATA LOCAL
1  VEHIC-VIEW VIEW OF VEHICLES
   2  MAKE
   2  YEAR
END-DEFINE
*
LIMIT 10
*
READ VEHIC-VIEW
END-ALL
SORT BY MAKE YEAR USING KEY
  DISPLAY NOTITLE (AL=15) MAKE (IS=ON) YEAR
  AT BREAK OF MAKE
    WRITE '- ' (20)
  END-BREAK
END-SORT
END

```

Ausgabe des Programms SRTEX2S:

MAKE	YEAR
FIAT	1980
	1982
	1984
PEUGEOT	1980
	1982
	1985

SORT

RENAULT	1980
	1980
	1982
	1982

125 STACK

▪ Funktion	842
▪ Syntax-Beschreibung	842
▪ Beispiel	845

```
STACK [TOP] { COMMAND operand1 [operand2 [(parameter)]] ... }
              { [DATA] [FORMATTED] {operand2 [(parameter)]] ... }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: [INPUT](#) | [RELEASE](#)

Funktion

Das Statement `STACK` dient dazu, Daten im Natural-Stack abzulegen. Hierbei kann es sich um folgende Daten handeln:

- den Namen eines Natural-Programms oder -Systemkommandos, das ausgeführt werden soll;
- Daten, die bei der Ausführung eines `INPUT`-Statements als Eingabedaten verwendet werden sollen.

Weitere Informationen zum Stack finden Sie im Kapitel *Weitere Programmieraspekte, Stack im Leitfaden zur Programmierung*.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A G N	A	ja	ja
<i>operand2</i>	C S A G N	A U N P I F B D T L G	ja	ja

Syntax-Element-Beschreibung:

TOP	<p>Normalerweise werden die Daten unten im Stack abgelegt. Das Schlüsselwort TOP bewirkt, dass die Daten oben auf dem Natural Stack abgelegt werden.</p> <p>Beispiel: Mit diesem Statement wird der Inhalt der Variablen #FIELDA oben auf dem Stack abgelegt:</p> <pre>STACK TOP #FIELDA</pre>
DATA	<p>Mit DATA (Standardeinstellung) legen Sie Daten im Stack ab, die von einem INPUT-Statement als Eingabedaten verwendet werden.</p> <p>Begrenzungszeichen oder <i>Input Assign</i>-Zeichen innerhalb der übergebenen Datenwerte werden als Begrenzung interpretiert und entsprechend verarbeitet. Einzelheiten darüber, wie die Daten von einem INPUT-Statement verarbeitet werden, können Sie der Beschreibung des INPUT-Statements <i>Eingabedaten aus dem Natural-Stack</i> entnehmen.</p> <p>Beispiel: Mit den folgenden Statements wird der Inhalt der Variablen #FIELD1 und #FIELD2 im Stack abgelegt:</p> <pre>MOVE 'ABC' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK #FIELD1 #FIELD2</pre> <p>Diese Variablen werden als Eingabedaten an das nächste INPUT-Statement im Natural-Programm übergeben, und zwar im Begrenzungs-Modus:</p> <pre>INPUT #FIELD1 #FIELD2</pre> <p>Anmerkung: Wenn <i>operand2</i> eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts im Stack abgelegt, aber nicht die Datumskomponente.</p>
FORMATTED	<p>Das Schlüsselwort FORMATTED bewirkt, dass alle Daten Feld für Feld an das nächste INPUT-Statement übergeben werden. Schlüsselzuordnungen oder Begrenzungszeichen werden nicht als solche interpretiert.</p> <p>Beispiele:</p>

	<p>Mit den folgenden Statements wird ABC ,DEF in #FIELD1 übertragen und XYZ in #FIELD2:</p> <pre>MOVE 'ABC,DEF' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK TOP DATA FORMATTED #FIELD1 #FIELD2 ... INPUT #FIELD1 #FIELD2</pre> <p>Angenommen, das Input-Begrenzungszeichen ist das Komma (Profil-/Session-Parameter ID= ,), dann wird mit folgenden Statements – ohne das Schlüsselwort FORMATTED – ABC in #FIELD1 übertragen und DEF in #FIELD2:</p> <pre>MOVE 'ABC,DEF' TO #FIELD1 STACK TOP DATA #FIELD1 ... INPUT #FIELD1 #FIELD2</pre> <p>Anmerkung: Die FORMATTED-Option sollte verwendet werden, wenn die zu übergebenden Daten Begrenzungs-, Steuer- oder DBCS-Zeichen enthalten, um eine unbeabsichtigte Interpretation dieser Zeichen zu vermeiden.</p>
<p>COMMAND <i>operand1</i></p>	<p>Um ein Kommando (bzw. einen Programmnamen) im Stack abzulegen, geben Sie das Schlüsselwort COMMAND gefolgt von dem betreffenden Kommando (<i>operand1</i>) an. Würde ein Programm normalerweise den Benutzer mit einer Eingabeaufforderung in Form einer NEXT-Zeile konfrontieren, so unterdrückt nun Natural die Anzeige der NEXT-Zeile und führt stattdessen das im Stack abgelegte Kommando aus.</p> <p>Beispiel: Mit dem folgenden Statement wird das Kommando RUN oben auf dem Stack abgelegt. Natural führt dieses Kommando aus, wenn normalerweise das nächstmal die NEXT-Zeile ausgegeben würde:</p> <pre>STACK TOP COMMAND 'RUN'</pre>
<p>COMMAND <i>operand1</i> <i>operand2 ...</i></p>	<p>Zusammen mit einem Kommando (<i>operand1</i>) können Sie auch Daten (<i>operand2</i>) im Stack ablegen. Diese Daten werden dann vom nächsten INPUT-Statement nach der Ausführung des Kommandos als Eingabedaten verarbeitet.</p> <p>Zusammen mit einem Kommando abgelegte Daten werden immer unformatiert abgelegt.</p> <p>Anmerkung: Wenn in den abzulegenden Daten leere alphanumerische Felder (d.h. Leerzeichen) enthalten sind, werden diese Leerzeichen als Delimiter zwischen Werten interpretiert und folglich von dem betreffenden INPUT-Statement falsch verarbeitet. Wenn Sie daher leere alphanumerische Felder als Daten zusammen mit einem Kommando im Stack ablegen möchten, müssen Sie hierzu zwei STACK-Statements verwenden: Ein STACK DATA <i>operand2. . .</i>, um die Daten abzulegen, und ein STACK COMMAND <i>operand1</i>, um das Kommando abzulegen.</p>

<i>parameter</i>	Wenn <i>operand2</i> eine Datumsvariable ist, können Sie den Session-Parameter DF (Datumsformat) als <i>parameter</i> für diese Variable angeben.
------------------	---

Beispiel

```

** Example 'STKEX1': STACK
*****
DEFINE DATA LOCAL
1 #CODE (A1)
END-DEFINE
*
INPUT //
  10X 'PLEASE SELECT COMMAND' //
  10X 'LIST VIEW      (V)' /
  10X 'LIST PROGRAM * (P)' /
  10X 'TECH INFO     (T)' /
  10X 'STOP          (.)' //
  20X 'CODE:' #CODE
*
*
DECIDE ON FIRST #CODE
  VALUE 'V'
    STACK TOP DATA      'VIEW'
    STACK TOP COMMAND   'LIST'
  VALUE 'P'
    STACK TOP COMMAND   'LIST PROGRAM *'
  VALUE 'T'
    STACK TOP COMMAND   'LAST *'
    STACK TOP COMMAND   'TECH'
    STACK TOP COMMAND   'SYSPROD'
  VALUE '.'
    STOP
  NONE
    REINPUT 'PLEASE ENTER VALID CODE'
END-DECIDE
*
*
END

```

Ausgabe des Programms STKEX1:

PLEASE SELECT COMMAND

- LIST VIEW (V)
- LIST PROGRAM * (P)
- TECH INFO (T)
- STOP (.)

CODE:P

Nach Eingabe und Bestätigung des Codes:

16:46:28 ***** NATURAL LIST COMMAND ***** 2005-01-19
 User HTR - LIST Objects in a Library - Library SYSEXSUN

Cmd	Name	Type	S/C	SM	Version	User ID	Date	Time
---	*	P	*	*	*	*	*	*
___	ACREX1	Program	S/C	S	4.1.03	RKE	2004-11-11	16:32:37
___	ACREX2	Program	S/C	S	4.1.03	RKE	2005-01-05	10:29:51
___	ADDEX1	Program	S/C	S	4.1.03	RKE	2004-11-11	16:36:49
___	AEDEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	16:40:34
___	AEDEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	16:39:57
___	AEPEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	16:41:57
___	AEPEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	16:42:31
___	AEPEX2	Program	S/C	S	4.1.03	RKE	2004-11-11	16:43:37
___	ASDEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:00:21
___	ASDEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:00:50
___	ASGEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:02:01
___	ASGEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:02:08
___	ATBEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:03:18
___	ATBEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:03:05

14 Objects found

Top of List.

Command ==>

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
 Help Print Exit Sort -- - + ++ > Canc

126 STOP

▪ Funktion	848
▪ Beispiel	848

STOP

Dieses Kapitel behandelt folgende Themen:

Funktion

Mit dem Statement `STOP` können Sie die Ausführung eines Programmes abbrechen und erhalten dann eine Kommandozeile.

Sie können ein `STOP`-Statement an beliebiger Stelle im Programm verwenden und auch mehrere `STOP`-Statements benutzen. Mit dem `STOP`-Statement wird die Ausführung des Programms sofort abgebrochen. Befindet sich das `STOP`-Statement in einer Subroutine, so wird vor dem Abbruch noch eine etwaige im Hauptprogramm angegebene Seitenende-Bedingung (End-of-Page) zur abschließenden Seitenende-Verarbeitung ausgeführt.

Beim Ausführen einer Methode verhält sich das `STOP`-Statement wie das `ESCAPE ROUTINE`-Statement. Die Ausführung der Methode wird sofort beendet, und es wird kein Rückmeldewert erzeugt.

Beim Natural Remote Procedure Call (RPC): Siehe *Notes on Natural Statements on the Server* in der *Natural Remote Procedure Call (RPC)*-Dokumentation.

Beispiel

```

** Example 'STPEX1': STOP
*****
DEFINE DATA LOCAL
1 #CODE (A1)
END-DEFINE
*
INPUT //
  10X 'PLEASE SELECT COMMAND' //
  10X 'LIST VIEW      (V)' /
  10X 'LIST PROGRAM * (P)' /
  10X 'TECH INFO      (T)' /
  10X 'STOP           (.)' //
  20X 'CODE:' #CODE
*
*
DECIDE ON FIRST #CODE
  VALUE 'V'
    STACK TOP DATA      'VIEW'
    STACK TOP COMMAND 'LIST'
  VALUE 'P'

```

```
    STACK TOP COMMAND 'LIST PROGRAM *'  
    VALUE 'T'  
    STACK TOP COMMAND 'LAST *'  
    STACK TOP COMMAND 'TECH'  
    STACK TOP COMMAND 'SYSPROD'  
    VALUE '.'  
    STOP  
    NONE  
    REINPUT 'PLEASE ENTER VALID CODE'  
END-DECIDE  
*  
*  
END
```

Ausgabe des Programms STPEX1:

```
PLEASE SELECT COMMAND  
  
LIST VIEW      (V)  
LIST PROGRAM * (P)  
TECH INFO     (T)  
STOP          (.)  
  
CODE:
```


127 STORE

▪ Funktion	852
▪ Datenbankspezifische Anmerkungen	853
▪ Syntax-Beschreibung	853
▪ Beispiel	855

Structured Mode-Syntax

```
STORE [RECORD] [IN] [FILE] view-name
      [PASSWORD=operand1]
      [CIPHER=operand2]
      [          [          USING          ] NUMBER operand3 ]
      [          [          GIVING         ] ]
```

Reporting Mode-Syntax

```
STORE [RECORD] [IN] [FILE] view-name
      [PASSWORD=operand1]
      [CIPHER=operand2]
      [          [          USING          ]          NUMBER operand3 ]
      [          [          GIVING         ] ]
      {          [USING] SAME [RECORD] [AS] [STATEMENT [(r)] ]          }
      [          [          SET           ]          [operand4=operand5] ]
      [          [          WITH          ]          ... ]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement STORE dient dazu, auf einer Datenbank einen Datensatz hinzuzufügen.

Datenbankspezifische Anmerkungen

Adabas	Die Natural-Systemvariable *ISN enthält die Adabas ISN, die dem neuen Datensatz als Ergebnis der Ausführung des STORE-Statements zugewiesen wurde. Eine anschließende Referenz auf *ISN muss die Statement-Nummer des betreffenden STORE-Statements enthalten.
SQL	Mit dem STORE-Statement können Sie einer Tabelle eine Reihe hinzufügen. Die PASSWORD-, CIPHER- und GIVING NUMBER-Klauseln sind nicht erlaubt. Das STORE-Statement entspricht dem SQL-Statement INSERT. Die Natural-Systemvariable *ISN steht nicht zur Verfügung.
XML	Mit dem STORE-Statement können Sie einer Datenbank ein XML-Objekt hinzufügen. Die PASSWORD-, CIPHER- und GIVING NUMBER-Klauseln sind nicht erlaubt. Für Tamino enthält die Natural-Systemvariable *ISN die XML-Objekt-ID, die dem neuen Datensatz als Ergebnis der Ausführung des STORE-Statements zugewiesen wurde. Eine anschließende Referenz auf *ISN muss die Statement-Nummer des betreffenden STORE-Statements enthalten.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur			Mögliche Formate										Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>	C	S		A											ja	nein
<i>operand2</i>	C	S			N										ja	nein
<i>operand3</i>		S			N	P		B *							nein	ja
<i>operand4</i>		S	A		A	U	N	P	I	F	B	D	T	L	nein	nein
<i>operand5</i>	C	S	A		A	U	N	P	I	F	B	D	T	L	ja	nein

* Format B von *operand3* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

<i>view-name</i>	<p>Als <i>view-name</i> geben Sie den Namen eines Views an, der entweder in einem DEFINE DATA-Block oder in einer programmexternen Global oder Local Data Area definiert ist.</p> <p>Im Reporting Mode ist <i>view-name</i> der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.</p>
PASSWORD= <i>operand1</i>	<p>Die PASSWORD-Klausel gilt nur bei Adabas-Datenbanken.</p> <p>Sie dient dazu, ein Passwort (<i>operand1</i>) anzugeben, um Daten auf einer passwortgeschützten Datei speichern zu können. Das Passwort (<i>operand1</i>) kann als eine alphanumerische Konstante oder als eine alphanumerische Variable angegeben werden. Es kann aus bis zu 8 Zeichen bestehen und darf keine Sonderzeichen oder eingebettete Leerzeichen enthalten. Wenn das Passwort als eine Konstante angegeben wird, muss es in Apostrophen stehen.</p> <p>Weitere Informationen siehe die Statements FIND und PASSW.</p>
CIPHER= <i>operand2</i>	<p>Die CIPHER-Klausel gilt nur bei Adabas-Datenbanken.</p> <p>Diese Klausel wird benutzt, um einen Chiffrierschlüssel (<i>operand2</i>) bei der Aktualisierung von Daten einer Datei anzugeben, die verschlüsselt ist. Der Chiffrierschlüssel (<i>operand2</i>) kann als eine numerische Konstante mit 8 Stellen oder als eine Benutzervariable mit Format/Länge N8 angegeben werden.</p> <p>Weitere Informationen siehe das Statement FIND.</p>
USING NUMBER <i>operand3</i> or GIVING NUMBER <i>operand3</i>	<p>Diese Klausel kann nur bei Adabas-Datenbanken benutzt werden.</p> <p>Mit dieser Klausel können Sie für einen zu speichernden Datensatz eine eigene Adabas-ISBN angeben. Ist die angegebene ISBN bereits vergeben, wird das Programm abgebrochen und eine entsprechende Fehlermeldung ausgegeben, es sei denn, es ist eine ON ERROR-Verarbeitung definiert.</p>
SET/WITH <i>operand4=operand5</i>	<p>Mit dieser Klausel können im Reporting Mode die Felder angegeben werden, für die Werte gespeichert werden sollen. Jedes in der Datei definierte Feld, das in der SET-Klausel nicht angegeben wird, erhält in dem neuen Datensatz einen Nullwert.</p> <p>Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das STORE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.</p>
USING SAME (r)	<p>Diese Klausel bewirkt im Reporting Mode, dass dieselben Feldwerte, die mit dem FIND-, GET- oder READ-Statement gelesen wurden, welches von dem STORE-Statement referenziert wird, als Werte des neuen Datensatzes gespeichert werden. Das Statement kann mit der Notation (<i>r</i>) mittels Sourcecode-Zeilenummer oder Statement-Label referenziert werden.</p>

	Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA -Statement verwendet wird, da sich in diesem Fall das STORE -Statement immer auf den gesamten im DEFINE DATA -Statement definierten View bezieht.
--	--

Beispiel

```

** Example 'STOEX1S': STORE (structured mode)
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 MAR-STAT
  2 BIRTH
  2 CITY
  2 COUNTRY
*
1 #PERSONNEL-ID (A8)
1 #NAME (A20)
1 #FIRST-NAME (A15)
1 #BIRTH-D (D)
1 #MAR-STAT (A1)
1 #BIRTH (A8)
1 #CITY (A20)
1 #COUNTRY (A3)
1 #CONF (A1)
END-DEFINE
*
REPEAT
  INPUT 'ENTER A PERSONNEL ID AND NAME (OR ''END'' TO END)' //
    'PERSONNEL-ID : ' #PERSONNEL-ID //
    'NAME : ' #NAME /
    'FIRST-NAME : ' #FIRST-NAME
  /*
  /* VALIDATE ENTERED DATA
  /*
  IF #PERSONNEL-ID = 'END' OR #NAME = 'END'
    STOP
  END-IF
  IF #NAME = ' '
    REINPUT WITH TEXT 'ENTER A LAST-NAME' MARK 2 AND SOUND ALARM
  END-IF
  IF #FIRST-NAME = ' '
    REINPUT WITH TEXT 'ENTER A FIRST-NAME' MARK 3 AND SOUND ALARM

```

```
END-IF
/*
/* ENSURE PERSON IS NOT ALREADY ON FILE
/*
FIND NUMBER EMPL-VIEW WITH PERSONNEL-ID = #PERSONNEL-ID
IF *NUMBER > 0
    REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
        MARK 1 AND SOUND ALARM
END-IF
MOVE 'N' TO #CONF
/*
/* GET FURTHER INFORMATION
/*
INPUT
'ADDITIONAL PERSONNEL DATA'                ////
'PERSONNEL-ID          :' #PERSONNEL-ID (AD=IO) /
'NAME                  :' #NAME          (AD=IO) /
'FIRST-NAME            :' #FIRST-NAME   (AD=IO) ///
'MARITAL STATUS        :' #MAR-STAT     /
'DATE OF BIRTH (YYYYMMDD) :' #BIRTH     /
'CITY                  :' #CITY         /
'COUNTRY (3 CHARACTERS) :' #COUNTRY    //
'ADD THIS RECORD (Y/N)  :' #CONF       (AD=M)
/*
/* ENSURE REQUIRED FIELDS CONTAIN VALID DATA
/*
IF NOT (#MAR-STAT = 'S' OR = 'M' OR = 'D' OR = 'W')
    REINPUT TEXT 'ENTER VALID MARITAL STATUS S=SINGLE ' -
        'M=MARRIED D=DIVORCED W=WIDOWED' MARK 1
END-IF
IF NOT (#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
    REINPUT TEXT 'ENTER CORRECT DATE' MARK 2
END-IF
IF #CITY = ' '
    REINPUT TEXT 'ENTER A CITY NAME' MARK 3
END-IF
IF #COUNTRY = ' '
    REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 4
END-IF
IF NOT (#CONF = 'N' OR= 'Y')
    REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 5
END-IF
IF #CONF = 'N'
    ESCAPE TOP
END-IF
/*
/* ADD THE RECORD
/*
MOVE EDITED #BIRTH TO #BIRTH-D (EM=YYYYMMDD)
/*
EMPL-VIEW.PERSONNEL-ID := #PERSONNEL-ID
EMPL-VIEW.NAME         := #NAME
```

```

EMPL-VIEW.FIRST-NAME := #FIRST-NAME
EMPL-VIEW.MAR-STAT   := #MAR-STAT
EMPL-VIEW.BIRTH      := #BIRTH-D
EMPL-VIEW.CITY       := #CITY
EMPL-VIEW.COUNTRY    := #COUNTRY
/*
STORE RECORD IN EMPL-VIEW
/*
END OF TRANSACTION
/*
WRITE NOTITLE 'RECORD HAS BEEN ADDED'
/*
END-REPEAT
END

```

Ausgabe des Programms STOEX1S:

```

ENTER A PERSONNEL ID AND NAME (OR 'END' TO END)

PERSONNEL-ID : 90001100

NAME          : JONES
FIRST-NAME    : EDWARD

```

Nach der Eingabe und Bestätigung der Personal-Schlüsseldaten werden zusätzliche Personal-Daten zur Eingabe angezeigt:

```

ADDITIONAL PERSONNEL DATA

PERSONNEL-ID          : 90001100
NAME                  : JONES
FIRST-NAME            : EDWARD

MARITAL STATUS       :
DATE OF BIRTH (YYYYMMDD) :
CITY                  :
COUNTRY (3 CHARACTERS) :

ADD THIS RECORD (Y/N) : N

```

Äquivalentes Reporting-Mode-Beispiel: [STOEX1R](#).

128 SUBTRACT

▪ Funktion	860
▪ Syntax-Beschreibung	860
▪ Beispiel	862

Dieses Kapitel behandelt folgende Themen:

Verwandte Statements: [ADD](#) | [COMPRESS](#) | [COMPUTE](#) | [DIVIDE](#) | [EXAMINE](#) | [MOVE](#) | [MOVE ALL](#) | [MULTIPLY](#) | [RESET](#) | [SEPARATE](#)

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion

Mit dem Statement `SUBTRACT` können Sie die Werte zweier oder mehrerer Operanden voneinander abziehen.

Verwenden Sie ein Datenbankfeld als Ergebnisfeld, so ändert sich der Wert des Feldes nur programmintern. Der Wert, den das Feld in der Datenbank hat, wird davon nicht beeinflusst.

Syntax-Beschreibung

Syntax 1 - SUBTRACT ohne GIVING-Klausel

```
SUBTRACT [ROUNDED] operand1 ... FROM operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A N	N P I F D T	ja	nein
<i>operand2</i>	S A M	N P I F D T	ja	nein

Syntax-Element-Beschreibung:

<i>operand1</i> FROM <i>operand2</i>	<p>Operanden:</p> <p><i>operand1</i> ist der Minuend, <i>operand2</i> ist der Subtrahend, folglich ist das Statement äquivalent zu:</p> $\langle operand2 \rangle := \langle operand2 \rangle - \langle operand1 \rangle$ <p>Zum Format der Operanden siehe auch <i>Regeln für arithmetische Operationen, Formatwahl im Hinblick auf die Verarbeitungszeit im Leitfaden zur Programmierung.</i></p>
ROUNDED	Runden:

	<p>Wenn Sie das Ergebnis gerundet wünschen, geben Sie das Schlüsselwort <code>ROUNDED</code> an.</p> <p>Die für das Runden gültigen Regeln finden Sie im Abschnitt <i>Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten im Leitfaden zur Programmierung</i>.</p>
--	--

Syntax 2 - SUBTRACT mit GIVING-Klausel

```
SUBTRACT [ROUNDED] operand1 ... FROM operand2 GIVING operand3
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A N	N P I F D T	ja	nein
<i>operand2</i>	C S A N	N P I F D T	ja	nein
<i>operand3</i>	S A M A U N P I F B* D T		ja	ja

* Format B von *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

GIVING	<p>Ergebnisfeld:</p> <p>Wenn Sie die <code>GIVING</code>-Klausel verwenden, erhalten Sie das Ergebnis der Subtraktion in <i>operand3</i>; der Wert von <i>operand2</i> ändert sich nicht.</p>
<i>operand1</i> FROM <i>operand2</i> <code>GIVING</code> <i>operand3</i>	<p>Operanden:</p> <p><i>operand2</i> ist der Minuend, <i>operand1</i> ist der Subtrahend, <i>operand3</i> ist das Ergebnisfeld, folglich ist das Statement äquivalent zu:</p> $\langle \textit{oper3} \rangle := \langle \textit{oper2} \rangle - \langle \textit{oper1} \rangle$ <p>Zum Format der Operanden siehe auch <i>Regeln für arithmetische Operationen, Formatwahl im Hinblick auf die Verarbeitungszeit im Leitfaden zur Programmierung</i>.</p> <p>Zum Format der Operanden siehe auch <i>Formatwahl im Hinblick auf die Verarbeitungszeit (im Leitfaden zur Programmierung)</i>.</p>
ROUNDED	<p>Runden:</p> <p>Wünschen Sie das Ergebnis gerundet, geben Sie das Schlüsselwort <code>ROUNDED</code> an. Die für das Runden gültigen Regeln finden Sie im Abschnitt <i>Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten im Leitfaden zur Programmierung</i>.</p>

Beispiel

```

** Example 'SUBEX1': SUBTRACT
*****
DEFINE DATA LOCAL
1 #A (P2) INIT <50>
1 #B (P2)
1 #C (P1.1) INIT <2.4>
END-DEFINE
*
SUBTRACT 6 FROM #A
WRITE NOTITLE 'SUBTRACT 6 FROM #A          ' 10X '=' #A
*
SUBTRACT 6 FROM 11 GIVING #A
WRITE          'SUBTRACT 6 FROM 11 GIVING #A  ' 10X '=' #A
*
SUBTRACT 3 4 FROM #A GIVING #B
WRITE          'SUBTRACT 3 4 FROM #A GIVING #B ' 10X '=' #A '=' #B
*
SUBTRACT -3 -4 FROM #A GIVING #B
WRITE          'SUBTRACT -3 -4 FROM #A GIVING #B' 10X '=' #A '=' #B
*
SUBTRACT ROUNDED 2.06 FROM #C
WRITE          'SUBTRACT ROUNDED 2.06 FROM #C  ' 10X '=' #C
*
END

```

Ausgabe des Programms SUBEX1:

```

SUBTRACT 6 FROM #A          #A: 44
SUBTRACT 6 FROM 11 GIVING #A #A: 5
SUBTRACT 3 4 FROM #A GIVING #B #A: 5 #B: -2
SUBTRACT -3 -4 FROM #A GIVING #B #A: 5 #B: 12
SUBTRACT ROUNDED 2.06 FROM #C #C: 0.3

```

129

SUSPEND IDENTICAL SUPPRESS

▪ Funktion	864
▪ Syntax-Beschreibung	864
▪ Beispiele	865

SUSPEND IDENTICAL [SUPPRESS] [(rep)]

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | WRITE | WRITE TITLE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Mit dem Statement SUSPEND IDENTICAL SUPPRESS können Sie den Session-Parameter IS=ON (Unterdrückung identischer Feldwerte bei der Ausgabe) für einzelne Datensätze außer Kraft setzen.

Vgl. Session-Parameter IS in der *Parameter-Referenz*.

Syntax-Beschreibung

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Mit der Notation <i>(rep)</i> kann ein bestimmter anderer Report angegeben werden, auf den sich das SUSPEND IDENTICAL SUPPRESS-Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Falls nichts anderes angegeben wird, bezieht sich das Statement SUSPEND IDENTICAL SUPPRESS auf den ersten Report (Report 0). Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
--------------	---

Beispiele

- Beispiel 1 — Programm mit SUSPEND IDENTICAL SUPPRESS
- Beispiel 2 — Programm ohne SUSPEND IDENTICAL SUPPRESS

Beispiel 1 — Programm mit SUSPEND IDENTICAL SUPPRESS

```

** Example 'SISEX1': SUSPEND IDENTICAL SUPPRESS
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
*
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/*
  SUSPEND IDENTICAL SUPPRESS
/*
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
  IF NO RECORDS FOUND
    MOVE '***NO CAR***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    NAME (RD.) (IS=ON)
    FIRST-NAME (RD.) (IS=ON)
    MAKE (FD.)
  END-FIND
/*
END-READ
END

```

Ausgabe des Programms SISEX1:

SUSPEND IDENTICAL SUPPRESS

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
JONES	MARSHA	CHRYSLER
		CHRYSLER
JONES	ROBERT	GENERAL MOTORS
JONES	LILLY	FORD
		MG
JONES	EDWARD	GENERAL MOTORS
JONES	MARTHA	GENERAL MOTORS
JONES	LAUREL	GENERAL MOTORS
JONES	KEVIN	DATSUN
JONES	GREGORY	FORD
JONES	EDWARD	***NO CAR***
JOPER	MANFRED	***NO CAR***
JOUSSELIN	DANIEL	RENAULT
JUBE	GABRIEL	***NO CAR***
JUNG	ERNST	***NO CAR***
JUNKIN	JEREMY	***NO CAR***

Beispiel 2 — Programm ohne SUSPEND IDENTICAL SUPPRESS

```
** Example 'SISEX2': SUSPEND IDENTICAL SUPPRESS (compare with SISEX1)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/*
/* SUSPEND IDENTICAL SUPPRESS      /* statement removed
/*
FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
  IF NO RECORDS FOUND
    MOVE '***NO CAR***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    NAME (RD.) (IS=ON)
    FIRST-NAME (RD.) (IS=ON)
    MAKE (FD.)
```

```
END-FIND
/*
END-READ
END
```

Ausgabe des Programms SISEX2:

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
	MARSHA	CHRYSLER
		CHRYSLER
	ROBERT	GENERAL MOTORS
	LILLY	FORD
		MG
	EDWARD	GENERAL MOTORS
	MARTHA	GENERAL MOTORS
	LAUREL	GENERAL MOTORS
	KEVIN	DATSUN
JOPER	GREGORY	FORD
	EDWARD	***NO CAR***
	MANFRED	***NO CAR***
	DANIEL	RENAULT
	GABRIEL	***NO CAR***
	ERNST	***NO CAR***
	JEREMY	***NO CAR***

130

TERMINATE

▪ Funktion	870
▪ Syntax-Beschreibung	870
▪ Kontrollübergabe nach Abbruch	871
▪ Beispiel	871

```
TERMINATE [operand1 [operand2]]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Das Statement `TERMINATE` bewirkt, dass die Natural-Session abgebrochen wird. Sie können das `TERMINATE`-Statement an beliebiger Stelle im Programm verwenden. Bei der Ausführung eines `TERMINATE`-Statements wird keine End-of-Page-Verarbeitung oder schleifenbeendende Verarbeitung mehr ausgeführt.

Das Verhalten des `TERMINATE`-Statements entspricht dem des `STOP`-Statements. Die Verarbeitung von Rückgabewerten wird nicht unterstützt.

Beim Remote Procedure Call (RPC): Siehe *Notes on Natural Statements on the Server* in der *Natural Remote Procedure Call (RPC)*-Dokumentation.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	N P I	ja	nein
<i>operand2</i>	C S A	A U	ja	ja

Syntax-Element-Beschreibung:

<i>operand1</i>	<p>Return Code:</p> <p><i>operand1</i> kann dazu verwendet werden, einen Return Code an das Programm zu übergeben, das die Kontrolle erhält, nachdem die Natural-Session abgebrochen wurde.</p> <p>Für <i>operand1</i> kann ein Wert von 0 bis 255 angegeben werden.</p>
<i>operand2</i>	<p>Übergabe zusätzlicher Informationen:</p> <p><i>operand2</i> kann dazu verwendet werden, zusätzliche Informationen an das Programm zu übergeben, das nach dem Session-Abbruch die Kontrolle erhält.</p>

Kontrollübergabe nach Abbruch

Nach dem Abbruch der Natural-Session erhält das Programm, dessen Name mit dem Profilparameter PROGRAM angegeben wurde, die Kontrolle.

Natural übergibt *operand2* und den Wert des Profilparameters PRGPAR an dieses Programm, falls diese Angaben gemacht wurden. Das Programm erhält diese Parameter wie üblich als Argumente:

```
int main(int argc, char *argv[])
{
    /* Number of arguments passed. */
    printf("Number of arguments: %d\n", argc);
    /* Program name. */
    if ( argc > 0 )
        printf("Program: %s\n", argv[0]);
    /* Value of operand2 of the TERMINATE statement. */
    if ( argc > 1 )
        printf("Operand 2: %s\n", argv[1]);
    /* Value of the profile parameter PRGPAR. */
    if ( argc > 2 )
        printf("PRGPAR: %s\n", argv[2]);
    return 0;
}
```

Falls der Profilparameter PROGRAM nicht gesetzt wurde, erhält der Kommando-Interpreter die Kontrolle nach dem Abbruch.

Beispiel

```
** Example 'TEREX1': TERMINATE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 SALARY (1)
*
1 #PNUM      (A8)
1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD:' #PASSWORD
*
```

TERMINATE

```
IF #PASSWORD NE 'USERPASS'  
  /*  
  TERMINATE  
  /*  
END-IF  
*  
INPUT 'ENTER PERSONNEL NUMBER:' #PNUM  
*  
FIND EMPLOY-VIEW WITH PERSONNEL-ID = #PNUM  
  DISPLAY NAME SALARY (1)  
END-FIND  
*  
END
```

131 UPDATE

▪ Funktion	874
▪ Einschränkungen	875
▪ Datenbankspezifische Anmerkungen	875
▪ Syntax-Beschreibung	875
▪ Beispiel	877

Structured Mode-Syntax

```
UPDATE [RECORD] [IN] [STATEMENT] [(r)]
```

Reporting Mode-Syntax

```
UPDATE      [RECORD] [IN] [STATEMENT] [(r)]
           [
             SET
             WITH
             USING
           ] { SAME [RECORD]
             { operand1=operand2 } ... }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION DATA | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das UPDATE-Statement dient dazu, die in der Datenbank gespeicherten Werte eines oder mehrerer Felder eines Datensatzes zu verändern. Der betreffende Datensatz muss vorher mit einem FIND-, GET- oder READ-Statement (oder bei Adabas auch mit einem STORE-Statement) ausgewählt werden.

Hold-Status

Das UPDATE-Statement bewirkt, dass jeder mit dem betreffenden FIND-Statement oder READ-Statement gelesene Datensatz in den „Hold“-Status gestellt wird.

Die Hold-Logik ist im *Leitfaden zur Programmierung* beschrieben.

Einschränkungen

- Das UPDATE-Statement darf nicht in derselben Sourcecode-Zeile stehen wie das Statement, mit dem der zu aktualisierende Datensatz ausgewählt wird.
- Mit Entire System Server ist das UPDATE-Statement nicht verfügbar.

Datenbankspezifische Anmerkungen

SQL	<p>Mit dem UPDATE-Statement können Sie eine Reihe einer Datenbanktabelle aktualisieren. Das UPDATE-Statement entspricht dem SQL-Statement UPDATE WHERE CURRENT OF CURSOR (Positioned UPDATE), d.h. nur die zuletzt gelesene Reihe kann aktualisiert werden.</p> <p>Auf Großrechnern werden nur Spalten (Felder) aktualisiert, die innerhalb des Programms geändert wurden, sowie Spalten, die außerhalb des Programms (z.B. als Eingabefelder in Maps) geändert worden sein könnten (aber nicht notwendigerweise auch geändert wurden). Auf allen anderen Plattformen werden alle Spalten aktualisiert.</p> <p>Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht aktualisiert werden.</p>
XML	Bei XML-Datenbanken steht das UPDATE-Statement nicht zur Verfügung.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur				Mögliche Formate										Referenzierung erlaubt	Dynam. Definition	
<i>operand1</i>		S	A		A	N	P	I	F	B	D	T	L			nein	nein
<i>operand2</i>	C	S	A		A	N	P	I	F	B	D	T	L			ja	nein

Syntax-Element-Beschreibung:

<i>(r)</i>	<p>Statement-Referenzierung:</p> <p>Mit der Notation (<i>r</i>) können Sie das Statement referenzieren, mit dem der Datensatz, der aktualisiert werden soll, gelesen wurde. <i>r</i> kann als Statement-Label oder Sourcecode-Zeilenummer angegeben werden. Falls keine Referenzierung erfolgt, bezieht sich das UPDATE-Statement auf die innerste aktive READ- bzw. FIND-Verarbeitungsschleife.</p> <p>Ist keine READ- oder FIND-Schleife aktiv, bezieht es sich auf das letzte vorhergehende GET-Statement (bzw. STORE-Statement).</p> <p>Anmerkung: Das UPDATE-Statement muss innerhalb der READ- bzw. FIND-Schleife stehen, auf die es sich bezieht.</p>
USING SAME	<p>USING SAME-Klausel:</p> <p>Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das UPDATE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.</p> <p>Das Layout des Satzpuffers oder des Formatpuffers kann mit dem OBTAIN-Statement deklariert werden.</p> <p>Mit USING SAME geben Sie im Reporting Mode an, dass dieselben Felder aktualisiert werden sollen, die mit dem Statement, welches vom UPDATE-Statement referenziert wird, gelesen wurden; dies bedeutet, dass zur Aktualisierung der Felder die Werte verwendet werden, die den Datenbankfeldern zuletzt zugeordnet waren. Ist kein neuer Wert zugeordnet worden, wird der alte verwendet.</p> <p>Wenn das zu aktualisierende Feld ein Array-Bereich eines multiplen Feldes oder einer Periodengruppe ist und Sie einen variablen Index für diesen Array-Bereich verwenden, wird der zuletzt gültige Array-Bereich aktualisiert. Wenn die Indexvariable modifiziert wird, nachdem der Datensatz gelesen wurde, aber bevor das UPDATE USING SAME- (Reporting Mode) bzw. UPDATE-Statement (Structured Mode) ausgeführt wird, bedeutet dies, dass ein anderer Array-Bereich aktualisiert wird als gelesen wurde.</p>
SET/WITH <i>operand1=operand2</i>	<p>SET/WITH-Klausel:</p> <p>Mit dieser Klausel können Sie im Reporting Mode die Felder angeben, die geändert werden sollen, sowie die neuen Werte für diese Felder.</p> <p>Diese Klausel ist nicht erlaubt, wenn ein DEFINE DATA-Statement verwendet wird, da sich in diesem Fall das UPDATE-Statement immer auf den gesamten im DEFINE DATA-Statement definierten View bezieht.</p>

Beispiel

```

** Example 'UPDEX1S': UPDATE (structured mode)
**
** CAUTION: Executing this example will modify the database records!
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
*
1 #NAME (A20)
END-DEFINE
*
INPUT 'ENTER A NAME:' #NAME (AD=M)
IF #NAME = ' '
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH NAME = #NAME
  IF NO RECORDS FOUND
    REINPUT WITH 'NO RECORDS FOUND' MARK 1
  END-NOREC
  INPUT 'NAME:          ' NAME (AD=0) /
        'FIRST NAME:' FIRST-NAME (AD=M) /
        'CITY:          ' CITY (AD=M)
  UPDATE
  END TRANSACTION
END-FIND
*
END

```

Ausgabe des Programms SUBEX1S:

```
ENTER A NAME: BROWN
```

Nach Eingabe und Bestätigung des Namens:

NAME: BROWN
FIRST NAME: KENNETH
CITY: DERBY

Äquivalentes Reporting-Mode-Beispiel: **UPDEX1R**.

132

WRITE

▪ Funktion	880
▪ Syntax 1 — Dynamische Formatierung	881
▪ Syntax 1 - Beschreibung	881
▪ Syntax 2 — Vordefinierte Form/Map benutzen	889
▪ Syntax 2 — Beschreibung	890
▪ Beispiele	891

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE TITLE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `WRITE` dient dazu, Ausgaben in Freiformat zu erzeugen, die nicht bereits vorformatiert sind (vgl. `DISPLAY`-Statement).

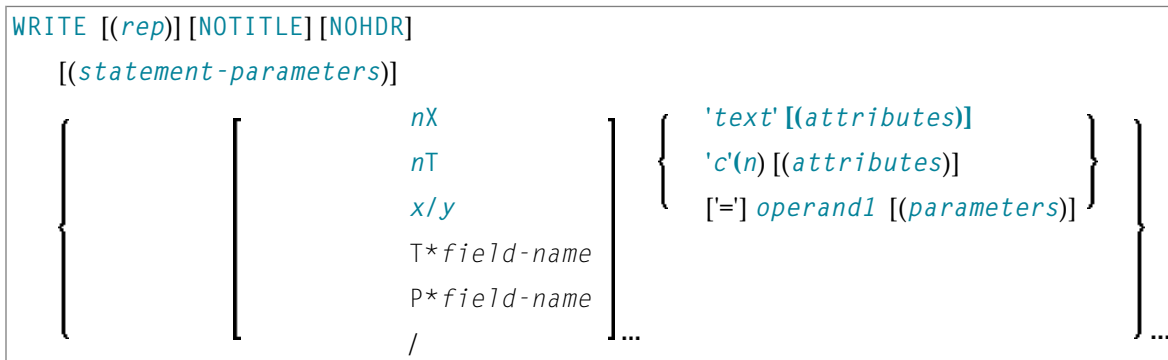
Das `WRITE`-Statement unterscheidet sich vom `DISPLAY`-Statement in folgenden Punkten:

- Passt ein Feld bzw. Textelement nicht mehr in eine Zeile, wird es automatisch in der nächsten Zeile ausgegeben. Ein Feld bzw. Textelement wird nicht auf zwei Zeilen verteilt.
- Es werden keine Standard-Spaltenüberschriften erzeugt. Die Ausgabelänge der Felder richtet sich nach der Länge der tatsächlich ausgegebenen Feldwerte.
- Mehrere Werte/Ausprägungen eines Arrays werden nicht untereinander sondern nebeneinander ausgegeben.

Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*:

- *Steuerung der Ausgabe von Daten*
- *Statements DISPLAY und WRITE*
- *Index-Notation für multiple Felder und Periodengruppen*
- *Beispiel für DISPLAY VERT mit WRITE-Statement*
- *Layout einer Ausgabeseite*

Syntax 1 — Dynamische Formatierung



Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax 1 – Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A U N P I F B D T L G O	ja	nein

Syntax-Element-Beschreibung:

(<i>rep</i>)	<p>Report-Spezifikation:</p> <p>Mit der Notation (<i>rep</i>) kann ein bestimmter Report angegeben werden, wenn ein Programm mehrere Ausgaben erzeugen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p> <p>Wenn (<i>rep</i>) nicht angegeben wird, bezieht sich das WRITE-Statement auf den ersten Report (Report 0).</p> <p>Wenn diese Druckdatei für Natural als PC definiert wird, wird der Report auf den PC heruntergeladen, siehe Beispiel 5.</p> <p>Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
----------------	---

NOTITLE	<p>Unterdrückung der Standard-Kopfzeile:</p> <p>Natural generiert für jede über ein WRITE-Statement ausgegebene Seite eine Kopfzeile. Diese Kopfzeile enthält die laufende Seitennummer, Uhrzeit und Datum. Die Uhrzeit wird zu Beginn der Programmausführung gesetzt. Die Ausgabe dieser Standard-Kopfzeile kann durch Angabe des Schlüsselwortes NOTITLE oder durch ein WRITE TITLE-Statement unterdrückt werden.</p> <p>Beispiele:</p> <ul style="list-style-type: none"> ■ Ausgabe einer Standard-Kopfzeile: <pre>WRITE NAME</pre> <ul style="list-style-type: none"> ■ Ausgabe einer eigenen Kopfzeile: <pre>WRITE NAME WRITE TITLE 'user-title'</pre> <ul style="list-style-type: none"> ■ Ausgabe ohne Kopfzeile: <pre>WRITE NOTITLE NAME</pre> <p>Anmerkung:</p> <ol style="list-style-type: none"> 1. Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben. 2. Natural prüft, wann ein Seitenvorschub erforderlich ist, bevor ein WRITE-Statement ausgeführt wird. Während der Ausführung eines WRITE-Statements werden keine neuen Seiten mit Kopf- oder Fußzeilen generiert.
NOHDR	<p>Unterdrückung der Spaltenüberschrift:</p> <p>Das WRITE-Statement selbst erzeugt keine Spaltenüberschriften. Wenn Sie allerdings das WRITE-Statement zusammen mit einem DISPLAY-Statement verwenden, können Sie mit der Option NOHDR des WRITE-Statements die vom DISPLAY-Statement generierten Spaltenüberschriften unterdrücken.</p> <p>Die NOHDR-Option ist nur relevant, wenn das WRITE-Statement nach einem DISPLAY-Statement steht, die Ausgabe sich insgesamt über mehr als eine Seite erstreckt und die Ausführung des WRITE-Statements zur Ausgabe einer neuen Seite führt.</p> <p>Ohne NOHDR-Option würden auf dieser neuen Seite die DISPLAY-Spaltenüberschriften ausgegeben, mit NOHDR werden sie dort nicht ausgegeben.</p>

<i>statement-parameters</i>	<p>Parameter-Definition auf Statement-Ebene:</p> <p>Unmittelbar nach dem WRITE-Statement können Sie auf Statement-Ebene in Klammern einzelne Session-Parameter angeben. Die Werte dieser Parameter haben dann für das betreffende Statement vor auf übergeordneter Ebene mittels GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT-Statement gesetzten Parameterwerten.</p> <p>Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Anmerkung: Die hier gültigen Parameter-Einstellungen kommen nur für Variablen-Felder in Betracht und haben keine Auswirkungen auf Text-Konstanten. Wenn Sie Feldattribute für eine Text-Konstante setzen möchten, dann müssen diese explizit für dieses Element gesetzt werden; siehe <i>Parameter-Definition auf Element-Ebene</i>.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ <i>Liste der Parameter</i> ■ <i>Beispiel für die Benutzung von Parametern auf Statement- und Element-Ebene</i> ■ <i>Beispiel 5 – WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene.</i>
<i>nX, nT, x/y,</i> <i>T*field-name,</i> <i>P*field-name, '=', l,</i>	<p>Notation Feld-Positionierung:</p> <p>Siehe <i>Feld-Positionierung</i> im Abschnitt <i>Formatierung der Ausgabe</i>.</p>
<i>'text', 'c'(n), attributes,</i> <i>operand1, parameters</i>	<p>Text/Attributzuweisung:</p> <p>Siehe <i>Text-, Attribut-Zuweisung, Ausgabe-Elemente</i> im Abschnitt <i>Formatierung der Ausgabe</i>.</p>

Liste der Parameter

Parameter, die mit dem WRITE-Statement angegeben werden können:		Spezifikation
		S = auf Statement-Ebene
		E =auf Element-Ebene
AD	Attribute Definition	SE
AL	Alphanumeric Length for Output	SE
CD	Color Definition	SE
CV	Control Variable	SE
DF	Date Format	SE

Parameter, die mit dem WRITE-Statement angegeben werden können:		Spezifikation
		S = auf Statement-Ebene
		E =auf Element-Ebene
DL	Display Length for Output	SE
DY	Dynamic Attributes	SE
EM	Edit Mask	SE
EMU	Unicode Edit Mask	E
FL	Floating Point Mantissa Length	SE
IS	Identical Suppress	SE
LS	Line Size	S
MC	Multiple-Value Field Count	S
MP	Maximum Number of Pages of a Report	S
NL	Numeric Length for Output	SE
PC	Periodic Group Count	S
PM	Print Mode	SE
PS	Page Size *	S
SG	Sign Position	SE
UC	Underlining Character	S
ZP	Zero Printing	SE

* Der Session-Parameter PS wird nicht berücksichtigt, wenn die Anzahl der Ausprägungen eines Arrays den PS-Wert überschreitet.

Ausführliche Beschreibungen der oben genannten Session-Parameter finden Sie in der *Parameter-Referenz*.

Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*:

- *Spaltenüberschriften zentrieren – der HC-Parameter*
- *Breite von Spaltenüberschriften – der HW-Parameter*
- *Füllzeichen für Überschriften – die Parameter FC und GC*
- *Unterstreichungszeichen für Überschriften – der UC-Parameter*

Beispiel für die Parameter-Benutzung auf Statement/Element-Ebene

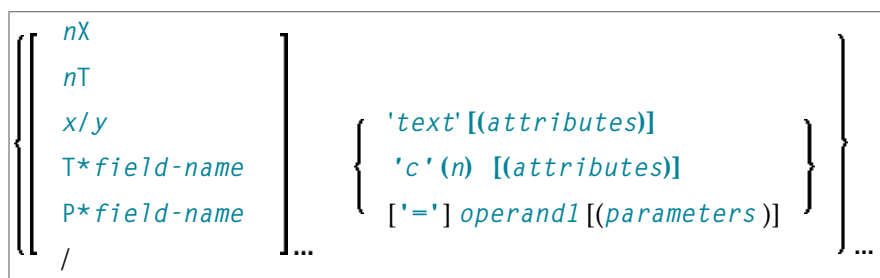
```

DEFINE DATA LOCAL
1 VARI (A4)      INIT <'1234'>          /*      Output
END-DEFINE      /*      Produced
*              /*      -----
WRITE           'Text'                 VARI          /*      Text 1234
WRITE (AD=U)    'Text'                 VARI          /*      Text  1234
WRITE           'Text' (AD=U)          VARI (AD=U)   /*      Text 1234
WRITE           'Text' (AD=U)          VARI          /*      Text 1234
END

```

Siehe auch [Beispiel 5 – WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene](#).

Formatierung der Ausgabe



Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Feld-Positionierung

<i>nX</i>	<p>Spaltenabstand:</p> <p>Mit der <i>nX</i> Notation können Sie zwischen zwei Feldern <i>n</i> Leerzeichen einfügen.</p> <p>Beispiel:</p> <pre>WRITE NAME 5X SALARY</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 2 – WRITE-Statement mit nX-, nT-Notation (weiter unten) ■ Spaltenabstand – der SF-Parameter und die Notation nX (im Leitfaden zur Programmierung)
<i>nT</i>	<p>Tabulator-Einstellungen:</p> <p>Mit der Notation <i>nT</i> setzen Sie Tabulatoren, d.h. die Ausgabe eines Feldes beginnt ab Spalte <i>n</i>. Ein Tabulator, der bereits durch eine andere Ausgabe belegt ist, darf nicht gesetzt werden.</p>

	<p>In dem folgenden Beispiel wird das Feld NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben:</p> <pre>WRITE 25T NAME 50T SALARY</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 2 – WRITE-Statement mit nX-, nT-Notation (weiter unten) ■ Tabulator-Notation nT (im Leitfaden zur Programmierung)
<i>x/y</i>	<p>x/y-Positionierung::</p> <p>Mit der Notation <i>x/y</i> erreichen Sie, dass ein Feld <i>x</i> Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte <i>y</i> ausgegeben wird. <i>y</i> darf nicht 0 sein. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.</p> <p>Siehe auch Positionierungsnotation x/y (im Leitfaden zur Programmierung).</p>
<i>T*field-name</i>	<p>Feldbezogene Positionierung:</p> <p>Mit der Notation <i>T*</i> können Sie die WRITE-Ausgabe nach der Position eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausrichten. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 3 – WRITE-Statement mit Notation T* (weiter unten) ■ Tabulator-Notation T*field (im Leitfaden zur Programmierung)
<i>P*field-name</i>	<p>Feld- und zeilenbezogene Positionierung:</p> <p>Mit der Notation <i>P*</i> können Sie die WRITE-Ausgabe nach der Position und Zeile eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (<i>field-name</i>) ausrichten. Diese Notation wird vor allem nach DISPLAY VERTICALLY-Statements verwendet. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Beispiel 4 – WRITE-Statement mit Notation P* (weiter unten) ■ Tabulator-Notation P*field (im Leitfaden zur Programmierung)
'='	<p>Feldinhalt hinter Feldüberschrift:</p> <p>Ein Gleichheitszeichen in Apostrophen ('=') vor einem Feld bewirkt, dass vor dem Feldwert die (im DEFINE DATA-Statement oder im DDM) für das Feld definierte Überschrift ausgegeben wird.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ WRITE-Statement mit '=', 'text', '/'

	<ul style="list-style-type: none"> ■ <i>WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene</i>
/	<p>Zeilenvorschub – Schrägstrich-Notation:</p> <p>Ein Schrägstrich (/) zwischen Feldern/Textelementen bewirkt einen Zeilenvorschub, d.h. die nachfolgenden Felder/ Textelemente werden in der nächsten Zeile ausgegeben.</p> <p>Beispiel:</p> <pre>WRITE NAME / SALARY</pre> <p>Für mehrfachen Zeilenvorschub geben Sie mehrere Schrägstriche an.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ <i>WRITE-Statement mit '=', 'text', '/'</i> (weiter unten) ■ <i>Zeilenvorschub – die Schrägstrich-Notation</i> (im Leitfaden zur Programmierung) ■ <i>Beispiel für Zeilenvorschub in WRITE-Statement</i> (im Leitfaden zur Programmierung)

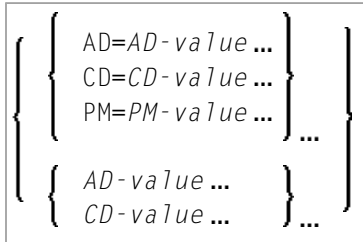
Text-, Attribut-Zuweisung, Ausgabe-Elemente

'text'	<p>Text-Zuweisung:</p> <p>Der in Apostrophen stehende Text wird ausgegeben.</p> <p>Beispiel:</p> <pre>WRITE 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT</pre> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ <i>WRITE-Statement mit '=', 'text', '/'</i> (weiter unten) ■ <i>Text Notation, Mit einem Statement zu benutzenden Text definieren - die 'text'-Notation</i> (im Leitfaden zur Programmierung)
'c'(n)	<p>Zeichen-Wiederholung:</p> <p>Das in Apostrophen stehende Zeichen (character) wird <i>n</i>-mal unmittelbar vor dem Feldwert ausgegeben.</p>

	<p>Zum Beispiel:</p> <pre>WRITE '*' (5) '=' NAME</pre> <p>führt zur Ausgabe von</p> <pre>***** SMITH</pre> <p>Siehe auch <i>Text-Notation</i>, Vor einem Feldwert <i>n</i> mal anzuzeigendes Zeichen definieren - die '<i>c</i>'(<i>n</i>)-Notation (im Leitfaden zur Programmierung).</p>
<i>attributes</i>	<p>Felddarstellung und Farbattribute:</p> <p>Es ist möglich, den auszugebenden Feldern/Texten Anzeige- und Farbattribute zuzuordnen. Diese Attribute und die zu benutzende Syntax sind im Abschnitt <i>Ausgabeattribute</i> weiter unten beschrieben.</p> <p>Beispiele:</p> <pre>WRITE 'TEXT' (BGR) WRITE 'TEXT' (B) WRITE 'TEXT' (BBLC)</pre>
<i>operand1</i>	<p>Name des auszugebenden Feldes:</p> <p><i>operand1</i> gibt das Feld an, dessen Inhalt an diese Stelle geschrieben wird.</p>
<i>parameters</i>	<p>Parameter-Definition auf Element-Ebene:</p> <p>Unmittelbar nach <i>operand1</i> können Sie auf Element-Ebene in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Feld Vorrang vor den mit einem GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT-Statement oder auf Statement-Ebene gesetzten Parameterwerten.</p> <p>Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine Parameterangabe darf sich jeweils nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Siehe auch:</p> <ul style="list-style-type: none"> ■ Liste der Parameter ■ Beispiel für die Parameter-Benutzung auf Statement- und Element-Ebene

Ausgabeattribute

Sie können den ausgegebenen Feldern/Textelementen Anzeige- und Farbattribute zuordnen. Sie können die folgenden Attribute angeben:



Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD* - Attribute Definition, Abschnitt *Feldanzeige*
- *CD* - Color Definition
- *PM* - Print Mode



Anmerkung: Der Compiler akzeptiert tatsächlich mehr als einem Attributwert für ein Ausgabefeld. Zum Beispiel können Sie Folgendes angeben: `AD=BDI`. In solch einem Fall gilt allerdings nur der letzte Wert. Im hier gezeigten Beispiel erhält nur der Wert `I` Gültigkeit, und das Ausgabefeld wird intensiviert (hell hervorgehoben) angezeigt.

Syntax 2 — Vordefinierte Form/Map benutzen

```
WRITE [(rep)] [NOTITLE] [NOHDR] [USING] { FORM } operand1 [operand2 ...]
                                     { MAP }
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax 2 — Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S	A	nein	nein
<i>operand2</i>	S A G N	A U N P I F B D T L	ja	nein

Syntax-Element-Beschreibung:

FORM/MAP	<p>Benutzung des vordefinierten Form/Map-Layouts:</p> <p>Diese Option verwenden Sie, wenn Sie für die Ausgabe eine (mit dem Natural Map Editor erstellte) Map verwenden wollen.</p> <p>WRITE USING MAP bedeutet nicht, dass jedesmal, wenn die Map ausgegeben wird, automatisch eine neue Seite ausgegeben wird.</p> <p>Für den Zeilenabstand muss der Parameter LS um ein Byte größer gesetzt werden als die Zeilenlänge der Map.</p>
<i>operand1</i>	<p>Form/Map-Name:</p> <p><i>operand1</i> ist der Name der zu verwendenden Map.</p>
<i>operand2</i>	<p>Auszugebendes Feld:</p> <p><i>operand2</i> ist der Name des auszugebenden Feldes bzw. der auszugebenden Felder.</p> <p>Ist <i>operand1</i> eine Konstante und wird <i>operand2</i> nicht angegeben, so werden bei der Kompilierung die Felder aus der Map-Source übernommen.</p> <p>Die Felder müssen bezüglich Anzahl, Reihenfolge, Format, Länge und (bei Arrays) Anzahl der Ausprägungen mit den Feldern in de/mr referenzierten Layout/Map übereinstimmen, sonst tritt ein Fehler auf.</p>
NOTITLE/NOHDR	<p>Unterdrückung der Kopfzeile/Spaltenüberschrift:</p> <p>Die Optionen NOTITLE und NOHDR sind unter <i>Syntax 1</i> des WRITE-Statements beschrieben.</p>

Beispiele

- Beispiel 1 – WRITE-Statement mit '=', 'text', '/'
- Beispiel 2 – WRITE-Statement mit nX-, nT-Notation
- Beispiel 3 – WRITE-Statement mit Notation T*
- Beispiel 4 – WRITE-Statement mit Notation P*
- Beispiel 5 – WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene
- Beispiel 6 – Report-Spezifikation mit für Natural als PC definierter Ausgabedatei

Beispiel 1 – WRITE-Statement mit '=', 'text', '/'

```

** Example 'WRTEX1': WRITE (with '=', 'text', '/')
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 FULL-NAME
    3 FIRST-NAME
    3 MIDDLE-I
    3 NAME
  2 CITY
  2 COUNTRY
END-DEFINE
*
LIMIT 1
READ EMPL-VIEW BY NAME
  /*
  WRITE NOTITLE
    '=' NAME '=' FIRST-NAME '=' MIDDLE-I //
    'L O C A T I O N' /
    'CITY: ' CITY /
    'COUNTRY:' COUNTRY //
  /*
END-READ
END

```

Ausgabe des Programms WRTEX1:

```

NAME: ABELLAN                FIRST-NAME: KEPA                MIDDLE-I:
L O C A T I O N
CITY:   MADRID
COUNTRY: E

```

Beispiel 2 – WRITE-Statement mit nX-, nT-Notation

```

** Example 'WRTEX2': WRITE (with nX, nT notation)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 JOB-TITLE
END-DEFINE
*
LIMIT 4
READ EMPL-VIEW BY NAME
  WRITE NOTITLE 5X NAME 50T JOB-TITLE
END-READ
END

```

Ausgabe des Programms WRTEX2:

ABELLAN	MAQUINISTA
ACHIESON	DATA BASE ADMINISTRATOR
ADAM	CHEF DE SERVICE
ADKINSON	PROGRAMMER

Beispiel 3 – WRITE-Statement mit Notation T*

```

** Example 'WRTEX3': WRITE (with T* notation)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY (1)
END-DEFINE
*
LIMIT 5
READ EMPL-VIEW BY CITY STARTING FROM 'ALBU'
  DISPLAY NOTITLE CITY NAME SALARY (1)
  AT BREAK CITY
  /*
  WRITE / 'CITY AVERAGE:' T*SALARY (1) AVER(SALARY(1)) //
  /*
  END-BREAK
END-READ
END

```

Ausgabe des Programms WRTEX3:

CITY	NAME	ANNUAL SALARY
ALBUQUERQUE	HAMMOND	22000
ALBUQUERQUE	ROLLING	34000
ALBUQUERQUE	FREEMAN	34000
ALBUQUERQUE	LINCOLN	41000
CITY AVERAGE:		32750
ALFRETON	GOLDBERG	4800
CITY AVERAGE:		4800

Beispiel 4 – WRITE-Statement mit Notation P*

```

** Example 'WRTEX4': WRITE (with P* notation)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 BIRTH
  2 SALARY (1)
END-DEFINE
*
LIMIT 3
READ EMPL-VIEW BY CITY FROM 'N'
  DISPLAY NOTITLE NAME CITY
    VERT AS 'BIRTH/SALARY' BIRTH (EM=YYYY-MM-DD) SALARY (1)
  SKIP 1
  AT BREAK CITY
    WRITE / 'CITY AVERAGE' P*SALARY (1) AVER(SALARY (1)) //
  END-BREAK
END-READ
END

```

Ausgabe des Programms WRTEX4:

WRITE

NAME	CITY	BIRTH SALARY
-----	-----	-----
WILCOX	NASHVILLE	1970-01-01 38000
MORRISON	NASHVILLE	1949-07-10 36000
CITY AVERAGE		37000
BOYER	NEMOURS	1955-11-23 195900
CITY AVERAGE		195900

Beispiel 5 – WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene

```
** Example 'WRTEX5': WRITE (using '=', statement/element parameters)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
  2 PHONE
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY NAME
  WRITE NOTITLE (AL=16 NL=8)
    '=' PERSONNEL-ID '=' NAME '=' PHONE (AL=10 EM=XXX-XXXXXX)
END-READ
END
```

Ausgabe des Programms WRTEX5:

```
PERSONNEL ID: 60008339      NAME: ABELLAN      TELEPHONE: 435-6726
PERSONNEL ID: 30000231      NAME: ACHIESON     TELEPHONE: 523-341
```

Beispiel 6 – Report-Spezifikation mit für Natural als PC definierter Ausgabedatei

```
** Example 'PCDIEX1': DISPLAY and WRITE to PC
**
** NOTE: Example requires that Natural Connection is installed.
*****
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
END-DEFINE
*
FIND PERS WITH CITY = 'NEW YORK'           /* Data selection
  WRITE (7) TITLE LEFT 'List of employees in New York' /
  DISPLAY (7)                               /* (7) designates the output file (here the PC).
    'Location' CITY
    'Surname'  NAME
    'ID'       PERSONNEL-ID
END-FIND
END
```


133

WRITE TITLE

▪ Funktion	898
▪ Einschränkungen	899
▪ Syntax-Beschreibung	899
▪ Beispiel	903

```
WRITE [(rep)] TITLE [LEFT [JUSTIFIED]] [UNDERLINED]
    [(statement-parameters)]
    { [ [ nX ] ] ... { 'text' [(attributes)] } }
    { [ [ nT ] ] ... { 'c'(n) [(attributes)] } }
    { [ [ x/y ] ] ... { ['='] operand1 [(parameters)] } } ...
    [SKIP operand2 [LINES]]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TRAILER

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `WRITE TITLE` dient dazu, statt einer Standard-Kopfzeile eine eigene Seitenüberschrift auszugeben. Das Statement wird immer dann ausgeführt, wenn eine neue Ausgabeseite initiiert wird.

Siehe auch die folgenden Abschnitte (im *Leitfaden zur Programmierung*):

- *Steuerung der Ausgabe von Daten*
- *Report-Spezifikation – Notation (rep)*
- *Layout einer Ausgabeseite*
- *Seitenüberschriften, Seitenvorschübe und Leerzeilen*
- *Eigene Seitenüberschrift definieren – das WRITE TITLE-Statement*
- *Text-Notation*

Verarbeitung

Dieses Statement ist nicht-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Wenn ein Report durch Statements in verschiedenen Objekten erzeugt wird, wird das `WRITE TITLE`-Statement nur ausgeführt, wenn es in demselben Objekt steht wie das Statement, das die Ausgabe einer neuen Seite auslöst.

Einschränkungen

- `WRITE TITLE` darf höchstens einmal pro Ausgabe-Report verwendet werden.
- `WRITE TITLE` darf nicht an eine logische Bedingung geknüpft sein.
- `WRITE TITLE` darf nicht in einer Subroutine stehen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A U N P I F B D T L G O	ja	nein
<i>operand2</i>	C S	N P I B	ja	nein

Syntax-Element-Beschreibung:

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Erzeugt ein Programm mehrere Reports, kann mit der Notation <i>(rep)</i> ein bestimmter anderer Report angegeben werden, auf den sich das Statement beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem <code>DEFINE PRINTER</code>-Statement zugewiesen wurde, angegeben werden.</p> <p>Wenn <i>(rep)</i> nicht angegeben wird, bezieht sich das Statement <code>WRITE TITLE</code> auf den ersten Report (Report 0).</p> <p>Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
--------------	---

<p>LEFT JUSTIFIED UNDERLINED</p>	<p>Kopfzeilenausrichtung und Unterstreichung:</p> <p>Normalerweise werden Seitenkopfzeilen zentriert und ohne Unterstreichung ausgegeben.</p> <p>Eine linksbündige Ausrichtung der Kopfzeile erreichen Sie durch Angabe des Schlüsselwortes <code>LEFT JUSTIFIED</code>.</p> <p>Eine unterstrichene Kopfzeile erhalten Sie durch Angabe des Schlüsselwortes <code>UNDERLINED</code>; als Unterstreichungszeichen wird das mit dem Parameter <code>UC</code> (auf Session-Ebene oder in einem <code>FORMAT</code>-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Kopfzeile (entsprechend der mit dem Parameter <code>LS</code> definierten Zeilenlänge).</p> <p>Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Bei einer Tabulator-Notation von <code>10T</code>, zum Beispiel, rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.</p>
<p><i>statement-parameters</i></p>	<p>Parameter-Definition auf Statement-Ebene:</p> <p>Unmittelbar nach dem <code>WRITE TITLE</code>-Statement können Sie auf Statement-Ebene in Klammern einzelne Session-Parameter angeben. Die Werte dieser Parameter haben dann für das betreffende Statement Vorrang vor auf übergeordneter Ebene mittels <code>GLOBALS</code>-Kommando, <code>SET GLOBALS</code>- (nur im Reporting Mode) oder <code>FORMAT</code>-Statement gesetzten Parameterwerten.</p> <p>Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Anmerkung: Die hier gültigen Parameter-Einstellungen werden nur bei Variablenfelder berücksichtigt haben und keine Auswirkung auf Textkonstanten. Wenn Sie Feldattribute für eine Textkonstante setzen möchten, müssen sie explizit für dieses Element gesetzt werden; siehe <i>Parameter-Definition auf Element-Ebene</i>.</p> <p>Informationen zur Benutzung der Parameter, siehe <i>Liste der Parameter</i> (beim <code>WRITE</code>-Statement).</p>
<p><i>nX</i> <i>nT</i> <i>x/y</i></p>	<p>Format-Notation und Abstandselemente:</p> <p>Siehe <i>Format-Notation und Abstandselemente</i> (weiter unten).</p>
<p>'text' 'c' (n) <i>attributes</i></p>	<p>Text/Attribut-Zuweisung:</p> <p>Siehe <i>Text/Attribut-Zuweisungen</i> (weiter unten).</p>

<i>operand1</i>	<p>In der Überschrift anzuzeigendes Feld:</p> <p>Als <i>operand1</i> können Sie ein oder mehrere Feld/er angeben, die in der Kopfzeile ausgegeben werden sollen.</p>
<i>parameters</i>	<p>Parameter-Definition auf Element-Ebene:</p> <p>Ein einzelner oder mehrere in Anführungszeichen stehende Parameter kann/können auf Element-Ebene unmittelbar hinter <i>operand1</i> angegeben werden. Jeder auf diese Art angegebene Parameter überschreibt den entsprechenden, vorher auf Statement-Ebene oder in einem GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT-Statement angegebenen Parameter.</p> <p>Wenn mehr als ein Parameter angegeben wird, müssen ein oder mehr Leerzeichen zwischen jedem Eintrag stehen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Informationen zur Benutzung der Parameter siehe Liste der Parameter (beim WRITE-Statement).</p>
SKIP <i>operand2</i> LINES	<p>Einfügen von Leerzeilen nach der Kopfzeile:</p> <p>Mit der SKIP-Klausel können Sie nach der Kopfzeile Leerzeilen einzufügen. Die Anzahl der einzufügenden Leerzeilen kann als numerische Konstante oder als Inhalt einer numerischen Variablen angegeben werden.</p> <p>Anmerkung: SKIP nach WRITE TITLE wird immer als SKIP-Klausel des WRITE TITLE-Statements interpretiert, und nicht als ein eigenständiges Statement. Falls Sie ein eigenständiges SKIP-Statement nach einem WRITE TITLE-Statement wünschen, trennen Sie die beiden Statements durch ein Semikolon (;) voneinander.</p>

Format-Notation und Abstandselemente

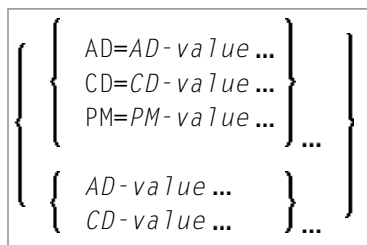
<i>nX</i>	<p>Spaltenabstand:</p> <p>Mit dieser Notation fügen Sie <i>n</i> Leerzeichen zwischen den Spalten ein.</p>
<i>nT</i>	<p>Setzen der Tabulatoren:</p> <p>Die Notation <i>nT</i> bewirkt die Positionierung (Tabulierung) an die Druck-Position <i>n</i>.</p> <p>Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.</p>
<i>x/y</i>	<p><i>x/y</i>-Positionierung:</p> <p>Mit dieser Notation erreichen Sie, dass ein Feld <i>x</i> Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte <i>y</i> ausgegeben wird. <i>y</i> darf nicht 0 sein.</p> <p>Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.</p>

Text/Attribut-Zuweisungen

<code>'text'</code>	<p>Text-Zuweisung:</p> <p>Es wird die in Apostrophen stehende Zeichenkette angezeigt.</p>
<code>'c'(n)</code>	<p>Zeichen-Wiederholung:</p> <p>Das in Apostrophen stehende Zeichen (<i>character</i>) wird unmittelbar vor dem Feldwert <i>n</i> mal angezeigt.</p>
<code>attributes</code>	<p>Felddarstellung und Farbattribute:</p> <p>Es ist möglich, verschiedene Attribute für die Text/Feldanzeige zuzuweisen. Diese Attribute und die Syntax, die benutzt werden kann, sind im Abschnitt Ausgabeattribute weiter unten beschrieben.</p> <p>Beispiele:</p> <pre style="background-color: #e0e0e0; padding: 5px;">WRITE TITLE 'TEXT' (BGR) WRITE TITLE 'TEXT' (B) WRITE TITLE 'TEXT' (BBLC)</pre>

Ausgabeattribute

`attributes` gibt die für die Text-Anzeige zu benutzenden Ausgabe-Attribute an. Es gibt die folgenden Attribute:



Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD* - Attribute Definition, Abschnitt *Feldanzeige*
- *CD* - Color Definition
- *PM* - Print Mode

Anmerkung: Der Compiler akzeptiert tatsächlich mehr als ein Attributwert für ein Ausgabefeld. Zum Beispiel können Sie Folgendes angeben: `AD=BDI`. In solch einem Fall gilt allerdings nur der letzte Wert. Im hier gezeigten Beispiel erhält nur der Wert `I` Gültigkeit, und das Ausgabefeld wird intensiviert (hell hervorgehoben) angezeigt.

Beispiel

```

** Example 'WTIEX1': WRITE (with TITLE option)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
END-DEFINE
*
*
FORMAT LS=70
*
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN NEW YORK CITY'
      11X 'PAGE:' *PAGE-NUMBER
SKIP 1
*
FIND EMPL-VIEW WITH CITY = 'NEW YORK'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
END-FIND
END

```

Ausgabe des Programms WTIEX1:

```

09:33:16.5  PEOPLE LIVING IN NEW YORK CITY                PAGE:      1
-----
          NAME                FIRST-NAME                CURRENT
                               POSITION
-----
RUBIN                SYLVIA                SECRETARY
WALLACE              MARY                ANALYST

```


134

WRITE TRAILER

▪ Funktion	906
▪ Einschränkungen	907
▪ Syntax-Beschreibung	907
▪ Beispiel	911

```

WRITE [(rep)] [TRAILER LEFT [JUSTIFIED]] [UNDERLINED]
      [(statement-parameters)]
      { [ nX ] { 'text' [(attributes)] } }
      [ nT ] { 'c'(n) [(attributes)] } }
      [ x/y ] ... [ [=] operand1 [(parameters)] } ...
      [SKIP operand2 [LINES]]

```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: AT END OF PAGE | AT TOP OF PAGE | CLOSE PRINTER | DEFINE PRINTER | DISPLAY | EJECT | FORMAT | NEWPAGE | PRINT | SKIP | SUSPEND IDENTICAL SUPPRESS | WRITE | WRITE TITLE

Gehört zur Funktionsgruppe: *Erstellen von Ausgabe-Reports*

Funktion

Das Statement `WRITE TRAILER` dient dazu, am Ende einer Ausgabeseite eine Fußzeile auszugeben.

Siehe auch die folgenden Abschnitte im *Leitfaden zur Programmierung*:

- *Steuerung der Ausgabe von Daten*
- *Report-Spezifikation – Notation (rep)*
- *Layout einer Ausgabeseite*
- *Seiten-Fußzeile – das WRITE TRAILER-Statement*
- *Text-Notation*

Verarbeitung

Dieses Statement ist nicht-prozedural (das heißt, seine Ausführung hängt von einem Ereignis ab, nicht davon, wo im Programm es steht).

Das Statement wird immer dann ausgeführt, wenn eine "End-of-Page"- oder "End-of-Data"-Bedingung auftritt, oder wenn aufgrund eines `SKIP`- oder `NEWPAGE`-Statements ein Seitenvorschub erfolgt. Es wird nicht ausgeführt, wenn ein Seitenvorschub aufgrund eines `EJECT`-Statements erfolgt.

Ob eine End-of-Page-Bedingung gegeben ist, wird erst überprüft, nachdem ein `DISPLAY`- oder `WRITE`-Statement vollständig ausgeführt ist; ist die logische Seitenlänge nicht richtig gesetzt, kann

es vorkommen, dass die DISPLAY-/WRITE-Ausgabe bereits das Ende einer physischen Ausgabeseite überschritten hat, bevor auf der logischen Seite eine End-of-Page-Bedingung auftritt.

Wenn ein Report durch Statements in verschiedenen Objekten erzeugt wird, wird das WRITE TRAILER-Statement nur ausgeführt, wenn es in demselben Objekt steht wie das Statement, das die End-of-Page-Bedingung auslöst.

Logische Seitenlänge

Um sicherzustellen, dass eine mit WRITE TRAILER definierte Fußzeile noch auf eine ausgegebene physische Seite passt, sollte die Länge der vom Programm erzeugten logischen Seite (mittels des Session-Parameters PS) entsprechend kleiner als die physische Seitenlänge gesetzt werden.

Einschränkungen

- WRITE TRAILER darf höchstens einmal pro Ausgabe-Report verwendet werden.
- WRITE TRAILER darf nicht an eine logische Bedingung geknüpft sein.
- WRITE TRAILER darf nicht in einer Subroutine stehen.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A G N	A U N P I F B D T L G O	ja	nein
<i>operand2</i>	C S	N P I B	ja	nein

Syntax-Element-Beschreibung:

<i>(rep)</i>	<p>Report-Spezifikation:</p> <p>Erzeugt ein Programm mehrere Reports, kann mit der Notation (<i>rep</i>) ein bestimmter anderer Report angegeben werden, auf den sich das Statement WRITE TRAILER beziehen soll.</p> <p>Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden.</p>
--------------	--

	<p>Wenn (<i>rep</i>) nicht angegeben wird, bezieht sich das Statement <code>WRITE TRAILER</code> auf den ersten Report (Report 0).</p> <p>Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt <i>Steuerung der Ausgabe von Daten im Leitfaden zur Programmierung</i>.</p>
LEFT JUSTIFIED UNDERLINED	<p>Fußzeilenausrichtung und Unterstreichung:</p> <p>Normalerweise werden Fußzeilen zentriert und ohne Unterstreichung ausgegeben.</p> <p>Eine linksbündige Ausrichtung der Fußzeile erreichen Sie durch die Angabe des Schlüsselwortes <code>LEFT JUSTIFIED</code>.</p> <p>Eine unterstrichene Fußzeile erhalten Sie durch Angabe des Schlüsselwortes <code>UNDERLINED</code>; als Unterstreichungszeichen wird das mit dem Parameter <code>UC</code> (auf Session-Ebene oder in einem <code>FORMAT</code>-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Fußzeile (entsprechend der mit dem Parameter <code>LS</code> definierten Zeilenlänge).</p> <p>Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Bei einer Tabulator-Notation von <code>10T</code>, zum Beispiel, rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.</p>
<i>statement-parameters</i>	<p>Parameter-Definition auf Statement-Ebene:</p> <p>Unmittelbar nach dem <code>WRITE TRAILER</code>-Statement können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Statement Gültigkeit vor auf übergeordneter Ebene mittels <code>GLOBALS</code>-Kommando, <code>SET GLOBALS</code>- (nur im Reporting Mode) oder <code>FORMAT</code>-Statement gesetzten Parameterwerten.</p> <p>Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken.</p> <p>Anmerkung: Die hier gültigen Parameter-Einstellungen werden nur bei Variablenfelderberücksichtigt, haben und keine Auswirkung auf Textkonstanten. Wenn Sie Feldattribute für eine Textkonstante setzen möchten, müssen sie explizit für dieses Element gesetzt werden; siehe <i>Parameter-Definition auf Element-Ebene</i>.</p> <p>Informationen zur Benutzung der Parameter siehe <i>Liste der Parameter</i> in der <code>WRITE</code>-Statement-Dokumentation.</p>
<i>nX</i> <i>nT</i> <i>x/y</i>	<p>Format-Notation und Abstandselemente:</p> <p>Siehe <i>Format-Notation und Abstandselemente</i> (weiter unten).</p>

'text' 'c'(n) attributes	Text/Attribut-Zuweisung: Siehe Text/Attribut-Zuweisungen (weiter unten).
operand1	Fußzeilen-Informationen: Als <i>operand1</i> können Sie ein oder mehrere Feld/er angeben, die in der Fußzeile ausgegeben werden sollen.
parameters	Parameter-Definition auf Element-Ebene: Ein einzelner oder mehrere in Anführungszeichen stehende Parameter können auf Element-Ebene unmittelbar hinter <i>operand1</i> angegeben werden. Jeder auf diese Art angegebene Parameter überschreibt den entsprechenden, vorher auf Statement-Ebene oder in einem GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT- Statement angegebenen Parameter. Wenn mehr als ein Parameter angegeben wird, müssen ein oder mehr Leerzeichen zwischen jedem Eintrag stehen. Eine einzelne Parameterangabe darf sich nicht über zwei Sourcecode-Zeilen erstrecken. Informationen zur Benutzung der Parameter; siehe Liste der Parameter (beim WRITE-Statement).
SKIP <i>operand2</i> LINES	Einfügen von Leerzeilen nach der Fußzeile: Mit der SKIP-Klausel können Sie nach der Fußzeile Leerzeilen einfügen. Mit <i>operand2</i> geben Sie an, wieviele Leerzeilen auf die Fußzeile folgen sollen; dies kann entweder eine numerische Konstante oder der Inhalt einer numerischen Variablen sein. Anmerkung: SKIP nach WRITE TRAILER wird immer als SKIP-Klausel des WRITE TRAILER-Statements interpretiert, und nicht als eigenständiges Statement. Falls Sie ein eigenständiges SKIP-Statement nach einem WRITE TRAILER-Statement wünschen, trennen Sie die beiden Statements durch ein Semikolon (;) voneinander.

Format-Notation und Abstandselemente

<i>nX</i>	Spaltenabstand: Mit dieser Notation fügen Sie <i>n</i> Leerzeichen zwischen den Spalten ein.
<i>nT</i>	Setzen der Tabulatoren: Die Notation <i>nT</i> bewirkt die Positionierung (Tabulierung) an die Druck-Position <i>n</i> . Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.
<i>x/y</i>	x/y-Positionierung:

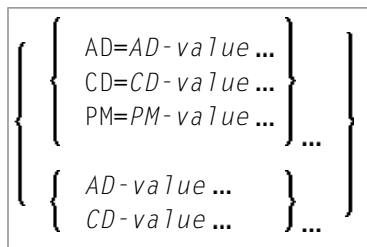
	<p>Mit dieser Notation erreichen Sie, dass ein Feld x Zeilen unter der Ausgabe des letzten Statements, und zwar ab Spalte y ausgegeben wird. y darf nicht 0 sein.</p> <p>Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.</p>
--	---

Text/Attribut-Zuweisungen

<i>'text'</i>	<p>Text-Zuweisung:</p> <p>Es wird die in Apostrophen stehende Zeichenkette angezeigt.</p>
<i>'c'(n)</i>	<p>Zeichen-Wiederholung:</p> <p>Das in Apostrophen stehende Zeichen (character) wird unmittelbar vor dem Feldwert n mal angezeigt.</p>
<i>attributes</i>	<p>Felddarstellung und Farbattribute:</p> <p>Es ist möglich, verschiedene Attribute für die Text/Feldanzeige zuzuweisen. Diese Attribute und die Syntax, die benutzt werden kann, sind im Abschnitt <i>Ausgabeattribute</i> weiter unten beschrieben.</p> <p>Beispiele:</p> <pre style="background-color: #cccccc; padding: 5px;">WRITE TRAILER 'TEXT' (BGR) WRITE TRAILER 'TEXT' (B) WRITE TRAILER 'TEXT' (BBLC)</pre>

Ausgabeattribute

attributes gibt die für die Text-Anzeige zu benutzenden Ausgabe-Attribute an. Es gibt die folgenden Attribute:



Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD - Attribute Definition, Abschnitt Felddarstellung*
- *CD - Color Definition*
- *PM - Print Mode*



Anmerkung: Der Compiler akzeptiert tatsächlich mehr als einen Attributwert für ein Ausgabefeld. Zum Beispiel können Sie Folgendes angeben: AD=BDI. In solch einem Fall gilt allerdings nur der letzte Wert. Im hier gezeigten Beispiel erhält nur der Wert I Gültigkeit, und das Ausgabefeld wird intensiviert (hell hervorgehoben) angezeigt.

Beispiel

```
** Example 'WTLEX1': WRITE (with TRAILER option)
*****
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 JOB-TITLE
END-DEFINE
*
FORMAT PS=15
WRITE TITLE LEFT JUSTIFIED UNDERLINED
      *TIME 3X 'PEOPLE LIVING IN BARCELONA'
      14X 'PAGE:' *PAGE-NUMBER
SKIP 1
*
WRITE TRAILER LEFT JUSTIFIED UNDERLINED
      / 'CITY OF BARCELONA REGISTER'
*
LIMIT 10
FIND EMPL-VIEW WITH CITY = 'BARCELONA'
  DISPLAY NAME FIRST-NAME 3X JOB-TITLE
END-FIND
END
```

Ausgabe des Programms WTLEX1 - Seite 1:

```
09:36:09.5  PEOPLE LIVING IN BARCELONA                PAGE:      1
-----
          NAME                FIRST-NAME                CURRENT
                               POSITION
-----
DEL CASTILLO                ANGEL                EJECUTIVO DE VENTAS
GARCIA                       M. DE LAS MERCEDES  SECRETARIA
GARCIA                       ENDIKA              DIRECTOR TECNICO
MARTIN                       ASUNCION            SECRETARIA
MARTINEZ                     TERESA              SECRETARIA
```

WRITE TRAILER

YNCLAN	FELIPE	ADMINISTRADOR
FERNANDEZ	ELOY	OFICINISTA
TORRES	ANTONI	OBRERA

CITY OF BARCELONA REGISTER

Ausgabe des Programms WTLEX1 - Seite 2:

09:37:26.0 PEOPLE LIVING IN BARCELONA PAGE: 2

NAME	FIRST-NAME	CURRENT POSITION
RODRIGUEZ	VICTORIA	SECRETARIA
GARCIA	GERARDO	INGENIERO DE PRODUCCION

CITY OF BARCELONA REGISTER

135

WRITE WORK FILE

▪ Funktion	914
▪ Syntax-Beschreibung	914
▪ Externe Darstellung der Felder	915
▪ Verarbeitung großer und dynamischer Variablen	916
▪ Beispiel	917

```
WRITE WORK [FILE] work-file-number [VARIABLE] operand1 ...
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Verwandte Statements: [DEFINE WORK FILE](#) | [READ WORK FILE](#) | [CLOSE WORK FILE](#)

Gehört zur Funktionsgruppe: [Verarbeitung von Arbeitsdateien/PC-Dateien](#)

Funktion

Das Statement `WRITE WORK FILE` dient dazu, Datensätze auf eine physisch-sequentielle Arbeitsdatei (Work File) zu schreiben.

Dieses Statement kann nur im Batch-Betrieb verwendet werden..

Es ist möglich, in einem Programm oder einer Verarbeitungsschleife eine Arbeitsdatei zu erstellen und diese dann in einem anderen Programm oder einer anderen eigenständigen Verarbeitungsschleife mit einem `READ WORK FILE`-Statement zu lesen.



Anmerkung: Bezüglich Unicode-Support siehe *Work Files and Print Files on Windows, UNIX and OpenVMS Platforms* in der *Unicode and Code Page Support*-Dokumentation.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	C S A G	A U N P I F B D T L C G	ja	nein



Anmerkung: Bei den Arbeitsdateitypen `ENTIRECONNECTION` oder `TRANSFER` darf *operand1* nicht im Format `C` oder `G` sein.

Syntax-Element-Beschreibung:

<i>work-file-number</i>	<p>Arbeitsdateinummer:</p> <p>Die für Natural definierte Nummer der Arbeitsdatei, die verwendet werden soll.</p>
VARIABLE	<p>Variablen-Eintrag:</p> <p>Es ist möglich, mittels verschiedener WRITE WORK FILE-Statements Datensätze mit verschiedenen Feldern auf dieselbe Arbeitsdatei zu schreiben. In diesem Fall müssen alle betreffenden WRITE WORK FILE-Statements das Schlüsselwort VARIABLE enthalten; die Datensätze werden dann mit variablem Format auf die externe Datei geschrieben.</p> <p>Wenn die Operanden-Liste eine dynamische Variable enthält (die je nach Ausführungsart des WRITE WORK FILE-Statements eine andere Größe annimmt), muss der VARIABLE-Eintrag in allen WRITE WORK FILE-Statements angegeben werden.</p> <p>Variabler Indexbereich:</p> <p>Wenn Sie ein Array auf eine Arbeitsdatei schreiben, können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel:</p> <pre>WRITE WORK FILE <i>work-file-number</i> VARIABLE #ARRAY (I:J)</pre>
<i>operand1</i>	<p>Felder:</p> <p>Als <i>operand1</i> geben Sie die Felder an, die auf die Arbeitsdatei geschrieben werden sollen. Dies können entweder Datenbankfelder, Benutzervariablen, Systemvariablen und/oder Felder sein, die mit einem READ WORK FILE-Statement von einer anderen Arbeitsdatei gelesen wurden.</p> <p>Bei Datenbank-Arrays kann durch eine einen Bereich umfassende Indexierung angegeben werden, welche Ausprägungen auf die Arbeitsdatei geschrieben werden sollen. Feldgruppen können durch Angabe des Gruppennamens referenziert werden; alle Felder einer Gruppe werden einzeln auf die Arbeitsdatei geschrieben.</p>

Externe Darstellung der Felder

Mit einem WRITE WORK FILE-Statement auf eine Arbeitsdatei geschriebene Felder werden auf der externen Datei entsprechend ihrer internen Definition dargestellt. Die Feldwerte werden nicht verändert.

Bei Feldern der Formate A oder B entspricht die Anzahl der Bytes auf der externen Datei der programminternen Längendefinition. Die Feldwerte werden nicht verändert; ein Komma (Dezimalpunkt) wird nicht wiedergegeben.

Bei Feldern des Formats N ergibt sich die Anzahl der Bytes auf der externen Datei aus der Summe der Stellen vor und nach dem Komma. Das Komma (Dezimalpunkt) wird auf der externen Datei nicht wiedergegeben.

Bei Feldern des Formats P ergibt sich die Anzahl der Bytes auf der externen Datei aus der Summe der Stellen vor und nach dem Komma plus einer Stelle für das Vorzeichen, geteilt durch 2, wobei auf ganze Bytes aufgerundet wird.



Anmerkung: Beim Schreiben von Feldern auf eine Arbeitsdatei erfolgt keine Umsetzung von Feldformaten.

Beispiele für Felddarstellung:

Fellddefinition	Ausgabelänge
#FIELD1 (A10)	10 Bytes
#FIELD2 (B15)	15 Bytes
#FIELD3 (N1.3)	4 Bytes
#FIELD4 (N0.7)	7 Bytes
#FIELD5 (P1.2)	2 Bytes
#FIELD6 (P6.0)	4 Bytes



Anmerkung: Wenn die Systemfunktionen AVER, NAVER, SUM oder TOTAL für numerische Felder (Format N oder P) auf eine Arbeitsdatei geschrieben werden, vergrößert sich intern die Länge dieser Felder um eine Stelle (z.B.: SUM eines Feldes vom Format P3 wird auf P4 verlängert). Dies ist beim Lesen der Arbeitsdatei zu berücksichtigen.

Verarbeitung großer und dynamischer Variablen

Arbeitsdateityp	Verarbeitung
ASCII ASCII-COMPRESSED SAG (binär)	Die Arbeitsdateitypen ASCII, ASCII-COMPRESSED und SAG (binär) können keine dynamischen Variablen verarbeiten und rufen einen Fehler hervor. Sie können jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 32766 Bytes verarbeiten.
ENTIRECONNECTION	Der Arbeitsdateityp ENTIRECONNECTION kann keine dynamische Variablen verarbeiten. Er kann jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 1073741824 Bytes verarbeiten.
PORTABLE UNFORMATTED	Große und dynamische Variablen können mit den beiden Arbeitsdateitypen PORTABLE und UNFORMATTED in Arbeitsdateien geschrieben oder aus Arbeitsdateien gelesen werden. Bei diesen Typen gibt es keine Größenbeschränkung für dynamische Variablen. Große Variablen dürfen jedoch eine maximale Feld-/Datensatzlänge von 32766 Bytes nicht überschreiten.

Arbeitsdateityp	Verarbeitung
	<p>Beim Arbeitsdateityp PORTABLE wird die Feldinformation in der Arbeitsdatei gespeichert. Während des READ wird die Größe einer dynamischen Variablen geändert, wenn die Feldgröße im Datensatz von der aktuellen Größe abweicht.</p> <p>Mit dem WRITE WORK FILE-Statement werden Felder mit ihrer Byte-Länge in die angegebene Datei geschrieben. Alle Datentypen (DYNAMIC oder nicht) werden gleich behandelt. Es werden keine strukturellen Informationen eingefügt. Natural verwendet einen Puffermechanismus. Daher sind die Daten erst nach einem CLOSE WORK vollständig geschrieben. Das ist dann besonders wichtig, wenn die Datei mit einem anderen Utility verarbeitet werden soll während Natural aktiv ist.</p> <p>Mit dem READ WORK FILE-Statement werden Felder mit fester Länge in ihrer ganzen Länge gelesen. Wenn das Ende der Datei erreicht wird, wird der Rest des aktuellen Feldes mit Leerzeichen aufgefüllt. Die nachfolgenden Felder werden nicht verändert. Bei DYNAMIC-Datentypen wird der ganze Rest der Datei gelesen, außer wenn sie länger als 1073741824 Bytes ist. Wenn das Ende der Datei erreicht wird, bleiben die restlichen Felder (Variablen) unverändert (normales Natural-Verhalten).</p>
CSV	Die maximale Feld-/Datensatzlänge für dynamische und große Variablen ist 32766 Bytes. Dynamische Variables werden unterstützt. X-Arrays sind nicht erlaubt und resultieren in einer Fehlermeldung.

Beispiel

```

** Example 'WWFEX1': WRITE WORK FILE
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'LONDON'
  WRITE WORK FILE 1
    PERSONNEL-ID NAME
END-FIND
*
END

```


136 SQL Statements

Neben den „eigentlichen“ Natural-Statements, d.h., den Statements der Data Manipulation Language (DML), bietet Natural SQL-Statements, so dass Sie in Natural-Programmen SQL direkt benutzen können.

Folgende SQL-Statements sind verfügbar:

`CALLDBPROC` | `COMMIT` | `DELETE` | `INSERT` | `PROCESS SQL` | `READ RESULT SET` | `ROLLBACK`
| `SELECT` | `UPDATE`



Anmerkung: Bezüglich der Portabilität von Natural-Anwendungen beachten Sie bitte, dass die Natural-SQL-Statements nur für Datenbanksysteme verwendet werden können, die SQL unterstützen, wohingegen Natural-DML-Statements wie `FIND` und `READ` für alle von Natural unterstützten Datenbanksysteme verwendet werden können.

Dieser Teil behandelt folgende Themen:

- **Common Set und Extended Set**
- **Grundlegende Syntaxbestandteile**
- **Das Natural-View-Konzept**
- **Skalar-Ausdrücke**
- **Suchbedingungen**
- **Select Expressions**
- **Flexible SQL**
- SQL-Statements in alphabetischer Reihenfolge

Es gibt eine weitere Möglichkeit, SQL-Statements einzusetzen: die sogenannte „flexible SQL“, mit der Sie die Möglichkeit haben, beliebige SQL-Syntax zu verwenden.

Die in Natural verfügbaren SQL-Statements umfassen zwei Syntax-Sätze auf Großrechnern:

■ **ein allgemeiner (Common Set)**

Der Common Set entspricht im Prinzip den Syntaxdefinitionen der Standard-SQL und kann für alle von Natural unterstützten SQL-fähigen Datenbanksysteme verwendet werden. Der Common Set ist bei allen SQL-Datenbank gültig.

■ **ein erweiterter (Extended Set)**

Der Extended Set bietet darüber hinaus einige spezielle Erweiterungen zur Unterstützung von bestimmten Funktionen verschiedener von Natural unterstützter Datenbanksysteme. Zur Zeit steht der Extended Set teilweise zur Verfügung und ist nur bei DB2-Datenbanken gültig.

Dieses Kapitel beschreibt in erster Linie den Common Set. Die Statement-Syntax entspricht weitestmöglich der in der betreffenden SQL-Literatur beschriebenen; Einzelheiten finden Sie in dieser Literatur.

138

Grundlegende Syntaxbestandteile

▪ Konstanten	924
▪ Namen	924
▪ Parameter	927
▪ Natural-Formate und SQL-Datentypen	931

Dieses Kapitel behandelt grundlegende Syntaxbestandteile, die dann in den Beschreibungen der einzelnen Statements nicht mehr näher erläutert werden. Diese Teile sind:

Konstanten

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Konstanten sind:

- *constant*
- *integer*

Diese Konstanten sind im Folgenden beschrieben.

<i>constant</i>	Das Element <i>constant</i> bezieht sich immer auf eine Natural-Konstante.
<i>integer</i>	Das Element <i>integer</i> bezieht sich immer auf eine Ganzzahl-Konstante.



Anmerkung: Wenn das Dezimalzeichen mit dem (Session-Parameter DC) auf Komma (,) gesetzt ist, darf unmittelbar nach einer numerischen Konstanten kein Komma angegeben werden, sondern es muss ein Leerzeichen dazwischen stehen, weil es sonst zu einem Systemfehler kommt oder falschen Ergebnissen kommen kann.

Ungültige Syntax:	Gültige Syntax:
VALUES (1,'A') leads to a syntax error	VALUES (1 , 'A')
VALUES (1,2,3) leads to wrong results	VALUES (1 , 2 , 3)

Namen

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Namen sind:

- *authorization-identifier*
- *dsm-name*
- *view-name*
- *column-name*
- *table-name*
- *correlation-name*

Diese Elemente sind im Folgenden beschrieben.

<i>authorization-identifier</i>	Ein <i>authorization-identifier</i> , der auch „creator name“ genannt wird, dient zur Qualifizierung von Datenbanktabellen und Views. Siehe auch weiter unten .
<i>dgm-name</i>	<i>dgm-name</i> ist jeweils der Name eines mit der Natural-Utility SYSDDM erzeugten Natural-DDMs.
<i>view-name</i>	<i>view-name</i> ist jeweils der Name eines im DEFINE DATA -Statement definierten Views.
<i>column-name</i>	<i>column-name</i> ist jeweils der Name einer physischen Datenbankspalte.
<i>table-name</i>	<p>Syntax:</p> <pre><i>authorization-identifier</i> <i>dgm-name</i></pre> <p>Das Element <i>table-name</i> in diesem Kapitel dient zur Referenzierung von SQL-Basistabellen und SQL-Viewed-Tabellen. Für jede Tabelle muss ein entsprechendes Natural-DDM existieren. Der Name des DDMs muss mit dem Namen der entsprechenden physischen Datenbanktabelle bzw. des Views identisch sein.</p> <p><i>authorization-identifier</i></p> <p>Es gibt zwei Arten, den <i>authorization-identifier</i> einer Datenbanktabelle bzw. eines Views anzugeben.</p> <p>Die eine Art entspricht der Standard-SQL-Syntax: <i>authorization-identifier</i> und Tabellenname werden durch einen Punkt miteinander verbunden. Hierbei muss der DDM-Name dem Namen der physischen Datenbanktabelle (ohne <i>authorization-identifier</i>) entsprechen.</p> <p>Beispiel:</p> <pre>DEFINE DATA LOCAL 01 PERS VIEW OF PERSONNEL 02 NAME 02 AGE END-DEFINE SELECT * INTO VIEW PERS FROM SQL.PERSONNEL ...</pre> <p>Die andere Möglichkeit besteht darin, den <i>authorization-identifier</i> als Teil des DDM-Namens selbst zu definieren. Der DDM-Name besteht dann aus dem <i>authorization-identifier</i> gefolgt von einem Bindestrich</p>

	<p>(-) und gefolgt vom Namen der Datenbanktabelle. Intern wird der Bindestrich zwischen <i>authorization-identifizier</i> und Tabellennamen in einen Punkt umgesetzt.</p> <p>Anmerkung: Diese Form des DDM-Namens kann auch in einem FIND- oder READ-Statement verwendet werden, da sie den für diese Statements geltenden DDM-Namenskonventionen entspricht.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 01 PERS VIEW OF SQL-PERSONNEL 02 NAME 02 AGE END-DEFINE SELECT * INTO VIEW PERS FROM SQL-PERSONNEL ... </pre> <p>Wenn der <i>authorization-identifizier</i> weder explizit noch als Teil des DDM-Namens angegeben wird, wird er vom betreffenden SQL-Datenbanksystem bestimmt.</p> <p><i>Table-names</i> können nicht nur in SELECT-Statements verwendet werden, sondern auch in den Statements DELETE, INSERT und UPDATE.</p> <p>Beispiele:</p> <pre> ... DELETE FROM SQL.PERSONNEL WHERE AGE IS NULL INSERT INTO SQL.PERSONNEL (NAME,AGE) VALUES ('ADKINSON',35) UPDATE SQL.PERSONNEL SET SALARY = SALARY * 1.1 WHERE AGE > 30 ... </pre>
<i>correlation-name</i>	<i>correlation-name</i> ist ein Alias-Name für einen <i>table-name</i> . Er kann zur Qualifizierung von Spaltennamen verwendet werden. Außerdem dient

	<p>er dazu, implizit Felder in einem Natural-View zu qualifizieren, der in der INTO-Klausel eines SELECT-Statements verwendet wird.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 01 PERS-NAME (A20) 01 EMPL-NAME (A20) 01 AGE (I2) END-DEFINE ... SELECT X.NAME , Y.NAME , X.AGE INTO PERS-NAME , EMPL-NAME , AGE FROM SQL-PERSONNEL X , SQL-EMPLOYEES Y WHERE X.AGE = Y.AGE END-SELECT ... </pre> <p>Die Verwendung von <i>correlation-names</i> ist zwar in der Regel nicht nötig, kann aber helfen, die Lesbarkeit eines Statements zu erleichtern.</p>
--	---

Parameter

parameter

`[:] host-variable [INDICATOR [:] host-variable] [LINDICATOR [:] host-variable]`

Im Folgenden sind die Syntaxelemente beschrieben.

<i>host-variable</i>	<p>Eine <i>host-variable</i> ist eine in einem SQL-Statement referenzierte Natural-Programmvariable (keine Systemvariable), die entweder ein eigenständiges Feld oder Teil eines Views sein kann.</p> <p>Wenn sie als empfangendes Feld (z.B. in einer INTO-Klausel) definiert wird, ist die <i>host-variable</i> ein Feld, das vom Datenbanksystem einen Wert erhält.</p> <p>Wenn sie als sendendes Feld (z.B. in einer WHERE-Klausel) definiert wird, ist die <i>host-variable</i> ein Feld, dessen Wert vom Programm an das Datenbanksystem übergeben wird.</p> <p>Siehe auch <i>Natural-Formate und SQL-Datentypen</i>.</p>
[:]	Doppelpunkt:

	<p>Gemäß den SQL-Standards kann einer <i>host-variable</i> ein Doppelpunkt (:) vorangestellt werden. Bei der Verwendung mit flexibler SQL muss ihr ein Doppelpunkt vorangestellt werden.</p> <p>Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 5px;">SELECT NAME INTO :#NAME FROM PERSONNEL WHERE AGE = :VALUE</pre> <p>Wenn ein Variablenname mit einem für SQL reservierten Wort identisch ist, ist der Doppelpunkt ebenfalls erforderlich. In Situationen, in denen entweder eine <i>host-variable</i> oder eine Spalte referenziert werden kann, wird ein Name ohne Doppelpunkt als Spaltenname interpretiert.</p>
<p>INDICATOR</p>	<p>INDICATOR-Klausel</p> <p>Diese Klausel ist optional und dient dazu, herauszufinden, ob eine zu lesende Spalte „Null“ ist, d.h. keinen Wert enthält, oder tatsächlich den Wert Null (0) bzw. Leerzeichen enthält.</p> <p>Wenn sie mit einer empfangenden <i>host-variable</i> (Zielfeld) verwendet wird, dient die <i>INDICATOR host-variable</i> (Null-Indikatorfeld) dazu, herauszufinden, ob eine zu lesende Spalte „Null“ ist.</p> <p>Beispiel:</p> <pre style="background-color: #f0f0f0; padding: 5px;">DEFINE DATA LOCAL 1 NAME (A20) 1 NAMEIND (I2) END-DEFINE SELECT * INTO NAME INDICATOR NAMEIND ... </pre> <p>In diesem Beispiel ist NAME die empfangende <i>host-variable</i> und NAMEIND das Null-Indikatorfeld.</p> <p>Ist ein Null-Indikatorfeld angegeben und die gelesene Spalte ist „Null“, wird das Indikatorfeld auf einen negativen Wert und das Zielfeld je nach Datentyp auf Null (0) bzw. Leerzeichen gesetzt. Andernfalls ist der Wert des Null-Indikatorfeldes größer gleich Null (0).</p> <p>Wenn sie mit einer sendenden <i>host-variable</i> (Ausgangsfeld) verwendet wird, dient die <i>INDICATOR host-variable</i> dazu, dem Ausgangsfeld einen Nullwert zuzuweisen.</p>

	<p>Beispiel:</p> <pre> DEFINE DATA LOCAL 1 NAME (A20) 1 NAMEIND (I2) UPDATE ... SET NAME = :NAME INDICATOR :NAMEIND WHERE ... </pre> <p>In diesem Beispiel ist :NAME die sendende <i>host-variable</i> und :NAMEIND das Null-Indikatorfeld. Durch Eingabe eines negativen Wertes in das Null-Indikatorfeld wird der Datenbankspalte ein Nullwert zugewiesen.</p> <p>Eine INDICATOR <i>host-variable</i> hat Format/Länge I2.</p>
<p>LINDICATOR</p>	<p>LINDICATOR-Klausel:</p> <p>Diese Klausel ist optional und dient zur Unterstützung von Spalten des Typs VARCHAR oder LONG VARCHAR.</p> <p>Wenn sie mit einer empfangenden <i>host-variable</i> (Zielfeld) verwendet wird, enthält die LINDICATOR <i>host-variable</i> (Längen-Indikatorfeld) die Anzahl der tatsächlich von der Datenbank in das Zielfeld geschriebenen Zeichen. Das Zielfeld wird immer mit Leerzeichen aufgefüllt.</p> <p>Enthält die VARCHAR- bzw. LONG VARCHAR-Spalte mehr Zeichen als in das Zielfeld passen, wird im Längen-Indikatorfeld die Anzahl der tatsächlich gelesenen Zeichen ausgegeben und im Null-Indikatorfeld (falls angegeben) die tatsächliche Gesamtlänge der Spalte.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 1 ADDRESSLIND (I2) 1 ADDRESS (A50/1:6) END-DEFINE SELECT * INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND ... </pre> <p>:ADDRESS(*) erhält die ersten 300 Bytes (falls vorhanden) der betreffenden VARCHAR- bzw. LONG VARCHAR-Spalte, und :ADDRESSLIND ist das Längen-Indikatorfeld, das die Anzahl der tatsächlich von der Datenbank gelesenen Zeichen enthält.</p> <p>Wenn sie mit einer sendenden <i>host-variable</i> (Ausgangsfeld) verwendet wird, gibt das Längen-Indikatorfeld an, wieviele Zeichen des Ausgangsfeldes an die Datenbank übergeben werden sollen.</p>

Beispiel:

```

DEFINE DATA LOCAL
1 NAMELIND (I2)
1 NAME      (A20)
1 AGE       (I2)
END-DEFINE
MOVE 4      TO NAMELIND
MOVE 'ABC%' TO NAME
SELECT AGE
  INTO :AGE
WHERE NAME LIKE :NAME LINDICATOR :NAMELIND
...
    
```

Eine LINDICATOR *host-variable* hat Format/Länge I2 oder I4. Um Verarbeitungszeit zu sparen, sollte sie unmittelbar vor dem betreffenden Ausgangs- bzw. Zielfeld angegeben werden; andernfalls würde sie zur Laufzeit in einen Zwischenspeicher kopiert.

Wenn das LINDICATOR-Feld als I2-Feld definiert ist, wird der SQL-Datentyp VARCHAR zum Senden/Erhalten der betreffenden Spalte verwendet. Wird die LINDICATOR *host-variable* als I4 angegeben, wird ein großer Objektdatentyp (CLOB/BLOB) verwendet.

Wenn das Feld als DYNAMIC (dynamisch) definiert wird, wird die Spalte in einer internen Schleife bis zu ihrer wirklichen Länge gelesen. Das LINDICATOR-Feld und *LENGTH werden auf diese Länge gesetzt. Bei Feldern fester Länge wird die Spalte bis zur definierten Länge gelesen. In beiden Fällen wird das Feld bis zum im LINDICATOR-Feld definierten Wert geschrieben.

Ein Feld fester Länge soll zum Beispiel mit einem als I2 angegebenen LINDICATOR-Feld definiert werden. Wenn die VARCHAR-Spalte mehr Zeichen enthält als in dieses Feld fester Länge passen, wird das Längenindikatorfeld auf die tatsächlich zurückgegebene Länge gesetzt, und das Nullindikatorfeld (falls angegeben) wird auf die Gesamtlänge dieser Spalte (Lesen) gesetzt. Dies ist bei Feldern fester Länge ≥ 32 KB nicht möglich (die Länge ist größer gewählt als die Länge des Nullindikatorfeldes).

Natural-Formate und SQL-Datentypen

Das Natural-Format einer *host-variable* wird entsprechend der folgenden Tabelle in einen SQL-Datentyp umgesetzt:

Natural-Format/Länge	SQL-Datentyp
A_n	CHAR (n)
B2	SMALLINT
B4	INT
B_n ; n ungleich 2 oder 4	CHAR (n)
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
$N_{nn.m}$	NUMERIC ($nn+m, m$)
$P_{nn.m}$	NUMERIC ($nn+m, m$)
T	TIME
D	DATE
G_n ; nur für Views	GRAPHIC (n)

Natural überprüft nicht, ob der SQL-Datentyp mit der Datenbankspalte kompatibel ist. Außer bei Feldern mit Format N wird keine Datenkonvertierung vorgenommen.

Bei Natural SQL gibt es zu den Standard-Natural-Formaten noch folgende Erweiterungen:

- Um alphanumerische Spalten zu unterstützen, die länger als 253 Bytes sind, kann ein eindimensionales Array vom Format A verwendet werden. Der Index dieses Arrays muss mit 1 anfangen und kann nur mit (*) referenziert werden. Der entsprechende SQL-Datentyp ist CHAR (n), wobei n die Gesamtanzahl der Bytes des Arrays ist.
- Um Spalten mit variabler Länge zu unterstützen, kann eine *host-variable* mit Schlüsselwort LINDICATOR verwendet werden. Der entsprechende SQL-Datentyp ist VARCHAR (n); vgl. [LINDICATOR-Klausel](#).
- Die Natural-Formate Datum (D) und Zeit (T) können mit Entire Access verwendet werden und werden in die entsprechenden datenbank-spezifischen Formate umgesetzt (Näheres siehe *Entire Access*-Dokumentation)

Ein sendendes Feld, das als eindimensionales Array ohne LINDICATOR-Feld angegeben wird, wird in den SQL-Datentyp VARCHAR umgesetzt. Seine Länge ist die Gesamtanzahl der Bytes des Arrays ohne Berücksichtigung nachgestellter Leerzeichen.

139

Das Natural-View-Konzept

Einige Natural-SQL-Statements erlauben auch die Verwendung von Natural-Views.

Ein Natural-View kann anstelle einer Parameterliste angegeben werden, wobei jedes Feld des Views — außer Gruppen, redefinierten Feldern sowie Feldern mit vorangestelltem Präfix L@ oder N@ — einem Parameter (*host-variable*) entspricht.

Felder, deren Namen mit L@ bzw. N@ anfangen, können nur zusammen mit entsprechenden Feldern gleichen Namens verwendet werden. Dabei werden:

- L@-Felder umgesetzt in LINDICATOR-Felder,
- N@-Felder umgesetzt in INDICATOR-Felder.

Ein L@-Feld sollte im View jeweils unmittelbar vor dem Feld, auf das es sich bezieht, definiert werden.

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 PERSID      (I4)
  02 NAME       (A20)
  02 N@NAME     (I2)           /* null indicator of NAME
  02 L@ADDRESS  (I2)           /* length indicator of ADDRESS
  02 ADDRESS    (A50/1:6)
  02 N@ADDRESS  (I2)           /* null indicator of ADDRESS
01 #PERSID     (I4)
END-DEFINE
...
SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
...
END-SELECT
```

Das obige Beispiel entspricht dem Folgenden:

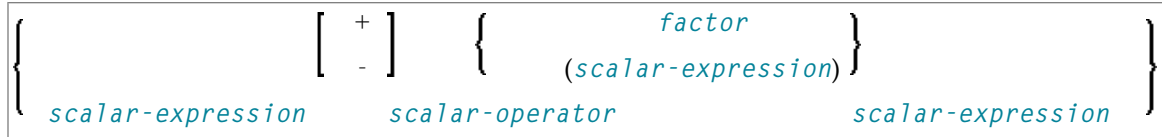
```
...
SELECT *
  INTO PERSID,
       NAME INDICATOR N@NAME,
       ADDRESS(*)INDICATOR N@ADDRESS LINDICATOR L@ADDRESS
  FROM SQL-PERSONNEL
  WHERE PERSID = #PERSID
...
END-SELECT
```



Anmerkung: Wenn mit Natural für Windows, Natural für UNIX oder Natural für OpenVMS auf *varchar*-Datentypen zugegriffen wird, muss in dem View eine entsprechende Längenindikatorvariable vorhanden sein.

140 Skalar-Ausdrücke

▪ scalar-expression	936
▪ scalar-operator	936
▪ factor	937



Dieses Kapitel behandelt folgende Themen:

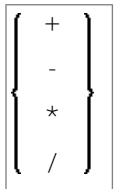
scalar-expression

Eine *scalar-expression* besteht aus einem *factor* und anderen *scalar-expressions* einschließlich *scalar-operators*.

In punkto Referenzierungspriorität gilt folgendes:

- Wenn in einer *scalar-expression* ein unqualifizierter Variablenname angegeben wird, wird zunächst angenommen, dass es sich um den Namen einer Spalte der referenzierten Tabelle handelt.
- Falls in der Tabelle eine Spalte dieses Namens nicht vorkommt, behandelt Natural die Variable als Benutzervariable (*host-variable*).

scalar-operator



Ein *scalar-operator* kann einer der oben aufgeführten Operatoren sein, wobei vor und nach den Operatoren – und / jeweils mindestens ein Leerzeichen stehen muss.

factor

```

{
  atom
  column-reference
  aggregate-function
  special-register
}

```

Ein *factor* kann eines der obigen Elemente sein, die im Folgenden beschrieben sind.

atom

```

{
  parameter
  constant
}

```

Ein *atom* kann entweder ein *parameter* oder eine Konstante (*constant*) sein; vgl. Abschnitt [Grundlegende Syntaxbestandteile](#).


column-reference

```

[
  table-name.
  correlation-name.
] column-name

```

Eine *column-reference* ist ein Spaltenname (*column-name*), optional qualifiziert durch einen Tabellennamen (*table-name*) oder einen *correlation-name* (vgl. Abschnitt [Grundlegende Syntaxbestandteile](#)). Qualifizierte Namen sind oft klarer als unqualifizierte und manchmal erforderlich.

 **Anmerkung:** Ein *table-name* darf hier nicht explizit mit einem *authorization-identifier* qualifiziert werden. Falls Sie einen qualifizierten *table-name* benötigen, verwenden Sie stattdessen einen *correlation-name*.

Wird eine Spalte mit einem *table-name* oder *correlation-name* referenziert, muss sie in der betreffenden Tabelle enthalten sein. Wird weder *table-name* noch *correlation-name* angegeben, muss die betreffende Spalte in einer der in der FROM-Klausel angegebenen Tabellen enthalten sein (siehe [table-expression](#)).

aggregate-function

COUNT	{ (*) (DISTINCT <i>column-reference</i>) }
AVG	{ (DISTINCT <i>column-reference</i>) ([ALL] <i>scalar-expression</i>) }
MAX	
MIN	
SUM	

SQL bietet eine Reihe spezieller Funktionen zur Erweiterung der grundlegenden Such-Möglichkeiten. Folgende sogenannte SQL *aggregate-functions* sind verfügbar und werden von Natural unterstützt:

AVG	gibt den Durchschnitt der Werte einer Spalte zurück.
COUNT	gibt die Anzahl der Werte einer Spalte zurück.
MAX	gibt den größten Wert einer Spalte zurück.
MIN	gibt den kleinsten Wert einer Spalte zurück.
SUM	gibt die Summe der Werte einer Spalte zurück.

Bis auf COUNT(*) sammelt jede dieser Funktionen die Skalarwerte in einem Argument, d.h. einer einzelnen Spalte oder einer *scalar-expression*, und gibt als Ergebnis einen Skalarwert zurück.

Beispiel:

```
DEFINE DATA LOCAL
1 AVGAGE (I2)
END-DEFINE
...
SELECT AVG (AGE)
INTO AVGAGE
FROM SQL-PERSONNEL
...
```

Im allgemeinen kann dem Argument *optional* das Schlüsselwort DISTINCT vorangestellt werden, um doppelte Werte zu eliminieren, bevor die Funktion ausgeführt wird.

Wenn Sie DISTINCT angeben, muss das Argument der Name einer einzelnen Spalte sein; wenn Sie DISTINCT nicht angeben, kann das Argument eine allgemeine *scalar-expression* sein.

DISTINCT ist nicht erlaubt mit der Funktion COUNT(*), mit der alle Reihen in einer Tabelle — ohne Eliminierung doppelt vorkommender Reihen — gezählt werden.

special-register

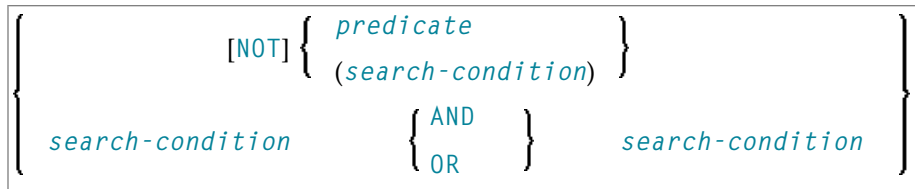
USER

Bei der Referenzierung eines *special-registers* erhält man einen Skalarwert.

141

Suchbedingungen

- search-condition 942
- predicate 942



Dieses Kapitel behandelt folgende Themen:

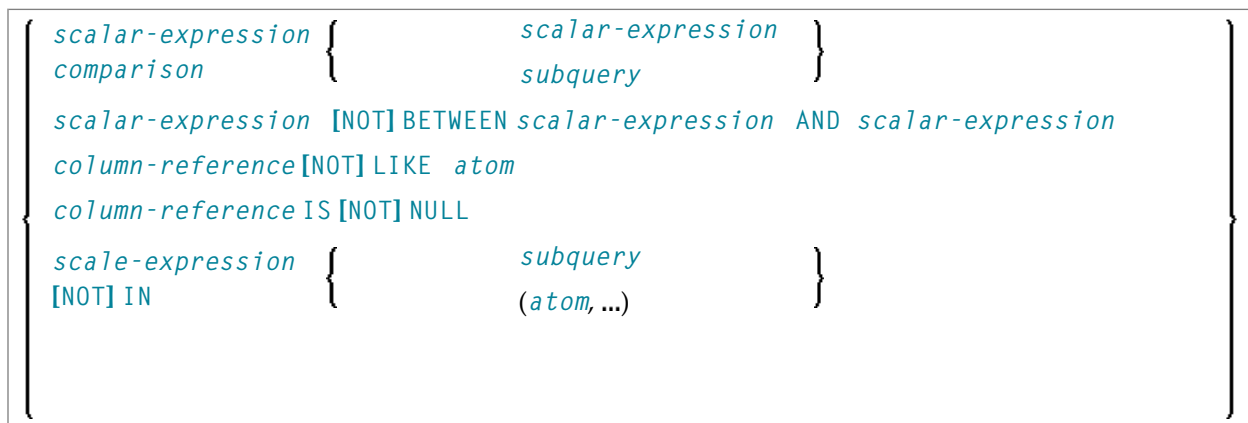
search-condition

Eine *search-condition* kann aus einer einfachen Bedingung (*predicate*) bestehen oder aus mehreren *search-conditions*, die durch die Boole'schen Operatoren AND, OR oder NOT verknüpft werden, wobei die Reihenfolge der Auswertung außerdem durch entsprechende Klammerung bestimmt werden kann.

Beispiel:

```
DEFINE DATA LOCAL
01 NAME    (A20)
01 AGE     (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32 AND NAME > 'K'
END-SELECT
...
```

predicate



<i>scalar-expression</i> <i>comparison</i> <i>EXISTS subquery</i>	{ ALL ANY SOME }	<i>subquery</i>
---	------------------------------	-----------------

Das *predicate* gibt eine Bedingung an, die wahr, falsch oder unbekannt sein kann.

In einer *search-condition* kann ein *predicate* aus einer einfachen oder komplexen Vergleichsoperation oder einer anderen Art von Bedingung bestehen.

Beispiel:

```
SELECT NAME, AGE
INTO VIEW PERS
FROM SQL-PERSONNEL
WHERE AGE BETWEEN 20 AND 30
      OR AGE IN ( 32, 34, 36 )
      AND NAME LIKE '%er'
...

```



Anmerkung: Das Prozentzeichen (%) kann zu Konflikten mit Natural-Terminalkommandos führen. In diesem Fall definieren Sie als Steuerzeichen für Terminalkommandos ein anderes Zeichen als % (siehe Session-Parameter CF).

Die einzelnen *predicates* sind auf den folgenden Seiten beschrieben (weitere Informationen zu *predicates* finden Sie in der betreffenden Literatur). Entsprechend der obigen Syntax heißen sie wie folgt:

- Comparison Predicate
- BETWEEN Predicate
- LIKE Predicate
- NULL Predicate
- IN Predicate
- Quantified Predicate

- EXISTS Predicate

Comparison Predicate

$$\text{scalar-expression comparison} \left\{ \begin{array}{l} \text{scalar-expression} \\ \text{subquery} \end{array} \right\}$$

Ein Comparison Predicate vergleicht zwei Werte.

Siehe Informationen zu *scalar-expression*.

comparison

$$\left\{ \begin{array}{l} = \\ < \\ > \\ \leq \\ \geq \\ < > \end{array} \right\}$$

comparison kann einer der folgenden Operatoren sein:

=	gleich
<	kleiner als
>	größer als
<=	kleiner gleich
>=	größer gleich
<>	ungleich

subquery

$$(select-expression)$$

Eine *subquery* ist eine *select-expression* innerhalb einer anderen *select-expression*.

Beispiel:

```

DEFINE DATA LOCAL
1 #NAME      (A20)
1 #PERSNR   (I4)
END-DEFINE
...
SELECT NAME, PERSNR
  INTO #NAME, #PERSNR
  FROM SQL-PERSONNEL
  WHERE PERSNR IN
    ( SELECT PERSNR
      FROM SQL-AUTOMOBILES
      WHERE COLOR = 'black' )
...
END-SELECT

```

Siehe *Select Expressions*.

BETWEEN Predicate

```
scalar-expression [NOT] BETWEEN scalar-expression AND scalar-expression
```

Ein BETWEEN Predicate vergleicht einen Wert mit einem Bereich von Werten.

Siehe *scalar-expression*.

LIKE Predicate

```
column-reference [NOT] LIKE atom
```

Ein LIKE Predicate sucht nach Zeichenketten, die ein bestimmtes Muster haben.

Siehe *column-reference* und *atom*.

NULL Predicate

```
column-reference IS [NOT] NULL
```

Ein NULL Predicate prüft auf Nullwerte.

Siehe *column-reference*.

IN Predicate

```
scalar-expression [NOT] IN { subquery ...  
                             (atom) }
```

Ein IN Predicate vergleicht einen Wert mit einer Sammlung von Werten.

Siehe *scalar-expression* und *atom*.

Siehe *subquery*.

Quantified Predicate

```
scalar-expression comparison { ALL  
                                ANY } subquery  
                                SOME
```

Ein Quantified Predicate vergleicht einen Wert mit einer Sammlung von Werten.

Siehe *scalar-expression*, *comparison* und *subquery*.

EXISTS Predicate

```
EXISTS subquery
```

Ein EXISTS Predicate prüft, ob bestimmte Reihen vorhanden sind.

Die Bedingung des EXISTS Predicate kann nur erfüllt werden, wenn die ausgewertete *subquery* tatsächlich ein Ergebnis liefert, d.h. wenn mindestens eine Reihe in der FROM-Tabelle der *subquery* die WHERE-Bedingung dieser *subquery* erfüllt.

Beispiel für EXISTS:

```
DEFINE DATA LOCAL
1 #NAME (A20)
END-DEFINE
...
SELECT NAME
  INTO #NAME
  FROM SQL-PERSONNEL
  WHERE EXISTS
    ( SELECT *
      FROM SQL-EMPLOYEES
      WHERE PERSNR > 1000
        AND NAME < 'L' )
```

```
...  
END-SELECT  
...
```

Siehe [subquery](#).

142

Select Expressions

- selection 950
- table-expression 951

```
SELECT selection table-expression
```

Eine *select-expression* gibt eine Ergebnistabelle an. Sie wird bei den folgenden Statements benutzt: **INSERT** | **SELECT**

Dieses Kapitel behandelt folgende Themen:

selection

```
[ ALL  
  DISTINCT ] { { scalar-expression [[AS] correlation-name], ... }  
              * }
```

In der *selection* geben Sie an, was ausgewählt werden soll.

ALL/DISTINCT

Doppelt vorkommende Reihen werden nicht automatisch aus dem Ergebnis einer *select-expression* entfernt. Wenn Sie dies wünschen, geben Sie das Schlüsselwort **DISTINCT** an.

Die Alternative zu **DISTINCT** ist **ALL**. Wenn Sie nichts angeben, gilt **ALL**.

scalar-expression

Anstelle von oder zusätzlich zu einfachen Spaltennamen können Sie auch allgemeine *scalar-expressions* angeben, die Skalar-Operatoren und Skalar-Funktionen, die berechnete Werte liefern, enthalten. Siehe [Scalar Expressions](#).

Beispiel:

```
SELECT NAME, 65 - AGE  
FROM SQL-PERSONNEL  
...
```

correlation-name

Es besteht die Möglichkeit, einer *scalar-expression* einen *correlation-name* als Alias-Namen für eine Ergebnisspalte zuzuweisen.

Der *correlation-name* braucht nicht eindeutig sein. Wenn für eine Ergebnisspalte kein *correlation-name* angegeben wird, wird der betreffende *column-name* genommen (falls sich die Ergebnisspalte von einem Spaltennamen ableitet; andernfalls hat die Ergebnisspalte keinen Namen). Der Name einer Ergebnisspalte kann beispielsweise als Spaltenname in der ORDER BY-Klausel eines SELECT-Statements angegeben werden.

Stern-Notation (*)

Alle Spalten aller in der FROM-Klausel angegebenen Tabellen werden ausgewählt.

Beispiel:

```
SELECT *
  FROM SQL-PERSONNEL, SQL-AUTOMOBILES
  ...
```

table-expression

```
FROM table-reference,...
  [WHERE search-condition]
  [GROUP BY column-reference,...]
  [HAVING search-condition]
```

Die *table-expression* gibt an, von wo und nach welchen Kriterien Reihen gelesen werden sollen.

table-reference

```
{
  table-name [[AS] correlation-name]
  subquery [AS] correlation-name
  joined-table
}
```

In der FROM-Klausel geben Sie eine oder mehrere Tabellen an, die die in der *selection-list* verwendeten Spaltenfelder enthalten müssen.

Es besteht die Möglichkeit, einem *table-name* eine *correlation-clause* zuzuweisen.

Ein `FINAL TABLE`-Schlüsselwort, gefolgt von einem `INSERT`-Statement in Klammern gehört zum SQL Extended Set und gibt an, dass die eingefügten Reihen für das betreffende `SELECT`-Statement zurückgegeben werden. Die Ergebnistabelle beinhaltet alle Reihen, die eingefügt wurden. Alle Spalten der eingefügten Tabelle können in der `select list` referenziert werden. Wenn das `INSERT`-Statement in der `table-reference` benutzt wird, kann das `subselect` noch die `WHERE`-Klausel, `GROUP BY`-Klausel, `HAVING`-Klausel und `aggregate-functions` angeben.

correlation-clause

```
[AS] correlation-name [(column-name, ...)]
```

Eine `correlation-clause` besteht aus `KEYWORD AS` als Option und einem `correlation-name`, und es folgt ihr als Option eine einfache `column-name`-Liste. Die `column-name`-Liste gehört zum SQL Extended Set.

joined-table

```
table-reference [ { INNER | LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] } ] JOIN table-reference ON join-condition
```

Eine `joined-table` gibt eine Zwischenergebnistabelle an, die das Ergebnis einer Join-Operation ist.

Der Join kann ein `INNER`, `LEFT OUTER`, `RIGHT OUTER` oder `FULL OUTER JOIN` sein. Falls Sie nichts angeben, gilt `INNER`.

Es ist möglich, mehrere Joins zu schachteln, d.h. die Tabellen, die die Zwischenergebnistabelle bilden, können ihrerseits Zwischenergebnistabellen einer Join-Operation oder einer `subquery` sein, wobei letztere wiederum ebenfalls eine `joined-table` oder eine weitere `subquery` in der `FROM`-Klausel haben kann.

join-condition

Bei INNER, LEFT OUTER **und** RIGHT OUTER Joins:

```
search-condition
```

Bei FULL OUTER Joins:

```
full-join-expression = full-join-expression [AND ...]
```

full-join-expression

```
{ column-name
  { VALUE
    COALESCE } (column-name , ...) }
```

In einer *join-expression* sind nur *column-names* und die *scalar-function* VALUE (bzw. ihr Synonym COALESCE) erlaubt. Siehe *column-name*.

WHERE-Klausel

```
[WHERE search-condition]
```

In der WHERE-Klausel geben Sie eine Suchbedingung (*search-condition*) an, nach der die Reihen gelesen werden sollen.

Beispiel:

```
DEFINE DATA LOCAL
01 NAME   (A20)
01 AGE    (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32
END-SELECT
...
```

Siehe *search-condition*.

GROUP BY-Klausel

[*GROUP BY column-reference*,...]

Die `GROUP BY`-Klausel sortiert die in der `FROM`-Klausel angegebene Tabelle nach Gruppen, und zwar so, dass alle Reihen einer Gruppe in der `GROUP BY`-Spalte den gleichen Wert haben. Jede *column-reference* in der Selektionsliste muss entweder eine `GROUP BY`-Spalte sein oder mit einer *aggregate-function* angegeben werden. *Aggregate-functions* werden auf einzelne Gruppen (nicht auf die ganze Tabelle) angewandt. Die Ergebnistabelle enthält soviele Reihen wie Gruppen.

Siehe *column-reference* und *aggregate-function*.

Beispiel:

```
DEFINE DATA LOCAL
1 #AGE      (I2)
1 #NUMBER  (I2)
END-DEFINE
...
SELECT AGE , COUNT(*)
  INTO #AGE, #NUMBER
  FROM SQL-PERSONNEL
  GROUP BY AGE
...
```

Steht vor der `GROUP BY`-Klausel eine `WHERE`-Klausel, werden vor dem Aussortieren nur diejenigen Reihen von der `GROUP BY`-Klausel erfasst, die die `WHERE`-Bedingung erfüllen.

HAVING-Klausel

[*HAVING search-condition*]

Wenn Sie eine `HAVING`-Klausel verwenden, sollten Sie auch eine `GROUP BY`-Klausel verwenden. Genau wie die `WHERE`-Klausel Reihen aus einer Ergebnistabelle aussortiert, sortiert die `HAVING`-Klausel Gruppen aus, und zwar auf Grundlage einer Suchbedingung (*search-condition*). *Scalar-expressions* in einer `HAVING`-Klausel dürfen pro Gruppe nur einen Wert enthalten.

Siehe *scalar-expression* und *search-condition*.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #AVGAGE    (I2)
1 #NUMBER    (I2)
END-DEFINE
...
SELECT NAME, AVG(AGE), COUNT(*)
  INTO #NAME, #AVGAGE, #NUMBER
  FROM SQL-PERSONNEL
  GROUP BY NAME
  HAVING COUNT(*) > 1
...
```


143 Flexible SQL

▪ Flexible SQL benutzen	958
▪ Textvariablen in Flexible SQL angeben	959

Dieses Kapitel behandelt folgende Themen:

Flexible SQL benutzen

Zusätzlich zu der im bisherigen Verlauf dieses Kapitels beschriebenen SQL-Syntax haben Sie mit flexibler SQL die Möglichkeit, beliebige SQL-Syntax zu verwenden.

Die Zeichen << und >>

Flexible SQL muss zwischen den Zeichen << und >> stehen. Sie kann beliebigen SQL-Text und *host-variables* enthalten. Mit flexibler SQL verwendete *host-variables* müssen als Präfix einen Doppelpunkt (:) haben.

Flexible SQL kann über mehrere Zeilen gehen und kann ganze oder teilweise Kommentarzeilen enthalten (vgl. [PROCESS SQL-Statement](#)).

Flexible SQL kann anstelle folgender SQL-Syntaxteile verwendet werden:

- *atom*
- *column-reference*
- *scalar-expression*
- *predicate*

Flexible SQL kann auch zwischen den Klauseln einer *select-expression* verwendet werden:

```
SELECT selection
  << ... >>
  INTO ...
  FROM ...
  << ... >>
  WHERE ...
  << ... >>
  GROUP BY ...
  << ... >>
  HAVING ...
  << ... >>
  ORDER BY ...
  << ... >>
```



Anmerkung: Der in flexibler SQL angegebene SQL-Text wird nicht vom Natural-Compiler verarbeitet, sondern (mit ausgetauschten *host-variables*) einfach in die SQL-Zeichenkette kopiert, die an das Datenbanksystem übergeben wird. Demzufolge werden Syntaxfehler in der flexiblen SQL erst zur Laufzeit erkannt, wenn die Datenbank das betreffende Statement ausführt.

Beispiel 1:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) >> = << MONTH (CURRENT_DATE) >>
```

Beispiel 2:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE << MONTH (BIRTH) = MONTH (CURRENT_DATE) >>
```

Beispiel 3:

```
SELECT NAME
FROM SQL-EMPLOYEES
WHERE SALARY > 50000
<< INTERSECT
  SELECT NAME
  FROM SQL-EMPLOYEES
  WHERE DEPT = 'DEPT10'
>>
```

Textvariablen in Flexible SQL angeben

Innerhalb der flexiblen SQL können Sie auch sogenannte Textvariablen angeben.

```
<<:T:host-variable [LINDICATOR:host-variable]>>
```

Die Syntax-Elemente sind im Folgenden beschrieben.

:T:	<p>Eine Textvariable ist eine <i>host-variable</i> mit dem Präfix <code>:T:</code>. Sie muss alphanumerisches Format haben.</p> <p>Zur Laufzeit wird eine Textvariable innerhalb eines SQL-Statements durch ihren Inhalt ersetzt, d.h. die in der Textvariablen enthaltene Textzeichenkette wird in die SQL-Zeichenkette eingefügt.</p> <p>Nach dem Ersetzen werden nachfolgende Leerzeichen aus der eingefügten Textzeichenkette entfernt.</p>
------------	---

	<p>Sie müssen selbst darauf achten, dass sich aus dem Inhalt einer Textvariablen beim Einfügen ein syntaktisch korrektes SQL-Statement ergibt. Insbesondere darf eine Textvariable keine <i>host-variables</i> enthalten.</p> <p>Ein Statement, das eine Textvariable enthält, wird immer im dynamischen SQL-Modus ausgeführt.</p>
LINDICATOR	<p>LINDICATOR-Option:</p> <p>Nach der Textvariablen können Sie das Schlüsselwort LINDICATOR sowie eine Längenindikator-Variable (d.h. eine <i>host-variable</i> mit vorangestelltem Doppelpunkt) angeben.</p> <p>Die Längenindikator-Variable muss Format/Länge I2 haben.</p> <p>Wenn Sie keine LINDICATOR-Variable angeben, wird der gesamte Inhalt der Textvariablen in die SQL-Zeichenkette eingefügt.</p> <p>Wenn Sie eine LINDICATOR-Variable angeben, werden nur die ersten <i>n</i> Zeichen (wobei <i>n</i> der Wert der LINDICATOR-Variablen ist) des Textvariableninhalts in die SQL-Zeichenkette eingefügt. Falls die Zahl in der LINDICATOR-Variablen größer als die Länge des Textvariableninhalts ist, wird der gesamte Textvariableninhalt eingefügt. Falls die Zahl in der LINDICATOR-Variablen negativ oder Null (0) ist, wird nichts eingefügt.</p> <p>Siehe auch allgemeine Informationen zu <i>host-variable</i>.</p>

Beispiel mit Textvariable:

```

DEFINE DATA LOCAL
01 TEXTVAR (A200)
01 TABLES VIEW OF SYSIBM-SYSTABLES
    02 NAME
    02 CREATOR
END-DEFINE
*
MOVE 'WHERE NAME > ''SYS'' AND CREATOR = ''SYSIBM''' TO TEXTVAR
*
SELECT * INTO VIEW TABLES
FROM SYSIBM-SYSTABLES
<< :T:TEXTVAR >>
DISPLAY TABLES
END-SELECT
*
END

```

Das generierte SQL-Statement (wie mit dem Systemkommando LISTSQL angezeigt) sieht wie folgt aus:

```
SELECT NAME, CREATOR FROM SYSIBM.SYSTABLES:T: FOR FETCH ONLY
```

Das ausgeführte SQL-Statement sieht wie folgt aus:

```
SELECT TABNAME, CREATOR FROM SYSIBM.SYSTABLES  
WHERE TABNAME > 'SYS' AND CREATOR = 'SYSIBM'
```


144 CALLDBPROC - SQL

▪ Funktion	964
▪ Syntax-Beschreibung	965
▪ Beispiel	966

```
CALLDBPROC dbproc ddm-name
[ [USING] { parameter [ AD= { M } ] ] } ... ]
[RESULT SETS result-set...]
[GIVING sqlcode]
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Funktion

Das Statement CALLDBPROC dient dazu, eine Stored Procedure des SQL-Datenbanksystems, mit dem Natural verbunden ist, aufzurufen.

Die Stored Procedure kann entweder ein Natural-Subprogramm oder ein in einer anderen Programmiersprache geschriebenes Programm sein. Neben der Möglichkeit, Parameter zwischen dem aufrufenden Objekt und der Stored Procedure zu übergeben, unterstützt CALLDBPROC sogenannte Result Sets; mit diesen ist es möglich, größere Datenmengen von der Stored Procedure an das aufrufende Objekt zu übergeben, als dies mittels Parametern möglich wäre.

Die Result Sets sind von der Stored Procedure erzeugte temporäre Ergebnistabellen, die das aufrufende Objekt mittels eines READ RESULT SET-Statements lesen und verarbeiten können.



Anmerkung: Im Prinzip ist der Aufruf einer Stored Procedure mit dem Aufruf eines Natural-Subprogramms vergleichbar: wenn das CALLDBPROC-Statement ausgeführt wird, wird die Kontrolle an die Stored Procedure übergeben; nach Verarbeitung der Stored Procedure wird die Kontrolle wieder an das aufrufende Objekt zurückgegeben, und die Verarbeitung wird mit dem nächsten Statement nach dem CALLDBPROC-Statement fortgesetzt.

Syntax-Beschreibung

<i>dbproc</i>	<p>Aufzurufende Stored Procedure:</p> <p>Als <i>dbproc</i> geben Sie den Namen der Stored Procedure an, die aufgerufen werden soll. Der Name kann entweder als alphanumerische Variable oder als Konstante in Apostrophen (') angegeben werden.</p> <p>Der Name muss den Regeln für Stored-Procedure-Namen des Ziel-Datenbanksystems entsprechen.</p> <p>Falls die Stored Procedure ein Natural-Subprogramm ist, darf der eigentliche Procedure-Name nicht länger als 8 Stellen sein.</p>							
<i>dsm-name</i>	<p>Name eines Natural-Datendefinitionsmoduls:</p> <p>Der Name eines DDMs muss angegeben werden, um die „Adresse“ der Datenbank, die die Stored Procedure ausführt, bereitzustellen. Weitere Informationen siehe <i>dsm-name</i>.</p>							
<p>[USING]</p> <p><i>parameter</i></p>	<p>Zu übergebende Parameter:</p> <p>Hier können Sie Parameter angeben, die vom aufrufenden Objekt an die Stored Procedure übergeben werden sollen.</p> <p>Als <i>parameter</i> können Sie Folgendes angeben:</p> <ul style="list-style-type: none"> ■ <i>host-variable</i> (optional mit INDICATOR- und LINDICATOR-Klauseln) ■ eine Konstante oder ■ das Schlüsselwort NULL. <p>Weitere Informationen siehe <i>host-variable</i>.</p>							
AD=	<p>Attribut-Definition:</p> <p>Wenn es sich bei dem <i>parameter</i> um eine <i>host-variable</i> handelt, können Sie ihn wie folgt markieren:</p> <table border="1" data-bbox="399 1388 1484 1812"> <tr> <td data-bbox="399 1388 911 1530">AD=O</td> <td data-bbox="911 1388 1484 1530">Nicht änderbar, siehe Session-Parameter AD=O. (Entsprechende Prozedur-Notation in DB2 for z/OS: IN.)</td> </tr> <tr> <td data-bbox="399 1530 911 1673">AD=M</td> <td data-bbox="911 1530 1484 1673">Änderbar, siehe Session-Parameter AD=M. (Entsprechende Prozedur-Notation in DB2 for z/OS: INOUT.)</td> </tr> <tr> <td data-bbox="399 1673 911 1812">AD=A</td> <td data-bbox="911 1673 1484 1812">Nur zur Eingabe, siehe Session-Parameter AD=A. (Entsprechende Prozedur-Notation in DB2 for z/OS: OUT.)</td> </tr> </table> <p>Wenn der <i>parameter</i> eine <i>constant</i> ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.</p>		AD=O	Nicht änderbar, siehe Session-Parameter AD=O. (Entsprechende Prozedur-Notation in DB2 for z/OS: IN.)	AD=M	Änderbar, siehe Session-Parameter AD=M. (Entsprechende Prozedur-Notation in DB2 for z/OS: INOUT.)	AD=A	Nur zur Eingabe, siehe Session-Parameter AD=A. (Entsprechende Prozedur-Notation in DB2 for z/OS: OUT.)
AD=O	Nicht änderbar, siehe Session-Parameter AD=O. (Entsprechende Prozedur-Notation in DB2 for z/OS: IN.)							
AD=M	Änderbar, siehe Session-Parameter AD=M. (Entsprechende Prozedur-Notation in DB2 for z/OS: INOUT.)							
AD=A	Nur zur Eingabe, siehe Session-Parameter AD=A. (Entsprechende Prozedur-Notation in DB2 for z/OS: OUT.)							

<p>RESULT SETS <i>result-set</i></p>	<p>Feld für Result-Set-Locator-Variable:</p> <p>Als <i>result-set</i> geben Sie ein Feld an, in das der Result-Set-Locator zurückgegeben werden soll.</p> <p>Ein <i>result-set</i> muss eine Variable mit Format/Länge I4 sein.</p> <p>Der Wert einer <i>result-set</i>-Variablen ist lediglich eine Zahl, die den Result Set identifiziert und die in einem nachfolgenden READ RESULT SET-Statement referenziert werden kann.</p> <p>Die Reihenfolge der <i>result-set</i>-Werte entspricht der Reihenfolge der von der Stored Procedure zurückgegebenen Result Sets.</p> <p>Der Inhalt der Result Sets kann von einem nachfolgenden READ RESULT SET-Statement verarbeitet werden. Wenn kein Result Set zurückgegeben wird, enthält die betreffende <i>result-set</i>-Variable Null (0).</p> <p>Es kann nur ein Result-Set angegeben werden.</p>
<p>GIVING <i>sqlcode</i></p>	<p>GIVING <i>sqlcode</i>-Option:</p> <p>Mit dieser Option können Sie den SQL-Code des SQL CALL-Statements erhalten, das die Stored Procedure aufruft.</p> <p>Wenn Sie diese Option angeben und der SQL-Code der Stored Procedure ist nicht Null (0), wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muss die als Reaktion auf den SQL-Codewert auszuführende Handlung im aufrufenden Natural-Objekt programmiert werden.</p> <p>Das <i>sqlcode</i>-Feld muss eine Variable von Format/Länge I4 sein.</p> <p>Wenn Sie die Option GIVING <i>sqlcode</i> nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQL-Code der Stored Procedure nicht Null (0) ist.</p>

Beispiel

Das folgende Beispiel zeigt ein Natural-Programm, das die Stored Procedure `DEMO_PROC` aufruft, um alle zu einem gegebenen Bereich gehörenden Namen der Tabelle `PERSON` einzulesen.

Drei Parameter-Felder werden an `DEMO_PROC` übergeben: der erste und zweite Parameter übergeben jeweils Start- und Endwerte des Bereichs von Namen an die Stored Procedure, und der dritte Parameter nimmt einen Namen auf, der das Kriterium erfüllt.

In diesem Beispiel werden die Namen in einem Result Set zurückgegeben, der mit dem `READ RESULT SET`-Statement verarbeitet wird.

```
DEFINE DATA LOCAL
1 PERSON VIEW OF DEMO-PERSON
  2 PERSON_ID
  2 LAST_NAME
1 #BEGIN      (A2) INIT <'AB'>
1 #END        (A2) INIT <'DE'>
1 #RESPONSE   (I4)
1 #RESULT     (I4)
1 #NAME      (A20)
END-DEFINE

...

CALLDBPROC 'DEMO_PROC' DEMO-PERSON #BEGIN (AD=0) #END (AD=0) #NAME (AD=A)
  RESULT SETS #RESULT
  GIVING #RESPONSE

READ RESULT SET #RESULT INTO #NAME FROM DEMO-PERSON
  GIVING #RESPONSE
  DISPLAY #NAME
END-RESULT

...

END
```


145 COMMIT - SQL

- Funktion 970
- Beispiel 970

COMMIT

Dieses Kapitel behandelt folgende Themen:

Funktion

Das SQL-Statement `COMMIT` entspricht dem `END TRANSACTION`-Statement. Es markiert das Ende einer logischen Transaktion und bewirkt, dass alle während der Transaktion gesperrten Daten freigegeben werden. Alle Datenänderungen werden bestätigt und auf der Datenbank physisch durchgeführt.



Wichtig: Da bei Beendigung einer logischen Arbeitseinheit alle Cursor geschlossen werden, darf ein `COMMIT`-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muss nach einer solchen stehen (bzw. bei geschachtelten Schleifen nach der äußersten Schleife).

Beispiel

```
...  
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'  
COMMIT  
...
```

146 DELETE - SQL

▪ Funktion	972
▪ Syntax-Beschreibung	972

Dieses Kapitel behandelt folgende Themen:

Funktion

Das SQL-Statement `DELETE` dient dazu, Reihen aus einer Tabelle zu löschen, ohne einen Cursor zu verwenden (Searched `DELETE`), oder Reihen aus einer Tabelle zu löschen, auf die der Cursor zeigt (Positioned `DELETE`).

Syntax-Beschreibung

Zwei unterschiedliche Strukturen sind möglich:

- [Syntax 1 — Searched DELETE](#)
- [Syntax 2 — Positioned DELETE](#)

Syntax 1 — Searched DELETE

Searched `DELETE` ist ein eigenständiges Statement, das unabhängig von einem `SELECT`-Statement verwendet werden kann. Mit einem einzigen Statement können sie keine, eine, mehrere oder alle Reihen einer Tabelle löschen. Welche Reihen gelöscht werden, bestimmen Sie mit einer Suchbedingung (*search-condition*), die auf die Tabelle angewandt wird. Außerdem ist es möglich, dem Tabellennamen einen *correlation-name* zuzuweisen.



Anmerkung: Die Anzahl der Reihen, die mit einem Searched `DELETE` tatsächlich gelöscht wurden, kann mit der Systemvariablen `*ROWCOUNT` ermittelt werden.

Common Set-Syntax:

```
DELETE FROM table-name [(correlation-name)] [WHERE search-condition]
```

Extended Set-Syntax:

```
DELETE FROM table-name [(correlation-name)] [WHERE search-condition]
    [ WITH { RR } ] [QUERYNO integer]
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax-Element-Beschreibung:

FROM <i>table-name</i>	FROM-Klausel: In dieser Klausel wird die Tabelle angegeben, aus der die Reihen gelöscht werden sollen.
<i>correlation-name</i>	Als Option kann dem Tabellen-Namen ein <i>correlation-name</i> zugewiesen werden.
WHERE <i>search-condition</i>	WHERE-Klausel: Diese Klausel wird benutzt, um die Selektionskriterien für die zu löschenden Reihen anzugeben. Wenn keine WHERE-Klausel angegeben wird, wird die gesamte Tabelle gelöscht.
WITH	WITH Isolation Level-Klausel: Diese Klausel gehört zum SQL Extended Set . Diese Klausel ermöglicht die explizite Angabe des beim Suchen der zu löschenden Reihe benutzten Isolation Level. Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
QUERYNO <i>integer</i>	QUERYNO-Klausel: Diese Klausel gehört zum SQL Extended Set . Diese Klausel wird zurzeit nicht unterstützt und wird ignoriert.

Syntax 2 — Positioned DELETE

Ein Positioned DELETE bezieht sich auf einen Cursor innerhalb einer Datenbankschleife. Es muss daher dieselbe Tabelle referenzieren wie das entsprechende SELECT-Statement, sonst wird eine Fehlermeldung ausgegeben. Ein Positioned DELETE kann nur mit cursor-orientierter Selektion verwendet werden.

In seiner Funktion entspricht Positioned DELETE dem gewöhnlichen Natural-Statement DELETE.

```
DELETE FROM table-name WHERE CURRENT OF CURSOR [(r)]
```

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Syntax-Element-Beschreibung:

FROM <i>table-name</i> WHERE CURRENT OF CURSOR	FROM-Klausel: In dieser Klausel wird die Tabelle angegeben, aus der die Reihen gelöscht werden sollen.
(<i>r</i>)	Statement-Referenz: Die Notation (<i>r</i>) dient zur Referenzierung des Statements, das zur Selektion der zu löschenden Reihe verwendet wurde. Wenn keine Statement-Referenz angegeben wird, dann bezieht sich das DELETE-Statement auf die jeweils innerste aktive Verarbeitungsschleife, mit der der Datensatz, der gelöscht werden soll, ausgewählt wurde.

147

INSERT - SQL

▪ Funktion	976
▪ Syntax-Beschreibung	976
▪ Beispiel	982

Common Set-Syntax:

```
INSERT INTO table-name { (*) [VALUES-clause]
                        [(column-list)] VALUE-LIST }
```

Extended Set-Syntax:

```
INSERT INTO table-name { (*) [OVERRIDING USER VALUE] [VALUES-clause]
                        [(column-list)] [OVERRIDING USER VALUE] VALUE-LIST }
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das SQL-Statement `INSERT` dient dazu, einer Tabelle eine oder mehrere neue Reihen hinzuzufügen.

Syntax-Beschreibung

INTO <i>table-name</i>	<p>INTO-Klausel:</p> <p>In der INTO-Klausel geben Sie an, welcher Tabelle Reihen hinzugefügt werden sollen.</p> <p>Siehe auch table-name.</p>
<i>column-list</i>	<p>column-list:</p> <p>Syntax:</p> <pre><i>column-name</i>...</pre> <p>In der <i>column-list</i> können Sie eine oder mehrere Spalten angeben, die in der hinzugefügten Reihe Werte erhalten sollen.</p> <p>Die Reihenfolge der angegebenen Spalten muss der Reihenfolge der Werte entsprechen, die in der <i>insert-item-list</i> oder im angegebenen View sind (siehe unten).</p> <p>Wenn Sie keine <i>column-list</i> angeben, werden die in der <i>insert-item-list</i> bzw. im View angegebenen Werte entsprechend der impliziten Liste aller Spalten eingefügt, und zwar in der Reihenfolge, in der sie in der Tabelle stehen.</p>

<i>VALUES-clause</i>	<p>Values-Klausel:</p> <p>Mit dieser Klausel fügen Sie eine einzelne Reihe in die Tabelle ein. Siehe VALUES-Klausel weiter unten.</p>
<i>insert-item-list</i>	<p>INSERT Single Row:</p> <p>In der <i>insert-item-list</i> können Sie einen oder mehrere Werte angeben, die den in der <i>column-list</i> angegebenen Spalten zugewiesen werden sollen. Die Reihenfolge der angegebenen Werte muss der der Spalten entsprechen.</p> <p>Wenn keine <i>column-list</i> angegeben wird, werden die Werte in der <i>insert-item-list</i> nach einer impliziten Liste mit allen Spalten in der Reihenfolge eingefügt, wie sie in der Tabelle vorkommen.</p> <p>Die in der <i>insert-item-list</i> anzugebenden Werten können <i>constants</i>, <i>parameters</i>, <i>special-registers</i> oder NULL sein.</p> <p>Informationen zu <i>view-name</i> siehe Abschnitt <i>Basic Syntactical Items</i>, <i>constant</i> und <i>parameter</i>. Siehe auch die Informationen zu <i>special-register</i>.</p> <p>Wenn der Wert NULL zugewiesen wurde, bedeutet dies, dass das adressierte Feld keinen Wert (auch nicht Wert 0 oder leer) erhalten soll.</p> <p>Beispiel - INSERT Single Row:</p> <pre> ... INSERT INTO SQL-PERSONNEL (NAME,AGE) VALUES ('ADKINSON',35) ... </pre>
OVERRIDING USER VALUE	<p>OVERRIDING USER VALUE-Klausel:</p> <p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.</p>

VALUES-Klausel

Mit der VALUES-Klausel fügen Sie eine einzelne Reihe in die Tabelle ein. Der VALUES-Klausel kann entweder ein Stern (*) oder eine *column-list* vorangestellt werden, und sie hat dementsprechend eine der folgenden Formen:

VALUES-Klausel mit vorangehender Stern-Notation

```
VALUES (VIEW view-name)
```

Wenn Sie Stern-Notation angeben, *müssen* Sie in der VALUES-Klausel einen View angeben. Mit den Feldwerten des Views wird dann eine neue Reihe in die Tabelle eingefügt, wobei die Feldnamen des Views als Spaltennamen der Reihe verwendet werden.

VALUES-Klausel mit vorangehender *column-list*

```
VALUES ( { VIEW view-name
          insert-item-list } )
```

Wenn Sie eine *column-list* angeben und in der VALUES-Klausel einen View referenzieren, muss die Anzahl der Spalten in der *column-list* der Anzahl der Felder im View innerhalb der VALUE-LIST entsprechen.

Wenn Sie keine *column-list* angeben, werden die im View angegebenen Werte entsprechend der impliziten Liste aller Spalten in der Reihenfolge, in der sie in der Tabelle stehen, eingefügt.

VALUE-LIST

Common Set-Syntax:

```
{ VALUES { (VIEW view-name)
            (insert-item-list) } [FOR-n-ROWS-clause] }
```

Extended Set-Syntax:

```
{ VALUES { (VIEW view-name)
            (insert-item-list) } [FOR-n-ROWS-clause]
  [WITH_CTE common-table-expression,...] select-expression [ WITH { RR
                                                                RS
                                                                CS } ] [QUERYNO
                                                                integer] ] }
```

Syntax-Beschreibung:

VIEW <i>view-name</i>	<p>View-Name:</p> <p>Mit den Feldwerten dieses Views wird eine neue Reihe in die angegebene Tabelle eingefügt, wobei die Feldnamen des Views als Spaltennamen der Reihe benutzt werden.</p>
<i>insert-item-list</i>	<p>INSERT Single Row:</p> <p>In der <i>insert-item-list</i> können Sie einen oder mehrere Werte angeben, die den in der <i>column-list</i> angegebenen Spalten zugewiesen werden sollen. Die Reihenfolge der angegebenen Werte muss mit der Reihenfolge der Spalten übereinstimmen.</p> <p>Wenn keine <i>column-list</i> angegeben wird, werden die Werte in der <i>insert-item-list</i> nach einer impliziten Liste mit allen Spalten in der Reihenfolge eingefügt, wie sie in der Tabelle vorkommen.</p> <p>Die in der <i>insert-item-list</i> anzugebenden Werte können Konstanten, Parameter, <i>special-registers</i> oder NULL sein.</p> <p>Informationen zu <i>view-name</i>, <i>constant</i> und <i>parameter</i> siehe <i>Basic Syntactical Items</i>. Siehe auch die Informationen zu <i>special-register</i>.</p> <p>Wenn der Wert NULL zugewiesen worden ist, bedeutet dies, dass das adressierte Feld keinen Wert erhalten soll (auch nicht den Wert 0 oder Leerzeichen).</p> <p>Beispiel - INSERT Single Row:</p> <pre> ... INSERT INTO SQL-PERSONNEL (NAME,AGE) VALUES ('ADKINSON',35) ... </pre>
<i>FOR-n-ROWS-clause</i>	Optionale Klausel, siehe weiter unten.
WITH_CTE <i>common-table-expression</i>	<p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Diese optionale Klausel ermöglicht die Definition einer Ergebnistabelle, die in einer FROM-Klausel des folgenden SELECT-Statements referenziert werden kann. Mehrere <i>common-table-expressions</i> können nach dem einzelnen Schlüsselwort WITH_CTE angegeben werden. Jede <i>common-table-expression</i> kann auch in der FROM-Klausel der nachfolgenden <i>common-table-expressions</i> referenziert werden.</p> <p>Weitere Informationen siehe <i>SELECT - Cursor-Oriented, WITH CTE common-table-expression,....</i></p>

<p><i>select-expression</i></p>	<p>INSERT Multiple Rows:</p> <p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Mit einem <i>select-expression</i> können Sie mehrere Reihen in eine Tabelle einfügen. Der <i>select-expression</i> wird ausgewertet und jede Reihe der Ergebnistabelle wird so behandelt, als ob die Werte in der Reihe als Werte in einer VALUES-Klausel einer Single-Row-Insert-Operation angegeben werden.</p> <p>Weitere Informationen siehe Select Expressions.</p> <p>Beispiel - INSERT Multiple Rows:</p> <pre> ... INSERT INTO SQL-RETIREE (NAME,AGE,SEX) SELECT LASTNAME, AGE, SEX FROM SQL-EMPLOYEES WHERE AGE > 60 ... </pre> <p>Anmerkung: Die Anzahl der tatsächlich eingefügten Reihen können Sie mit der Systemvariablen *ROWCOUNT überprüfen (siehe <i>Systemvariablen-Dokumentation</i>).</p>
<p>WITH RR/RS/CS</p>	<p>WITH Isolation Level-Klausel:</p> <p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Diese Klausel ermöglicht die explizite Angabe des zum Auffinden der einzufügenden Reihen benutzten Isolation Level. Sie ist nur bei DB2-Datenbanken gültig. Wird sie bei anderen Datenbanken verwendet, verursacht sie Laufzeitfehler.</p>
<p>QUERYNO_<i>integer</i></p>	<p>QUERYNO-Klausel:</p> <p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Diese Klausel wird zurzeit nicht unterstützt und wird ignoriert.</p>

FOR-n-ROWS-Klausel:

```

FOR { [:_host-variable]
      integer
    } ROWS [ { ATOMIC
              NOT ATOMIC CONTINUE ON SQLEXCEPTION
            } ]

```

Diese Klausel setzt sich aus den folgenden Subklauseln zusammen:

FOR [:] *hostvariable/integer* ROWS-Klausel:

FOR { [:]_ <i>host-variable</i> } ROWS { <i>integer</i> }
--

Die Angabe dieser Klausel ist optional. Sie sollte nur angegeben werden, wenn

- die Compiler-Option `DB2ARRAY` angegeben wird und
- mehrere Reihen von Arrays eingefügt werden sollen, die in der *insert-item-list* der `VALUES`-Klausel angegeben worden sind.

Wenn sie angegeben wird, legt die Option [:] *hostvariable/integer* die Anzahl der Reihen fest, die in die DB2-Tabelle eingefügt werden sollen, und zwar von den Arrays, die in der *insert-item-list* der `VALUES`-Klausel ab der ersten Ausprägung angegeben wurden.

Diese Klausel soll die Verarbeitungszeit der Programme verbessern, mittels derer Reihen von Natural-Arrays in einer Schleife eingefügt werden. Anhand dieser Klausel können die in den Arrays enthaltenen Reihen von einem SQL-Statement eingefügt werden.

Siehe Beispiel weiter unten.

ATOMIC-Klausel:

{ ATOMIC NOT ATOMIC CONTINUE ON SQLEXCEPTION }

Diese Klausel gibt an, ob die Einfügung mehrerer Reihen von DB2 als eine Atomic-Operation behandelt werden sollte oder nicht.

Sie sollte nur angegeben werden, wenn

- die Compiler-Option `DB2ARRAY` angegeben wird und
- mehrere Reihen von Arrays eingefügt werden sollen, die in der *insert-item-list* der `VALUES`-Klausel angegeben worden sind.

Syntax-Beschreibung:

ATOMIC	Gibt an, dass im Falle eines Fehlers keine Reihe in die Zieltabelle eingefügt wird. Dies ist die Voreinstellung.
NOT ATOMIC CONTINUE ON SQLEXCEPTION	Gibt an, dass im Falle von Fehlern alle Reihen eingefügt werden, für die keine Fehler aufgetreten sind, während diejenigen Reihen, für die Fehler aufgetreten sind, von DB2 entfernt werden.

In solchen Fällen zurückgegebene *sqlcodes* entnehmen sie der DB2 SQL REFERENCE.

Beispiel

```

DEFINE DATA LOCAL
01 NAME          (A20/1:10)  INIT <'ZILLER1','ZILLER2','ZILLER3','ZILLER4'
                                , 'ZILLER5','ZILLER6','ZILLER7','ZILLER8'
                                , 'ZILLER9','ZILLERA'>
01 ADDRESS       (A100/1:10) INIT <'ANGEL STREET 1','ANGEL STREET 2'
                                , 'ANGEL STREET 3','ANGEL STREET 4'
                                , 'ANGEL STREET 5','ANGEL STREET 6'
                                , 'ANGEL STREET 7','ANGEL STREET 8'
                                , 'ANGEL STREET 9','ANGEL STREET 10'>
01 DATENATD (D/1:10)  INIT <D'1954-03-27',D'1954-03-27',D'1954-03-27'
                                ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                                ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                                ,D'1954-03-27'>
01 SALARY        (P4.2/1:10) INIT <1000,2000,3000,4000,5000
                                ,6000,7000,8000,9000,9999>
01 L$ADDRESS     (I2/1:10)  INIT <14,14,14,14,14,14,14,14,14,15>
01 N$ADDRESS     (I2/1:10)  INIT <00,00,00,00,00,00,00,00,00,00>
01 ROWS          (I4)
01 INDEX         (I4)
01 V1 VIEW OF NAT-DEMO_ID
02 NAME
02 ADDRESS       (EM=X(20))
02 DATEOFBIRTH
02 SALARY
01 ROWCOUNT    (I4)
END-DEFINE
OPTIONS DB2ARRY=ON          /* <-- ENABLE DB2 ARRAY
ROWCOUNT := 10
INSERT INTO NAT-DEMO_ID
  (NAME,ADDRESS,DATEOFBIRTH,SALARY)
  VALUES
  (:NAME(*),                /* <-- ARRAY
   :ADDRESS(*)              /* <-- ARRAY
   INDICATOR :N$ADDRESS(*)  /* <-- ARRAY
   LINDICATOR :L$ADDRESS(*), /* <-- ARRAY DB2 VCHAR
   :DATENATD(1:10),        /* <-- ARRAY NATURAL DATES
   :SALARY(01:10)          /* <-- ARRAY NATURAL PACKED
  )
  FOR :ROWCOUNT ROWS
SELECT * INTO VIEW V1 FROM NAT-DEMO_ID WHERE NAME > 'Z'
DISPLAY V1                  /* <-- VERIFY INSERT
END-SELECT
END

```

148

PROCESS SQL

- Funktion 984
- Syntax-Beschreibung 984
- Entire Access-Optionen 985
- Beispiele 985

```
PROCESS SQL ddm-name <<statement-string>>
```

Dieses Kapitel behandelt folgende Themen:

Funktion

Das Statement `PROCESS SQL` dient dazu, mit SQL-Statements auf eine Datenbank zuzugreifen.

Syntax-Beschreibung

<i>ddm-name</i>	Mit dem DDM-Namen geben Sie an, für welche die Stored Procedure ausführende Datenbank die angegebenen SQL-Statements abgearbeitet werden sollen. Weitere Informationen zu ddm-name .
<i>statement-string</i>	<p>Die Statements, die Sie im <i>statement-string</i> angeben können, sind dieselben, die Sie auch mit dem SQL-Statement EXECUTE (vgl. Flexible SQL) ausführen können.</p> <p>Vorsicht: Um Transaktionssynchronisationsprobleme zwischen Natural und der zugrundeliegenden Datenbank zu vermeiden, dürfen die Statements COMMIT und ROLLBACK im <code>PROCESS SQL</code>-Statement nicht verwendet werden.</p> <p>Der <i>statement-string</i> kann über mehrere Zeilen gehen, ohne dass am Zeilenende ein Fortsetzungszeichen erforderlich ist. Er kann ganze oder teilweise Kommentarzeilen enthalten.</p> <p>Der <i>statement-string</i> darf auch Parameter enthalten; siehe Parameter weiter unten.</p>

Parameter

```
[ :U ] :host-variable [INDICATOR:host-variable] [LINIDICATOR:host-variable]
```

Im Gegensatz zu den an anderer Stelle beschriebenen [Parametern](#) muss hier den *host-variables* ein Doppelpunkt (:) vorangestellt werden. Außerdem kann ihnen ein weiterer Qualifier (:U bzw. :G) vorangestellt werden.

Weitere Informationen siehe [host-variable](#).

Syntax-Element-Beschreibung:

:U: <i>host-variable</i>	Der Präfix :U qualifiziert die <i>host-variable</i> als sogenannte Using-Variable; d.h. ihr Wert wird an die Datenbank <i>übergeben</i> . :U ist der Standardpräfix.
:G: <i>host-variable</i>	Der Präfix :G qualifiziert die <i>host-variable</i> als sogenannte Giving-Variable; d.h. sie <i>erhält</i> einen Wert <i>von</i> der Datenbank.

Entire Access-Optionen

Mit Entire Access können Sie Folgendes auch als *statement-string* angeben:

- SET SQLOPTION *option* = *value*
- SQLCONNECT *option* = *value*
- SQLDISCONNECT

Diese Optionen gelten nur für Entire Access. Sie sind im Abschnitt *Daten in einer SQL-Datenbank aufrufen* (im Leitfaden zur Programmierung) beschrieben.

Beispiele

Beispiel für Adabas D:

```
PROCESS SQL ADABAS_D_DDM << LOCK TABLE EMPLOYEES IN SHARE MODE >>
```

Beispiel für den Aufruf einer in Adabas D gespeicherten Prozedur:

Die aufgerufene Prozedur berechnet die Summe zweier Zahlen.

```
...
COMPUTE #N1 = 1
COMPUTE #N2 = 2
COMPUTE #SUM = 0
...
PROCESS SQL ADABAS_D_DDM << DBPROCEDURE DEMO.SUM (:#N1, :#N2, :G:#SUM) >>
...
WRITE #N1 '+' #N2 ' =' #SUM
...
```


149

READ RESULT SET - SQL

▪ Funktion	988
▪ Syntax-Beschreibung	988
▪ Beispiel	989

Common Set-Syntax:

```

READ [(limit)] RESULT SET result-set INTO { VIEW view-name } FROM ddm-name
      parameter
[GIVING [:] sql-code]
END-RESULT
    
```

Extended Set-Syntax:

```

READ [(limit)] RESULT SET result-set { VIEW view-name } FROM ddm-name
INTO parameter
[WITH INSENSITIVE SCROLL [:] scroll-hv]
[GIVING [:] sql-code]
integer
END-RESULT
    
```

Dieses Kapitel behandelt folgende Themen:

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt [Syntax-Symbole](#).

Funktion

Das SQL-Statement `READ RESULT SET` kann nur in Verbindung mit einem `CALLDBPROC`-Statement verwendet werden. Es dient dazu, einen Result Set zu lesen, der von einer mit einem vorhergehenden `CALLDBPROC`-Statement aufgerufenen Stored Procedure erzeugt wurde.

Syntax-Beschreibung

<i>limit</i>	Sie können die Anzahl der zu lesenden Reihen begrenzen. Sie können das <i>limit</i> entweder als numerische Konstante (0 bis 99999999) oder als Variable mit Format N, P oder I angeben.
<i>result-set</i>	Als <i>result-set</i> geben Sie eine Result-Set-Locator-Variable an, die mit einem vorhergehenden <code>CALLDBPROC</code> -Statement gefüllt wurde. <i>Result-set</i> muss eine Variable von Format/Länge I4 sein. Anmerkung: Falls zwischen dem <code>CALLDBPROC</code> -Statement und dem <code>READ RESULT SET</code> -Statement eine Syncpoint-Operation stattfand, kann das <code>READ RESULT SET</code> -Statement nicht mehr auf die Result Sets zugreifen.

FROM <i>dgm-name</i>	Als <i>dgm-name</i> geben Sie den Namen des DDMs an, das benutzt wird, um auf die Datenbank zuzugreifen, die die Stored Procedure ausführt. Weitere Informationen siehe <i>dgm-name</i> .
WITH INSENSITIVE SCROLL [:] <i>scroll_hv</i>	Diese Klausel gehört zum SQL Extended Set . Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.
GIVING <i>sqlcode</i>	Mit dieser Option erhalten Sie den SQL-Code der SQL-Fetch-Operation, mit der der Result Set verarbeitet wurde. Wenn Sie diese Option angeben und der SQL-Code der SQL-Operation ist nicht 0, wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muss die als Reaktion auf den SQL-Codewert auszuführende Handlung im aufrufenden Natural-Objekt programmiert werden. Das <i>sqlcode</i> -Feld muss eine Variable mit Format/Länge I4 sein. Wenn Sie die Option GIVING <i>sqlcode</i> nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQL-Code nicht 0 ist.
END-RESULT	Das für Natural reservierte Schlüsselwort END-RESULT muss zum Beenden des READ RESULT SET -Statements verwendet werden.

Beispiel

Siehe das **Beispiel** beim **CALLDBPROC**-Statement.

150

ROLLBACK - SQL

▪ Funktion	992
▪ Hinweis für Nicht-Natural-Programme	992
▪ Beispiel	992

ROLLBACK

Dieses Kapitel behandelt folgende Themen:

Funktion

Das SQL-Statement `ROLLBACK` entspricht dem Natural-Statement `BACKOUT TRANSACTION`. Es macht alle seit dem Beginn der letzten Recovery Unit ausgeführten Datenbankänderungen rückgängig. Eine Recovery Unit beginnt entweder zu Beginn der Session oder nach einem `SYNCPPOINT-`, `COMMIT-`, `END TRANSACTION-` oder `BACKOUT TRANSACTION-`Statement. Außerdem bewirkt `ROLLBACK`, dass alle während der Transaktion gehaltenen Datensätze freigegeben werden.

Bei dem Versuch, Datenbankänderungen, die bereits durch einen Terminal-I/O bestätigt wurden, mit `ROLLBACK` wieder rückgängig zu machen, gibt Natural die Fehlermeldung NAT3711 aus.



Vorsicht: Da bei Beendigung einer logischen Arbeitseinheit alle Cursor geschlossen werden, darf ein `ROLLBACK`-Statement nicht innerhalb einer datenbankverändernden Verarbeitungsschleife stehen, sondern muss nach einer solchen stehen (bzw. bei geschachtelten Schleifen nach der äußersten Schleife).

Hinweis für Nicht-Natural-Programme

Wenn ein Natural-Programm ein Nicht-Natural-Programm aufruft, sollte das aufgerufene Programm kein eigenes `ROLLBACK`-Statement enthalten, falls das aufrufende Natural-Programm selbst auch Datenbankaufrufe durchführt. In diesem Falle sollte das Natural-Programm das `ROLLBACK`-Statement für das Nicht-Natural-Programm enthalten.

Beispiel

```
...  
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'  
ROLLBACK  
...
```

151

SELECT - SQL

▪ Funktion	994
▪ Syntax-Beschreibung	994
▪ Join-Abfragen	1008
▪ SELECT – Cursor-orientierte Auswahl	1009

Dieses Kapitel behandelt folgende Themen:

Funktion

Gemäß der Standard-SQL-Funktionalität unterstützt Natural sowohl das cursor-orientierte SELECT, mit dem eine beliebige Anzahl von Reihen gelesen werden kann, als auch das nicht cursor-orientierte Singleton SELECT, das maximal eine Reihe liest.

Mit dem Konstrukt SELECT ... END-SELECT verwendet Natural die gleiche Datenbankschleifen-Verarbeitung wie beim FIND-Statement.

Syntax-Beschreibung

Zwei verschiedene Strukturen sind möglich:

Syntax 1 – Cursor-orientierte Auswahl

Common Set-Syntax

SELECT <i>selection</i> INTO	{	<i>parameter</i> ,...	
		VIEW { <i>view-name</i>	
		[<i>correlation-name</i>]},...	
[{	UNION	
		EXCEPT	
		INTERSECT	
		[ALL] [(SELECT <i>selection table-expression</i> [])]]
ORDER BY	{	<i>integer</i>	
		<i>column-reference</i>	
		<i>expression</i>	
		[ASC
			DESC
]	
<i>statement</i> ...			
{	END-SELECT (<i>structured mode only</i>)	}	
	LOOP (<i>reporting mode only</i>)		

Extended Set-Syntax:

[WITH_CTE <i>common-table-expression</i> ,...]			
SELECT <i>selection</i> INTO	{	<i>parameter</i> ,... VIEW { <i>view-name</i> [<i>correlation-name</i>]},...	
[{	UNION EXCEPT INTERSECT]
		[ALL] [(SELECT <i>selection table-expression</i>)]	
ORDER BY	{	<i>integer</i> <i>column-reference</i> <i>expression</i> INPUT SEQUENCE	[ASC DESC]
[OPTIMIZE FOR <i>integer</i> ROWS]			
	{	CS RR UR RS RS KEEP UPDATE LOCKS RR KEEP UPDATE LOCKS	}
QUERYNO <i>integer</i>			
[FETCH FIRST	[{ 1 <i>integer</i> }]	[{ ROW ROWS } ONLY]
[WITH HOLD]			
[WITH RETURN]			
	{	ASENSITIVE SCROLL INSENSITIVE SCROLL SENSITIVE STATIC SCROLL SENSITIVE DYNAMIC SCROLL	[:] <i>scroll_hv</i> [GIVING [:] <i>sqlcode</i>]
[WITH ROWSET POSITIONING FOR	{ [:] <i>row_hv</i> <i>integer</i> }	ROWS] ROWS_RETURNED [:] <i>ret_row</i>]
[IF-NO-RECORDS-FOUND- <i>clause</i>]			
<i>statement</i> ...			
{	END-SELECT (<i>structured mode only</i>)	}	

LOOP
(*reporting mode
only*)

Syntax-Elementbeschreibung – Syntax 1:

SELECT <i>selection</i>	<p>Das cursor-orientierte SELECT-Statement dient (wie das FIND-Statement) dazu, ausgehend von einem Suchkriterium Reihen (Rows) von einer oder mehreren Datenbanktabellen auszuwählen. Darüber hinaus wird die Cursor-Verwaltung von Natural automatisch erledigt und muss daher nicht mehr im Anwendungsprogramm kodiert werden.</p> <p>Weitere Informationen siehe SELECT-Cursor-Oriented weiter unten.</p>
INTO	<p>In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen.</p> <p>Weitere Informationen und Beispiele siehe INTO-Klausel weiter unten.</p>
VIEW	<p>Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der <i>selection</i> gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt).</p> <p>Weitere Informationen und Beispiele siehe VIEW-Klausel weiter unten.</p>
<i>table-expression</i>	<p>Die <i>table-expression</i> besteht aus einer FROM-Klausel und außerdem einer optionalen WHERE-Klausel.</p> <p>Weitere Informationen und Beispiele siehe <i>table-expression</i> weiter unten.</p>
UNION	<p>UNION vereinigt die Ergebnisse von zwei oder mehr <i>select-expressions</i> miteinander. Weitere Informationen und ein Beispiel siehe Abfrage mit UNION weiter unten.</p>
ORDER BY	<p>Die ORDER BY-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.</p> <p>Weitere Informationen und Beispiele siehe ORDER BY-Klausel weiter unten.</p>
IF NO RECORDS FOUND	<p>Mit der IF NO RECORDS FOUND-Klausel können Sie eine Schleifen-Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen SELECT-Statement angegebenen Selektionskriterien erfüllt.</p> <p>Weitere Informationen siehe IF NO RECORDS FOUND-Klausel weiter unten.</p>
END-SELECT	<p>Das für Natural reservierte Schlüsselwort END-SELECT muss zum Beenden des SELECT-Statements benutzt werden.</p>

Die folgenden Sytax-Elemente gehören zum **SQL Extended Set**:

WITH_CTE <i>common-table-expression,...</i>	<p>WITH_CTE <i>common-table-expression</i>:</p> <p>Mit dieser Option können Sie eine Ergebnistabelle definieren, die in einer FROM-Klausel eines nachfolgenden SELECT-Statements referenziert werden kann. Nach dem Schlüsselwort WITH_CTE können mehrere <i>common-table-expressions</i> angegeben werden. Jeder dieser Ausdrücke kann in der FROM-Klausel einer nachfolgenden <i>common-table-expression</i> referenziert werden.</p> <p>Weitere Informationen siehe SELECT- Cursor-Oriented.</p>
OPTIMIZE FOR	<p>OPTIMIZE FOR-Klausel:</p> <p>Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.</p>
WITH CS/RS/UR/...	<p>WITH CS/RS/UR/...-Klausel:</p> <p>Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.</p>
QUERYNO	<p>QUERYNO-Klausel:</p> <p>Diese Klausel wird zurzeit nicht unterstützt und wird ignoriert.</p>
FETCH FIRST	<p>FETCH FIRST-Klausel:</p> <p>Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.</p>
WITH HOLD	<p>WITH HOLD-Klausel:</p> <p>Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.</p>
WITH RETURN	<p>WITH RETURN-Klausel:</p> <p>Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.</p>
WITH ... SCROLL	<p>WITH ... SCROLL-Klausel:</p> <p>Beliebig positionierbare RDBMS-Cursor werden mit dieser Klausel aktiviert. Beliebig positionierbare Cursor können entweder <code>ASENSITIVE</code>, <code>INSENSITIVE</code>, <code>SENSITIVE STATIC</code> oder <code>SENSITIVE DYNAMIC</code> sein.</p> <p>Anmerkung: Nicht alle SQL-Datenbanksysteme unterstützen alle Optionen.</p> <ul style="list-style-type: none"> ■ Mit <code>WITH ASENSITIVE SCROLL</code> wird angegeben, dass der Cursor entweder <code>INSENSITIVE</code> oder <code>SENSITIVE DYNAMIC</code> ist. Dies wird von der Datenbank zum Zeitpunkt der Öffnung des Cursors ermittelt, und zwar in Abhängigkeit von der Fähigkeit des Cursors, nur Lesezugriff zu gewährleisten. Wenn über den Cursor nur gelesen werden kann, wird der Cursor <code>INSENSITIVE</code>. Wenn der Cursor Lese- und

	<p>Schreibzugriff garantiert, wird er <code>SENSITIVE DYNAMIC</code>. Dies wird nicht bei DB2-Datenbanken unterstützt.</p> <ul style="list-style-type: none">■ Mit <code>WITH INSENSITIVE SCROLL</code> wird angegeben, dass der Cursor insensitiv ist für Änderungen, Löschungen und auch für Einfügungen, die auf der Datenbanktabelle ausgeführt werden, nachdem er geändert worden ist. <code>Positioned Updates</code> und <code>Deletes</code> sind nicht zulässig beim <code>INSENSITIVE SCROLL</code>-Cursor. Dies wird unterstützt bei Oracle-, Adabas D-, Informix-, MS SQL Server ODBC- und DB2-Datenbanken.■ Mit <code>WITH SENSITIVE STATIC</code> wird angegeben, dass der Cursor sensitiv ist für Änderungen und Löschungen, die gegen die Datenbanktabelle ausgeführt werden, aber nicht für Einfügungen, nachdem der Cursor geöffnet worden ist. <code>Positioned Updates</code> und <code>Deletes</code> sind für <code>SENSITIVE STATIC SCROLL</code>-Cursor zulässig. Dies wird unterstützt bei Adabas D-, MS SQL Server ODBC- und DB2-Datenbanken.■ Mit <code>WITH SENSITIVE DYNAMIC</code> wird angegeben, dass der Cursor sensitiv ist für Aktualisierungen, Löschungen und Einfügungen gegen die Basistabelle, nachdem der Cursor geöffnet wurde. <code>Positioned Updates</code> und <code>Deletes</code> sind für <code>SENSITIVE DYNAMIC SCROLL</code>-Cursor zulässig. Dies wird unterstützt bei Adabas D-, MS SQL Server ODBC- und DB2-Datenbanken. <p>Beliebig positionierbare Cursor ermöglichen es der Anwendung, eine beliebige Reihe im Result Set jederzeit zu positionieren, solange der Cursor offen ist.</p> <p>Beliebig positionierbare Cursor werden bei Sybase-Datenbanken überhaupt nicht unterstützt. Beliebig positionierbare Cursor werden beim MS SQL Server DBLIB Interface nicht unterstützt, sondern nur beim MS SQL Server ODBC Interface.</p> <p>Die Positionierung wird durchgeführt in Abhängigkeit vom Inhalt von <code>scroll_hv</code>. Der Inhalt wird jedesmal ausgewertet, wenn ein <code>FETCH</code> auf der Datenbank ausgeführt wird.</p> <p>Weitere Informationen siehe SELECT – Cursororientierte Auswahl.</p>
--	---

Syntax 2 – Nicht cursor-orientierte Auswahl

Common Set-Syntax

```

SELECT SINGLE

selection INTO          {   parameter ,...
                          VIEW {view-name
                              [correlation-name ]}, ... } table-expression

[IF-NO-RECORDS-FOUND-clause]
statement...

{ END-SELECT (structured mode only) }
{ LOOP (reporting mode only) }

```

Extended Set-Syntax

```

SELECT SINGLE

selection INTO          {   parameter ,...
                          VIEW {view-name
                              [correlation-name ]}, ... } table-expression

[ WITH { CS
        RR
        UR } ]

[IF-NO-RECORDS-FOUND-clause]
statement...

{ END-SELECT (structured mode only) }
{ LOOP (reporting mode only) }

```

Syntax-Elementbeschreibung – Syntax 2:

SELECT SINGLE	SELECT SINGLE unterstützt die Funktionalität eines keine Cursor verwendenden Singleton SELECT, das maximal eine Reihe liest. Es kann nicht von einem Positioned UPDATE- bzw. DELETE-Statement referenziert werden.
INTO	In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen. Weitere Informationen und Beispiele siehe INTO-Klausel weiter unten.
VIEW	Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der <i>selection</i> gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder

	und Indikatorfelder nicht mitgezählt). Weitere Informationen und Beispiele siehe VIEW-Klausel weiter unten.
<i>table-expression</i>	Die <i>table-expression</i> besteht aus einer FROM-Klausel und einer optionalen WHERE-Klausel. Weitere Informationen und Beispiele siehe Abschnitt table-expression weiter unten.
WITH CS/RR/UR	WITH CS/RR/UR-Klausel: Diese Klausel gehört zum SQL Extended Set. Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
IF NO RECORDS FOUND	In der IF NO RECORDS FOUND-Klausel können Sie eine Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen SELECT-Statement angegebenen Selektionskriterien erfüllt. Weitere Informationen, siehe IF NO RECORDS FOUND-Klausel weiter unten.
END-SELECT	Das für Natural reservierte Schlüsselwort END-SELECT muss zum Beenden des SELECT-Statements benutzt werden.

INTO-Klausel

```
INTO { parameter ,...
      VIEW {view-name [correlation-name]},... }
```

In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen. Sie können in der INTO-Klausel entweder einzelne *parameters* oder ganze im DEFINE DATA-Statement definierte Views angeben.

Die Zielfelder können aus einer einzigen Tabelle kommen bzw. bei einer Join-Operation (vgl. Abschnitt [Join-Abfragen](#)) auch aus mehreren.



Anmerkung: In der Standard-SQL-Syntax wird die INTO-Klausel nur in nicht cursor-orientierten Singleton SELECT-Operationen verwendet, bei denen eine einzelne Reihe gelesen werden soll. Natural erlaubt die INTO-Klausel jedoch sowohl in cursor-orientierten als auch in nicht cursor-orientierten SELECT-Operationen.

Die *selection* kann auch aus nur einem Stern (*) bestehen. In einer standardmäßigen *selection-expression* steht dieser für eine Liste aller Spaltennamen der in der FROM-Klausel angegebenen Tabelle(n). Im Natural-SELECT-Statement hat der Ausdruck SELECT * jedoch eine andere Bedeutung: Alle in der INTO-Klausel gemachten Angaben werden auch in der *selection* verwendet, ohne dass sie dort explizit angegeben werden müssen. Ihre Namen müssen vorhandenen Datenbank-Spaltennamen entsprechen.

Beispiele:**Beispiel 1:**

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT *
  INTO NAME, AGE
```

Beispiel 2:

```
...
SELECT *
  INTO VIEW PERS
```

Obige Beispiele sind mit den folgenden gleichwertig:

Beispiel 3:

```
...
SELECT NAME, AGE
  INTO NAME, AGE
```

Beispiel 4:

```
...
SELECT NAME, AGE
  INTO VIEW PERS
```

VIEW-Klausel:

```
VIEW {view-name [correlation-name]}, ...
```

Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der *selection* gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt).



Anmerkung: Die Natural-Zielfelder wie auch die Tabellenspalten müssen im Natural-DDM definiert sein (wobei die Namen allerdings unterschiedlich sein dürfen, da die Zuweisung entsprechend ihrer Reihenfolge erfolgt).

Beispiel einer INTO-Klausel mit View:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT FIRSTNAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
...
```

Die Zielfelder *NAME* und *AGE*, die Teil eines Natural-Views sind, erhalten den Inhalt der Datenbankspalten *FIRSTNAME* und *AGE*.

parameter

Wenn Sie einzelne *parameter* als Zielfelder angeben, müssen sie in Anzahl und Format mit den in der entsprechenden *selection* angegebenen *columns* bzw. *scalar-expressions* übereinstimmen, wie oben beschrieben. Siehe [scalar-expressions](#).

Beispiel:

```
DEFINE DATA LOCAL
01 #NAME (A20)
01 #AGE (I2)
END-DEFINE
...
SELECT NAME, AGE
  INTO #NAME, #AGE
  FROM SQL-PERSONNEL
...
```

	Die Zielfelder <code>#NAME</code> und <code>#AGE</code> , die Natural-Programmvariablen sind, erhalten den Inhalt der Datenbankspalten <code>NAME</code> und <code>AGE</code> .
<i>correlation-name</i>	<p>Wenn die <code>VIEW</code>-Klausel in einem <code>SELECT *</code> verwendet wird, in dem mehrere Tabellen mit <code>JOIN</code> verknüpft werden, sind <i>correlation-names</i> erforderlich, falls der angegebene View Felder enthält, die Spalten referenzieren, welche in mehreren dieser Tabellen vorkommen. Um zu bestimmen, von welcher Spalte ausgewählt werden soll, werden bei der Generierung der Auswahlliste alle diese Spalten mit dem angegebenen <i>correlation-name</i> qualifiziert. Der einem View zugewiesene <i>correlation-name</i> muss einem der <i>correlation-names</i> entsprechen, mit denen die verknüpften Tabellen qualifiziert werden. Siehe auch Join-Abfragen.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 01 PERS VIEW OF SQL-PERSONNEL 02 NAME 02 FIRST-NAME 02 AGE END-DEFINE ... SELECT * INTO VIEW PERS A FROM SQL-PERSONNEL A, SQL-PERSONNEL B ... </pre>

table-expression

Die *table-expression* besteht aus einer `FROM`-Klausel und einer optionalen `WHERE`-Klausel. Die `GROUP BY`- und `HAVING`-Klauseln sind nicht erlaubt.

Beispiel 1:

```

DEFINE DATA LOCAL
01 #NAME      (A20)
01 #FIRSTNAME (A15)
01 #AGE       (I2)
...
END-DEFINE
...
SELECT NAME, FIRSTNAME, AGE
  INTO #NAME, #FIRSTNAME, #AGE
  FROM SQL-PERSONNEL
  WHERE NAME IS NOT NULL
  AND AGE > 20
...
DISPLAY #NAME #FIRSTNAME #AGE

```

```
END-SELECT
...
END
```

Beispiel 2:

```
DEFINE DATA LOCAL
01 #COUNT      (I4)
...
END-DEFINE
...
SELECT SINGLE COUNT(*) INTO #COUNT FROM SQL-PERSONNEL
...
```

Weitere Informationen siehe [selection](#) und [table-expression](#).

Abfrage mit UNION



Anmerkung: Im Folgenden wird der Begriff `SELECT-Statement` verwendet im Sinne einer vollständigen *query-expression*, die aus mehreren mit `UNION` verknüpften *select-expressions* besteht.

`UNION` vereinigt die Ergebnisse von zwei oder mehr *select-expressions* miteinander. Die in den einzelnen *select-expressions* angegebenen Spalten müssen `UNION`-kompatibel sein, d.h. in Anzahl, Typ und Format zueinander passen.

Redundante doppelte Reihen werden immer aus dem Ergebnis einer `UNION` eliminiert, es sei denn, der `UNION`-Operator enthält ausdrücklich ein `ALL`. Allerdings ist es bei `UNION` nicht möglich, `DISTINCT` explizit als Alternative zu `ALL` anzugeben.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
  02 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
  FROM SQL-EMPLOYEES
```



```

WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...

```

Grundsätzlich ist die Anzahl der *select-expressions*, die mit UNION verknüpft werden können, beliebig.

Nur die erste *select-expression* darf eine INTO-Klausel enthalten.

Wird eine ORDER BY-Klausel verwendet, muss sie nach der letzten *selection-expression* angegeben werden. Die zu sortierenden Spalten müssen durch die Spaltennummern identifiziert werden, nicht durch die Spaltennamen.

ORDER BY-Klausel

```

ORDER BY { { integer } [ ASC ] } { column-reference } [ DESC ] } ,...

```

Die ORDER BY-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.

Jede ORDER BY-Klausel muss eine Spalte der Ergebnistabelle spezifizieren. In den meisten ORDER BY-Klauseln wird die Spalte entweder durch eine *column-reference* (also den optional qualifizierten Spaltennamen) oder durch die Spaltennummer identifiziert.

In einer Abfrage mit UNION muss eine Spalte durch die Spaltennummer identifiziert werden. Die Spaltennummer ist die Ordinalzahl, die die Position einer Spalte (von links nach rechts) innerhalb der *selection* angibt, also eine Ganzzahl (*Integer*). Dadurch ist es möglich, ein Ergebnis auf der Grundlage einer berechneten Spalte, die keinen Namen hat, zu sortieren.

<i>expression</i>	Gibt einen Ausdruck mit Operatoren an (d.h. nicht nur einen <i>column-name</i> oder <i>integer</i>).
INPUT SEQUENCE	Gibt an, dass die Ergebnistabelle die Eingabe-Reihenfolge der Reihen reflektiert, die in der VALUES-Klausel eines INSERT-Statements angegeben sind. Eine Sortierung der INPUT SEQUENCE kann nur angegeben werden, wenn ein INSERT-Statement in einer FROM-Klausel vorhanden ist.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME          (A20)
1 #YEARS-TO-WORK (I2)
END-DEFINE
...
SELECT NAME , 65 - AGE
      INTO #NAME, #YEARS-TO-WORK
      FROM SQL-PERSONNEL
      ORDER BY 2
      ...
```

Als Sortierreihenfolge können Sie entweder aufsteigend (ASC = Ascending) oder absteigend (DESC = Descending) angeben. Standardmäßig gilt ASC.

Beispiel:

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS      (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
      INTO VIEW PERS
      FROM SQL-PERSONNEL
      WHERE AGE = 55
      ORDER BY NAME DESC
      ...
```

Weitere Informationen siehe [integer](#)-Werte und [column-reference](#).

IF NO RECORDS FOUND-Klausel:



Anmerkung: Diese Klausel ist eigentlich nicht Bestandteil von Natural SQL; sie stellt eine Natural-Funktion dar, die für SQL-Schleifenverarbeitung zur Verfügung gestellt wird.

Structured Mode-Syntax

```
IF NO [RECORDS] [FOUND]
{
    ENTER
    statement...
}
END- NOREC
```


Reporting Mode-Syntax

```
IF NO [RECORDS] [FOUND]
{
    ENTER
    statement
    DO statement... DOEND
}
```

In der `IF NO RECORDS FOUND`-Klausel können Sie eine Schleifenverarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen `SELECT`-Statement angegebenen Selektionskriterien erfüllt.

Wenn kein Datensatz die angegebenen Selektionskriterien erfüllt, dann löst die `IF NO RECORDS FOUND`-Klausel eine Verarbeitungsschleife aus, die einmal mit einem leeren Datensatz durchlaufen wird. Falls Sie dies nicht wünschen, geben Sie in der `IF NO RECORDS FOUND`-Klausel das Statement `ESCAPE BOTTOM` an.

Enthält die `IF NO RECORDS FOUND`-Klausel ein oder mehrere Statements, werden diese ausgeführt, unmittelbar bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine weiteren Statements ausgeführt werden, muss die `IF NO RECORDS FOUND`-Klausel das Schlüsselwort `ENTER` enthalten.

 **Anmerkung:** Falls das Ergebnis-Set des `SELECT`-Statements aus einer einzelnen Reihe von `NULL`-Werten besteht, wird die `IF NO RECORDS FOUND`-Klausel nicht ausgeführt. Dies kann der Fall sein, wenn die *selection*-Liste nur aus einer der *aggregate-functions* `SUM`, `AVG`, `MIN` oder `MAX` auf Spalten besteht, und der Set, mit dem diese *aggregate-functions* operieren, leer ist. Bei obengenannter Verwendung dieser *aggregate-functions* sollten Sie daher keine `IF NO RECORDS FOUND`-Klausel benutzen, sondern stattdessen die Werte der betreffenden Null-Indikator-Felder abfragen.

Datenbankwerte

Natural setzt alle Datenbankfelder, die die in der aktuellen Verarbeitungsschleife angegebene Datei referenzieren, auf Leerwerte, es sei denn, eines der in der `IF NO RECORDS FOUND`-Klausel angegebenen Statements weist den Feldern andere Werte zu.

Auswertung von Systemfunktionen

Natural-Systemfunktionen werden einmal für den leeren Datensatz ausgewertet, der für die aus der `IF NO RECORDS FOUND`-Klausel resultierende Verarbeitung erstellt wurde.

Join-Abfragen

Ein Join ist eine Abfrage, bei der Daten von mehr als einer Tabelle gelesen werden. Alle betroffenen Tabellen müssen in der `FROM`-Klausel angegeben werden.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #MONEY     (I4)
END-DEFINE
...
SELECT NAME, ACCOUNT
  INTO #NAME, #MONEY
  FROM SQL-PERSONNEL P, SQL-FINANCE F
  WHERE P.PERSNR = F.PERSNR
        AND F.ACCOUNT > 10000
  ...
```

Ein Join bildet immer zunächst das kartesische Produkt der in der `FROM`-Klausel angegebenen Tabellen und eliminiert später von diesem kartesischen Produkt alle Reihen, die die in der `WHERE`-Klausel angegebene Join-Bedingung nicht erfüllen.

Bei längeren Tabellennamen können Sie sich durch Verwendung von *Correlation-names* Schreibarbeit sparen. Wird in einer Join-Abfrage eine Tabelle mit sich selbst verknüpft, ist die Angabe von *correlation-names* erforderlich, um die beiden nötigen Referenzen auf dieselbe Tabelle voneinander zu unterscheiden

SELECT – Cursor-orientierte Auswahl

Wie das Natural `FIND`-Statement wird das cursor-orientierte `SELECT`-Statement benutzt, um mittels einer Suchbedingung eine Untermenge von Reihen (Datensätzen) von einer oder mehreren DB2-Tabelle/n auszuwählen. Da eine Datenbankschleife initiiert wird, muss die Schleife durch ein `LOOP`- (Reporting Mode) oder `END-SELECT`-Statement geschlossen werden. Bei dieser Statement-Struktur verwendet Natural dieselbe Schleifenverarbeitung wie beim `FIND`-Statement.

Außerdem ist vom Anwendungsprogramm keine Cursor-Verwaltung erforderlich; sie wird automatisch von Natural durchgeführt.

Im Folgenden finden Sie Informationen zu:

- `WITH_CTE common-table-expression,...`
- `OPTIMIZE FOR integer ROWS`
- `WITH - Isolation Level`
- `FETCH FIRST`
- `WITH INSENSITIVE/SENSITIVE`

`WITH_CTE common-table-expression,...`

Mit dieser Klausel können Sie Ergebnistabellen definieren, die in jeder `FROM`-Klausel des nachfolgenden `SELECT`-Statements referenziert werden können.

Das Natural-spezifische Schlüsselwort `WITH_CTE` entspricht dem SQL-Schlüsselwort `WITH`. `WITH_CTE` wird durch den Natural-Compiler in das SQL-Schlüsselwort `WITH` umgesetzt.

Jeder *common-table-expression* muss folgender Syntax entsprechen:

```
[common-table-expression-name [(column-name,...)] AS (fullselect) ]
```

Syntax-Beschreibung:

<i>common-table-expression-name</i>	Muss ein nicht qualifizierter SQL-Identifizier sein und muss sich von allen anderen, im selben Statement angegebenen <i>common-table-expression-names</i> unterscheiden. Jeder <i>common-table-expression-name</i> kann in der <code>FROM</code> -Klausel eines darauffolgenden <i>common-table-expression-name</i> oder in der <code>FROM</code> -Klausel des folgenden <code>SELECT</code> -Statements angegeben werden.
<i>column-name</i>	Muss ein nicht qualifizierter SQL-Identifizier sein und muss innerhalb eines <i>common-table-expression-name</i> eindeutig sein.

AS (<i>fullselect</i>)	Die Anzahl der <i>column-names</i> muss gleich der Anzahl der Spalten des <i>fullselect</i> sein.
--------------------------	---

Ein *common-table-expression* kann verwendet werden

- anstelle einer View, wenn man das Erstellen einer View vermeiden möchte;
- wenn dieselbe Ergebnistabelle in einem *fullselect* gemeinsam genutzt werden muss ;
- wenn das Ergebnis durch Rekursion abgeleitet werden muss.

Rekursive Abfragen sind in Anwendungen wie zum Beispiel Stücklisten von Nutzen.

Beispiel:

```
WITH_CTE
RPL (PART,SUBPART,QUANTITY) AS
(SELECT ROOT.PART,ROOT.SUBPART,ROOT.QUANTITY
  FROM HGK-PARTLIST ROOT
  WHERE ROOT.PART ='01'
 UNION ALL
 SELECT CHILD.PART,CHILD.SUBPART,CHILD.QUANTITY
  FROM  RPL PARENT, HGK-PARTLIST CHILD
  WHERE PARENT.SUBPART = CHILD.PART
 )
SELECT DISTINCT PART,SUBPART,QUANTITY
 INTO VIEW V1
 FROM RPL
 ORDER BY PART,SUBPART,QUANTITY
END-SELECT
```

OPTIMIZE FOR integer ROWS

[OPTIMIZE FOR *integer* ROWS]

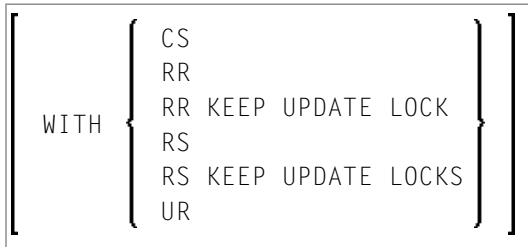
Die OPTIMIZE FOR *integer* ROWS-Klausel wird verwendet, um DB2 im Voraus über die Anzahl (Ganzzahl) von Reihen zu informieren, die von der Ergebnistabelle eingelesen werden sollen. Ohne diese Klausel geht DB2 davon aus, dass alle Reihen der Ergebnistabelle eingelesen werden sollen und führt dementsprechend eine Optimierung durch.

Diese optionale Klausel ist nützlich, wenn Sie wissen, wie viele Reihen wahrscheinlich ausgewählt werden, weil eine Optimierung von Integer-Reihen die Verarbeitungszeit verbessern kann, wenn die Anzahl der tatsächlich ausgewählten Reihen nicht den Ganzzahlwert überschreitet (der im Bereich von 0 bis 2147483647 liegen kann).

Beispiel:

```
SELECT name INTO
#name FROM table WHERE AGE = 2 OPTIMIZE FOR 100 ROWS
```

WITH - Isolation Level



Diese WITH-Klausel ermöglicht es Ihnen, einen expliziten Isolation Level anzugeben, mit dem das Statement ausgeführt werden soll. Die folgenden Optionen stehen zur Verfügung:

Option	Bedeutung
CS	Cursor-Stabilität
RR	Wiederholbarer Lesevorgang
RS	Lese-Stabilität
RS KEEP UPDATE LOCKS	Nur gültig, wenn eine FOR UPDATE OF-Klausel angegeben wird. Lese-Stabilität und Beibehaltung von Aktualisierungssperren.
RR KEEP UPDATE LOCKS	Nur gültig, wenn eine FOR UPDATE OF-Klausel angegeben wird. Wiederholbarer Lesevorgang und Beibehaltung von Aktualisierungssperren.
UR	Freier Lesevorgang

WITH UR kann nur bei einem SELECT-Statement angegeben werden, und wenn es für die Tabelle nur eine Lesezugriffsberechtigung gibt. Der standardmäßige Isolation Level wird durch die Trennung des Pakets oder Plans festgelegt, in den das Statement eingebunden ist. Der standardmäßige Isolation Level ist auch abhängig davon, ob für die Ergebnistabelle nur eine Lesezugriffsberechtigung besteht oder nicht. Um den standardmäßigen Isolation Level zu ermitteln, greifen Sie auf die IBM-Literatur zurück.



Anmerkung: Diese Option funktioniert auch für eine Nicht-Cursor-Auswahl.

FETCH FIRST

```
[ FETCH FIRST {  $\underline{1}$  } { ROWS } ONLY ]
```

Die `FETCH FIRST`-Klausel begrenzt die Anzahl der über `FETCH` abzurufenden Reihen. Es verbessert die Verarbeitungszeit der Abfragen mit möglicherweise großen Result-Sets, wenn nur eine beschränkte Anzahl von Reihen erforderlich ist.

WITH INSENSITIVE/SENSITIVE

```
[ WITH { ASENSITIVE SCROLL
          INSENSITIVE SCROLL
          SENSITIVE STATIC SCROLL
          SENSITIVE DYNAMIC SCROLL } [:] scroll_hv [GIVING [:] sqlcode] ]
```

Natural unterstützt Scrollable Cursor von SQL über die Klauseln `WITH ASENSITIVE SCROLL`, `WITH SENSITIVE STATIC SCROLL` und `SENSITIVE DYNAMIC SCROLL`. Scrollable Cursor ermöglichen es Natural-Anwendungen, eine Reihe in einem Result Set beliebig zu positionieren. Mit Non-Scrollable Cursors können die Daten nur sequentiell vom Anfang zum Ende gelesen werden.

`ASENSITIVE Scrollable Cursors` sind entweder `INSENSITIVE` – wenn der Cursor `READ-ONLY` ist – oder `SENSITIVE DYNAMIC` – wenn der Cursor nicht `READ-ONLY` ist.

`INSENSITIVE` und `SENSITIVE STATIC Scrollable Cursors` benutzen Zwischenergebnis-Tabellen, und für sie ist deshalb eine `TEMP`-Datenbank in DB2 erforderlich (siehe die betreffende DB2-Literatur von IBM).

`INSENSITIVE SCROLL` bezieht sich auf einen Cursor, der bei `Positioned UPDATE-` oder `Positioned DELETE-Operationen` nicht benutzt werden kann. Außerdem reflektiert ein einmal geöffneter `INSENSITIVE SCROLL-Cursor` keine `UPDATES`, `DELETES` oder `INSERTS` gegen die Basistabelle, nachdem der Cursor geöffnet wurde.

`SENSITIVE STATIC SCROLL` bezieht sich auf einen Cursor, der für `Positioned UPDATE-` oder `Positioned DELETE-Operationen` benutzt werden kann. Außerdem reflektiert ein `SENSITIVE STATIC SCROLL-Cursor` `UPDATES`, `DELETES` der Basistabellen-Reihen. Der Cursor reflektiert keine `INSERT-Operationen`.

`SENSITIVE DYNAMIC Scrollable Cursors` reflektieren `UPDATES`, `DELETES` und `INSERTS` gegen die Basistabelle, während der Cursor geöffnet ist.

Nachfolgend finden Sie Informationen zu:

- [scroll_hv](#)
- [scroll_hv - Optionen](#)

- **GIVING** [:] *sqlcode*

scroll_hv

Die Variable *scroll_hv* muss alphanumerisch sein.

Die Variable *scroll_hv* gibt an, welche Reihe der Ergebnistabelle während einer Ausführung der Datenbank-Verarbeitungsschleife abgerufen wird.

Der Inhalt von *scroll_hv* wird jedesmal ausgewertet, wenn der Datenbank-Verarbeitungsschleifenzyklus ausgeführt wird.

$\left[\left\{ \begin{array}{l} \underline{I}NSENSITIVE \\ \underline{S}ENSITIVE \end{array} \right\} \right]$	$\left\{ \begin{array}{l} \underline{C}URRENT \\ \underline{F}IRST \\ \underline{L}AST \\ \underline{P}RIOR \\ \underline{N}EXT \mid N \\ \\ \left\{ \begin{array}{l} \underline{A}BSOLUTE \\ \underline{R}ELATIVE \end{array} \right\} \quad [+ \mid -] \textit{integer} \end{array} \right.$
---	--

scroll_hv - Optionen

Option	Erläuterung
CURRENT	Ruft die aktuelle Reihe (erneut) ab.
FIRST	Ruft die erste Reihe ab.
LAST	Ruft die letzte Reihe ab.
NEXT	Ruft die Reihe nach der aktuellen Reihe ab. Dies ist die Voreinstellung.
PRIOR	Ruft die Reihe vor der aktuellen Reihe ab.
+/- <i>integer</i>	Gilt nur in Verbindung mit ABSOLUTE oder RELATIVE. Gibt die Position der abzurufenden Reihe ABSOLUTE oder RELATIVE an. Geben Sie ein Plus- (+) oder Minus-Zeichen (-) und dahinter eine Ganzzahl ein. Die Voreinstellung ist ein Plus-Zeichen (+).
ABSOLUTE	Gilt nur in Verbindung mit +/- <i>integer</i> . Benutzt <i>integer</i> als die absolute Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.
RELATIVE	Gilt nur in Verbindung mit +/- <i>integer</i> . Benutzt <i>integer</i> als Position <i>relativ</i> zur aktuellen Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.

Bei bestimmten RDBMS-Systemen gelten einige Einschränkungen:

- DB2 unterstützt nicht das Schlüsselwort `CURRENT`.
- In einer `SELECT FOR UPDATE`-Schleife unterstützt DB2 nur `NEXT` als Positionierungsoption.
- MS SQL Server (ODBC-Interface) unterstützt nicht das Schlüsselwort `CURRENT`.
- Adabas D unterstützt keine `RELATIVE`-Positionierung.

GIVING [:] sqlcode

Die Angabe von `GIVING [:] sqlcode` ist optional. Falls angegeben, muss die Natural-Variable `[:] sqlcode` das Format `I4` haben. Die Werte für diese Variable werden von DB2 `SQLCODE` der zugrundeliegenden `FETCH`-Operation zurückgegeben. Dadurch wird es der Anwendung ermöglicht, auf verschiedene, bei geöffnetem "scrollable cursor" vorgefundene Status zu reagieren. Die wichtigsten, von `SQLCODE` angezeigten Status-Codes sind in der folgenden Tabelle aufgeführt

SQLCODE	Erläuterung
0	<code>FETCH</code> -Operation erfolgreich, Daten zurückgegeben, außer bei einem <code>FETCH</code> mit der Option <code>BEFORE</code> oder <code>AFTER</code> .
+100	Reihe nicht gefunden, Cursor noch geöffnet, keine Daten zurückgegeben.
-1	Allgemeiner Fehler bei <code>Fetch</code> -Operation auf eine Reihe.

Wenn Sie `GIVING [:] sqlcode` angeben, muss die Anwendung auf die verschiedenen Status reagieren. Wenn ein `SQLCODE +100` fünfmal hintereinander ohne Terminal I/O eingegeben wird, gibt die NDB-Laufzeit den Natural-Fehler `NAT3296` aus, um Anwendungsschleifen zu verhindern. Die Anwendung kann die Verarbeitungsschleife durch Ausführen eines `ESCAPE`-Statements beenden.

Wenn Sie `GIVING [:] sqlcode` nicht angeben, außer bei `SQLCODE 0` und `SQLCODE +100`, erzeugt jeder `SQLCODE` den Natural-Fehler `NAT3700`, und die Verarbeitungsschleife wird beendet. Mit `SQLCODE +100` (Reihe nicht gefunden) wird die Verarbeitungsschleife beendet.

Siehe auch das Beispielprogramm `DEM2SCR` in der Natural-Systembibliothek `SYSDB2`.

152 UPDATE - SQL

▪ Funktion	1016
▪ Syntax-Beschreibung	1016
▪ Beispiele	1019

Dieses Kapitel behandelt folgende Themen:

Funktion

Das SQL-Statement `UPDATE` dient dazu, Reihen in einer Tabelle zu ändern, ohne einen Cursor zu verwenden („**Searched**“ `UPDATE`), oder Spalten in der Reihe zu ändern, auf die der Cursor zeigt („**Positioned**“ `UPDATE`).

Syntax-Beschreibung

Es sind zwei unterschiedliche Strukturen möglich:

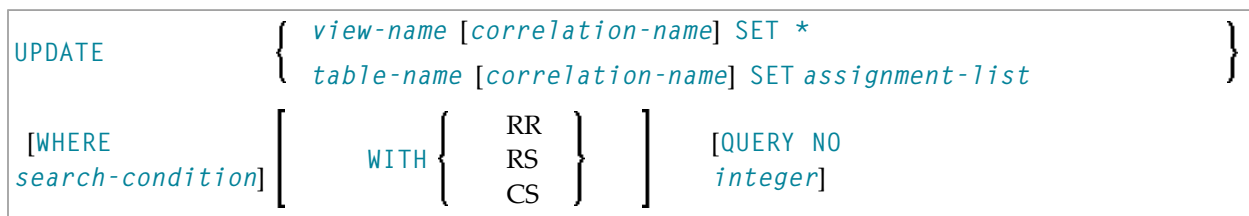
- Syntax 1 — Searched `UPDATE`
- Syntax 2 — Positioned `UPDATE`

Syntax 1 — Searched `UPDATE`

Searched `UPDATE` ist ein eigenständiges Statement, das unabhängig von einem `SELECT`-Statement verwendet werden kann. Mit einem einzigen Statement können Sie keine, eine, mehrere oder alle Reihen einer Tabelle ändern. Welche Reihen geändert werden, bestimmen Sie mit der Suchbedingung (*search-condition*), die auf die Tabelle angewendet wird. Außerdem ist es möglich, einem Tabellen- oder View-Namen einen *correlation-name* zuzuweisen.



Anmerkung: Die Anzahl der Reihen, die mit einem Searched `UPDATE` tatsächlich geändert wurden, kann mit der Systemvariablen `*ROWCOUNT` (siehe *Systemvariablen*-Dokumentation) ermittelt werden.



Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Syntax-Elementbeschreibung – Syntax 1:

<i>view-name</i>	View-Name: <i>view-name</i> ist jeweils der Name eines im DEFINE DATA-Statement definierten Natural-Views. Siehe <i>view-name</i> im Abschnitt <i>Grundlegende Syntaxbestandteile</i> .
SET	SET-Klausel: Wenn sich die Änderungen auf einen View beziehen, müssen Sie in der SET-Klausel einen Stern (*) angeben, da alle Spalten des Views geändert werden müssen. Wenn sich die Änderungen auf eine Tabelle beziehen, können Sie in der SET-Klausel entweder eine <i>assignment-list</i> angeben oder den Namen eines Views, der die zu ändernden Spalten enthält.
<i>assignment-list</i>	Siehe <i>Assignment List</i> weiter unten.
WHERE <i>search-condition</i>	WHERE-Klausel: In der WHERE-Klausel geben Sie eine Suchbedingung (<i>search-condition</i>) an, die bestimmt, welche Reihen geändert werden sollen. Wenn Sie keine WHERE-Klausel angeben, wird die gesamte Tabelle geändert.

assignment-list

$$\left\{ \text{column-name} = \left\{ \text{scalar-expression NULL} \right\} \right\} \dots$$

In einer *assignment-list* können Sie einer oder mehreren Spalten Werte zuweisen. Ein Wert kann entweder eine *scalar-expression* oder NULL sein. Weitere Informationen siehe *Scalar Expressions*.

Wenn Sie NULL zuweisen, bedeutet dies, dass das betreffende Feld keinen Wert enthalten soll (auch nicht den Wert 0 oder Leerzeichen).

SQL Extended Set

Die folgenden Syntax-Elemente gehören zum **SQL Extended Set**:

<i>correlation-name</i>	Das Element <i>correlation-name</i> ist ein Alias-Name für <i>table-name</i> . Weitere Informationen siehe <i>correlation-name</i> (im Abschnitt <i>Grundlegende Syntaxbestandteile</i>).
WITH	WITH Isolation Level-Klausel: Diese Klausel ermöglicht die explizite Angabe des beim Suchen der zu ändernden Reihen benutzten Isolation Level.

	Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
QUERYNO <i>integer</i>	QUERYNO-Klausel: Diese Klausel wird zurzeit nicht unterstützt und wird ignoriert.

Syntax 2 — Positioned UPDATE

Ein Positioned UPDATE bezieht sich auf einen Cursor innerhalb einer Datenbankschleife. Es muss daher dieselbe Tabelle bzw. denselben View referenzieren wie das entsprechende SELECT-Statement, sonst erfolgt eine Fehlermeldung. Ein Positioned UPDATE kann nur bei cursor-orientierter Selektion verwendet werden.

Common Set-Syntax

```
UPDATE { view-name SET *
        view-name SET assignment-list } [WHERE CURRENT OF CURSOR (r)]
```

Extended Set-Syntax

```
UPDATE { view-name SET *
        view-name SET
        assignment-list } [WHERE CURRENT OF CURSOR (r) [FOR ROW { [:]host-variable } OF ROWSET ]]
```

Syntax-Elementbeschreibung – Syntax 2:

<i>view-name</i>	View-Name: Bezieht sich auf den Namen eines im DEFINE DATA-Statement definierten Natural-Views. Weitere Informationen siehe <i>view-name</i> unter <i>Grundlegende Syntaxbestandteile</i> .
SET * SET <i>assignment-list</i>	SET-Klausel: Wenn ein View zum Ändern spezifiziert wurde, muss ein Stern (*) in der SET-Klausel angegeben werden, weil alle Spalten des Views aktualisiert werden müssen. Wurde eine Tabelle zum Ändern spezifiziert, muss die SET-Klausel entweder eine <i>assignment-list</i> oder den Namen des Views enthalten, der die zu ändernden Spalten enthält.
WHERE CURRENT OF CURSOR (<i>r</i>)	Statement-Referenz: Die Notation (<i>r</i>) dient zur Referenzierung des Statements, das zur Selektion der zu ändernden Reihe verwendet wurde. Wird keine Statement-Referenz angegeben, bezieht sich das UPDATE-Statement auf die jeweils innerste aktive Verarbeitungsschleife, mit der ein Datensatz ausgewählt wurde.

FOR ROW ... OF ROWSET	<p>FOR ROW ... OF ROWSET-Klausel:</p> <p>Diese Klausel gehört zum SQL Extended Set.</p> <p>Die optionale <code>FOR ROW ... OF ROWSET</code>-Klausel für <code>Positioned SQL UPDATE</code>-Statements gibt an, welche Reihe des aktuellen Rowset aktualisiert werden muss. Sie sollte nur angegeben werden, wenn das <code>UPDATE</code>-Statement sich auf ein <code>SELECT</code>-Statement bezieht, das die Rowset-Positionierung verwendet und das Spalten-Arrays in der INTO-Klausel hat.</p> <p>Wenn diese Klausel weggelassen wird, werden alle Reihen des aktuellen Rowset durch die Werte in der <i>assignment-list</i> überschrieben.</p> <p>Diese Klausel kann nicht angegeben werden, wenn <code>view-name SET *</code> angegeben wird.</p>
------------------------------	--

Beispiele

- [Beispiel 1 - Searched UPDATE](#)
- [Beispiel 2 - Searched UPDATE mit assignment-list](#)
- [Beispiel 3 - Positioned UPDATE](#)
- [Beispiel 4 - Positioned UPDATE mit assignment-list](#)

Beispiel 1 - Searched UPDATE

```

DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
ASSIGN AGE = 45
ASSIGN NAME = 'SCHMIDT'
UPDATE PERS SET * WHERE NAME = 'SCHMIDT'
...

```

Beispiel 2 - Searched UPDATE mit assignment-list

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE NAME = 'SCHMIDT'
...
```

Beispiel 3 - Positioned UPDATE

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL_PERSONNEL WHERE NAME = 'SCHMIDT'
COMPUTE AGE = AGE + 1
UPDATE PERS SET * WHERE CURRENT OF CURSOR
END-SELECT
...
```

Beispiel 4 - Positioned UPDATE mit assignment-list

```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
2 NAME
2 AGE
...
END-DEFINE
...
SELECT * INTO PERS FROM SQL-PERSONNEL WHERE NAME = 'SCHMIDT'
UPDATE SQL-PERSONNEL SET AGE = AGE + 1 WHERE CURRENT OF CURSOR
END-SELECT
...
```


153

Referenzierte Beispielprogramme

▪ ASSIGN	1022
▪ AT BREAK	1023
▪ AT END OF DATA	1025
▪ AT END OF PAGE	1026
▪ AT START OF DATA	1027
▪ AT TOP OF PAGE	1028
▪ DEFINE SUBROUTINE	1029
▪ FIND	1030
▪ FOR	1032
▪ HISTOGRAM	1033
▪ IF	1034
▪ PERFORM BREAK PROCESSING	1035
▪ READ	1036
▪ REPEAT	1037
▪ SORT	1039
▪ STORE	1040
▪ UPDATE	1042
▪ Beispielprogramme für Systemvariablen	1043

Dieses Kapitel enthält zusätzliche Beispielprogramme, die in der Natural Statements- und in der Systemvariablen-Dokumentation) referenziert werden. Es handelt sich dabei hauptsächlich um Beispiele für den Reporting Mode. Alle diese Beispiele sind in der Library SYSEXSYN enthalten.



Anmerkung: Grundsätzlich sind die in den Statement-Beschreibungen gezeigten Beispielprogramme im Structured Mode geschrieben. Bei Statements, bei denen die Syntax im Reporting Mode stark von der Syntax im Structured Mode abweicht, finden Sie außerdem Verweise auf äquivalente Reporting Mode-Beispiele. Die im folgenden abgebildeten Beispielprogramme sind auch online in Sourcecode-Form verfügbar, und zwar in der Natural-Library SYSEXSYN. Weitere Beispielprogramme zur Benutzung der Natural-Statements sind im *Leitfaden zur Programmierung* dokumentiert. Diese Beispielprogramme stehen online in der Natural-Library SYSEXPG zur Verfügung. Näheres zur Verfügbarkeit dieser Libraries in Ihrem Unternehmen erfahren Sie von Ihrem Natural-Administrator. Die Beispielprogramme verwenden Daten aus der Datei EMPLOYEES (Angestellendaten), die die Software AG zu Demonstrationszwecken ausliefert.

ASSIGN

Das folgende Beispiel wird in der [ASSIGN/COMPUTE](#)-Statement-Beschreibung referenziert.

ASGEX1R - ASSIGN (Reporting Mode)

```

** Beispiel 'ASGEX1R': ASSIGN (reporting mode)
*****
RESET #A (N3)
      #B (A6)
      #C (NO.3)
      #D (NO.5)
      #E (N1.3)
      #F (N5)
      #G (A25)
      #H (A3/1:3)
*
#A = 5                                WRITE NOTITLE '=' #A
#B = 'ABC'                            WRITE '=' #B
#C = .45                               WRITE '=' #C
#D = #E = -0.12345                    WRITE '=' #D / '=' #E
ASSIGN ROUNDED #F = 199.999           WRITE '=' #F
#G = 'HELLO'                          WRITE '=' #G
*
#H (1) = 'UVW'
#H (3) = 'XYZ'                        WRITE '=' #H (1:3)
*
END

```

Ausgabe des Programms AEDEX1R:

```
#A: 5
#B: ABC
#C: .450
#D: -.12345
#E: -0.123
#F: 200
#G: HELLO
#H: UVW XYZ
```

AT BREAK

Die folgenden Beispiele werden in der [AT BREAK](#)-Statement-Beschreibung referenziert.

ATBEX1R - AT BREAK (Reporting Mode)

```
** Beispiel 'ATBEX1R': AT BREAK (reporting mode)
*****
*
LIMIT 10
READ EMPLOYEES BY CITY
  AT BREAK OF CITY DO
    SKIP 1
  DOEND
/*
  DISPLAY NOTITLE CITY (IS=ON) COUNTRY (IS=ON) NAME
LOOP
END
```

Ausgabe des Programms ATBEX1R:

CITY	COUNTRY	NAME
AIKEN	USA	SENKO
AIX EN OTHE	F	GODEFROY
AJACCIO		CANALE
ALBERTSLUND	DK	PLOUG
ALBUQUERQUE	USA	HAMMOND ROLLING

		FREEMAN LINCOLN
ALFRETON	UK	GOLDBERG
ALICANTE	E	GOMEZ

ATBEX5R - AT BREAK-Statement mit mehreren Gruppenwechselebenen (Reporting Mode)

```

** Beispiel 'ATBEX5R': AT BREAK (multiple break levels) (reporting mode)
*****
RESET LEAVE-DUE-L (N4)
*
LIMIT 5
FIND EMPLOYEES WITH CITY = 'PHILADELPHIA' OR = 'PITTSBURGH'
      SORTED BY CITY DEPT
      MOVE LEAVE-DUE TO LEAVE-DUE-L
      DISPLAY CITY (IS=ON) DEPT (IS=ON) NAME LEAVE-DUE-L
      AT BREAK OF DEPT
        WRITE NOTITLE /
          T*DEPT OLD(DEPT) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) /
      AT BREAK OF CITY
        WRITE NOTITLE
          T*CITY OLD(CITY) T*LEAVE-DUE-L SUM(LEAVE-DUE-L) //
LOOP
*
END
    
```

Ausgabe des Programms ATBEX5R:

CITY	DEPARTMENT CODE	NAME	LEAVE-DUE-L
PHILADELPHIA	MGMT30	WOLF-TERROINE MACKARNESS	11 27
	MGMT30		38
	TECH10	BUSH NETTLEFOLDS	39 24
	TECH10		63
PHILADELPHIA			101
PITTSBURGH	MGMT10	FLETCHER	34

MGMT10	34
PITTSBURGH	34

AT END OF DATA

Das folgende Beispiel wird in der [AT END OF DATA](#)-Statement-Beschreibung referenziert.

AEDEX1R - AT END OF DATA (Reporting Mode)

```

** Beispiel 'AEDEX1R': AT END OF DATA (reporting mode)
*****
LIMIT 5
EMP. FIND EMPLOYEES WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
    SALARY (1) CURR-CODE (1)
/*
AT END OF DATA DO
  IF *COUNTER (EMP.) = 0 DO
    WRITE 'NO RECORDS FOUND'
    ESCAPE BOTTOM
  DOEND
  WRITE NOTITLE / 'SALARY STATISTICS:'
    / 7X 'MAXIMUM:' MAX(SALARY(1)) CURR-CODE (1)
    / 7X 'MINIMUM:' MIN(SALARY(1)) CURR-CODE (1)
    / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
DOEND
LOOP
END

```

Ausgabe des Programms AEDEX1R:

PERSONNEL ID	NAME	FIRST-NAME	ANNUAL SALARY	CURRENCY CODE
11100328	BERGHAUS	ROSE	70800	DM
11100329	BARTHEL	PETER	42000	DM
11300313	AECKERLE	SUSANNE	55200	DM
11300316	KANTE	GABRIELE	61200	DM
11500304	KLUGE	ELKE	49200	DM
SALARY STATISTICS:				

```

MAXIMUM:      70800 DM
MINIMUM:      42000 DM
AVERAGE:     55680 DM
    
```

AT END OF PAGE

Das folgende Beispiel wird in der [AT END OF PAGE](#)-Statement-Beschreibung referenziert.

AEPEX1R - AT END OF PAGE (Reporting Mode)

```

** Beispiel 'AEPEX1R': AT END OF PAGE (reporting mode)
*****
FORMAT PS=10
LIMIT 10
READ EMPLOYEES BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
    NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*
  AT END OF PAGE DO
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  DOEND
  /*
LOOP
END
    
```

Ausgabe des Programms AEPEX1R:

NAME	CURRENT POSITION	SALARY	CURRENCY CODE
CREMER	ANALYST	34000	USD
MARKUSH	TRAINEE	22000	USD
GEE	MANAGER	39500	USD
KUNEY	DBA	40200	USD
NEEDHAM	PROGRAMMER	32500	USD
JACKSON	PROGRAMMER	33000	USD
AVERAGE SALARY: ...		33533	USD

AT START OF DATA

Das folgende Beispiel wird in der [AT START OF DATA](#)-Statement-Beschreibung referenziert.

ASDEX1R - AT START OF DATA (Reporting Mode)

```

** Beispiel 'ASDEX1R': AT START OF DATA (reporting mode)
*****
RESET #CITY (A20) #CNTL (A1)
*
REPEAT
  INPUT 'ENTER VALUE FOR CITY' #CITY
  /*
  IF #CITY = ' ' OR= 'END' DO
    STOP
  DOEND
  FIND EMPLOYEES WITH CITY = #CITY
  IF NO RECORDS FOUND DO
    WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
    ESCAPE
  DOEND
  /*
  AT START OF DATA DO
    INPUT (AD=0) 'RECORDS FOUND' *NUMBER //
    'ENTER ''D'' TO DISPLAY RECORDS' #CNTL (AD=A)
    IF #CNTL NE 'D' DO
      ESCAPE BOTTOM
    DOEND
  DOEND
  /*
  DISPLAY NAME FIRST-NAME
  LOOP
LOOP
END

```

Ausgabe des Programms ASDEX1R:

```
ENTER VALUE FOR CITY PARIS
```

Nach Eingabe und Bestätigung des Namens der Stadt:

```
RECORDS FOUND      26
ENTER 'D' TO DISPLAY RECORDS D
```

Nach Eingabe und Bestätigung von D:

NAME	FIRST-NAME
MAIZIERE	ELISABETH
MARX	JEAN-MARIE
REIGNARD	JACQUELINE
RENAUD	MICHEL
REMOUE	GERMAINE
LAVENDA	SALOMON
BROUSSE	GUY
GIORDA	LOUIS
SIECA	FRANCOIS
CENSIER	BERNARD
DUC	JEAN-PAUL
CAHN	RAYMOND
MAZUY	ROBERT
FAURIE	HENRI
VALLY	ALAIN
BRETON	JEAN-MARIE
GIGLEUX	JACQUES
KORAB-BRZOZOWSKI	BOGDAN
XOLIN	CHRISTIAN
LEGRIS	ROGER
VVVV	

AT TOP OF PAGE

Das folgende Beispiel wird in der [AT TOP OF PAGE](#)-Statement-Beschreibung referenziert.

ATPEX1R - AT TOP OF PAGE (Reporting Mode)

```
** Beispiel 'ATPEX1R': AT TOP OF PAGE (reporting mode)
*****
*
FORMAT PS=15
LIMIT 15
*
READ EMPLOYEES BY NAME STARTING FROM 'L'
  DISPLAY 2X NAME 4X FIRST-NAME CITY DEPT
```



```

WRITE TITLE UNDERLINED 'EMPLOYEE REPORT'
WRITE TRAILER '-' (78)
/*
AT TOP OF PAGE DO
  WRITE 'BEGINNING NAME:' NAME
DOEND
/*
AT END OF PAGE DO
  SKIP 1
  WRITE 'ENDING NAME:  ' NAME
DOEND
LOOP
END

```

DEFINE SUBROUTINE

Das folgende Beispiel wird in der `DEFINE SUBROUTINE`-Statement-Beschreibung referenziert.

DSREX1R - DEFINE SUBROUTINE (Reporting Mode)

```

** Beispiel 'DSREX1R': DEFINE SUBROUTINE (reporting mode)
*****
RESET #ARRAY-ALL (A300)
  #X (N2) #Y (N2)
REDEFINE #ARRAY-ALL (#ARRAY (A75/1:4))
  #ARRAY-ALL (#ALINE (A25/1:4,1:3))
*
FORMAT PS=20
LIMIT 5
*
MOVE 1 TO #X #Y
*
FIND EMPLOYEES WITH NAME = 'SMITH'
OBTAIN ADDRESS-LINE (1:2)
/*
MOVE NAME           TO #ALINE (#X,#Y)
MOVE ADDRESS-LINE(1) TO #ALINE (#X+1,#Y)
MOVE ADDRESS-LINE(2) TO #ALINE (#X+2,#Y)
MOVE PHONE          TO #ALINE (#X+3,#Y)
IF #Y = 3 DO
  MOVE 1 TO #Y
  PERFORM PRINT
DOEND
ELSE DO
  ADD 1 TO #Y
DOEND
AT END OF DATA DO
  PERFORM PRINT

```

```
DOEND
LOOP
*
DEFINE SUBROUTINE PRINT
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
RETURN
*
END
```

Ausgabe des Programms AEDEX1R:

```
SMITH          SMITH          SMITH
ENGLANDSVEJ 222  3152 SHETLAND ROAD  14100 ESWORTHY RD.
554349          MILWAUKEE          MONTERREY
                877-4563          994-2260

SMITH          SMITH
5 HAWTHORN     13002 NEW ARDEN COUR
OAK BROOK     SILVER SPRING
150-9351      639-8963
```

FIND

Die folgenden Beispiele werden in der [FIND](#)-Statement-Beschreibung referenziert.

FNDFIR - FIND-Statement mit FIRST-Option (Reporting Mode)

```
** Beispiel 'FNDFIR': FIND FIRST
*****
*
FIND FIRST EMPLOYEES WITH CITY = 'DERBY'
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:' *NUMBER
SKIP 2
WRITE '***FIRST PERSON SELECTED***' //
  'NAME:      ' NAME /
  'DEPARTMENT:' DEPT /
  'JOB TITLE: ' JOB-TITLE
*
END
```

Ausgabe des Programms FNDFIR:

```
TOTAL RECORDS SELECTED:          141
```

```
***FIRST PERSON SELECTED***
```

```
NAME:          DEAKIN
DEPARTMENT:    SALE01
JOB TITLE:     SALES ACCOUNTANT
```

FNDNUM - FIND-Statement mit NUMBER-Option (Reporting Mode)

```
** Beispiel 'FNDNUM': FIND NUMBER
*****
RESET #BIRTH (D)
*
MOVE EDITED '19500101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOYEES WITH CITY = 'MADRID'
          WHERE BIRTH LT #BIRTH
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:          ' *NUMBER
          / 'TOTAL BORN BEFORE 1 JAN 1950: ' *COUNTER
*
END
```

Ausgabe des Programms FNDNUM:

```
TOTAL RECORDS SELECTED:          41
TOTAL BORN BEFORE 1 JAN 1950:    16
```

FNDUNQ - FIND-Statement mit UNIQUE-Option (Reporting Mode)

```
** Beispiel 'FNDUNQ': FIND UNIQUE
*****
RESET #NAME (A20)
*
*
INPUT 'ENTER EMPLOYEE NAME: ' #NAME
IF #NAME = ' '
  STOP
*
FIND UNIQUE EMPLOYEES WITH NAME = #NAME
*
DISPLAY NOTITLE NAME FIRST-NAME JOB-TITLE
*
ON ERROR DO
  WRITE 'NAME EITHER NOT UNIQUE OR DOES NOT EXIST'
```

```
  FETCH 'FNDUNQ'  
DOEND  
*  
END
```

Ausgabe des Programms FNDUNQ:

```
ENTER EMPLOYEE NAME: HEURTEBISE
```

Nach Eingabe und Bestätigung des Namens HEURTEBISE:

NAME	FIRST-NAME	CURRENT POSITION
HEURTEBISE	MICHEL	CONTROLEUR DE GESTION

FOR

Das folgende Beispiel wird in der [FOR-Statement-Beschreibung](#) referenziert.

FOREX1R - FOR (Reporting Mode)

```
** Beispiel 'FOREX1R': FOR (reporting mode)  
*****  
RESET #INDEX (I1)  
      #ROOT (N2.7)  
*  
FOR #INDEX 1 TO 5  
  COMPUTE #ROOT = SQRT (#INDEX)  
  WRITE NOTITLE '=' #INDEX 3X '=' #ROOT  
LOOP  
*  
SKIP 1  
FOR #INDEX 1 TO 5 STEP 2  
  COMPUTE #ROOT = SQRT (#INDEX)  
  WRITE '=' #INDEX 3X '=' #ROOT  
LOOP  
*  
END
```

Ausgabe des Programms FOREX1R:

```
#INDEX: 1 #ROOT: 1.0000000
#INDEX: 2 #ROOT: 1.4142135
#INDEX: 3 #ROOT: 1.7320508
#INDEX: 4 #ROOT: 2.0000000
#INDEX: 5 #ROOT: 2.2360679

#INDEX: 1 #ROOT: 1.0000000
#INDEX: 3 #ROOT: 1.7320508
#INDEX: 5 #ROOT: 2.2360679
```

HISTOGRAM

Das folgende Beispiel wird in der [HISTOGRAM](#)-Statement-Beschreibung referenziert.

HSTEX1R - HISTOGRAM (Reporting Mode)

```
** Beispiel 'HSTEX1R': HISTOGRAM (reporting mode)
*****
*
LIMIT 8
HISTOGRAM EMPLOYEES CITY STARTING FROM 'M'
  DISPLAY NOTITLE CITY
    'NUMBER OF/PERSONS' *NUMBER *COUNTER
LOOP
*
END
```

Ausgabe des Programms HSTEX1R:

CITY	NUMBER OF PERSONS	CNT
MADISON	3	1
MADRID	41	2
MAILLY LE CAMP	1	3
MAMERS	1	4
MANSFIELD	4	5
MARSEILLE	2	6
MATLOCK	1	7
MELBOURNE	2	8

IF

Das folgende Beispiel wird in der [IF-Statement-Beschreibung](#) referenziert.

IFEX1R - IF (Reporting Mode)

```

** Beispiel 'IFEX1R': IF (reporting mode)
*****
RESET #BIRTH (D)
*
MOVE EDITED '19450101' TO #BIRTH (EM=YYYYMMDD)
SUSPEND IDENTICAL SUPPRESS
LIMIT 20
*
FND. FIND EMPLOYEES WITH CITY = 'FRANKFURT'
      SORTED BY NAME BIRTH
  IF SALARY (1) LT 40000
    WRITE NOTITLE '*****' NAME 30X 'SALARY LT 40000'
  ELSE DO
    IF BIRTH GT #BIRTH DO
      FIND VEHICLES WITH PERSONNEL-ID = PERSONNEL-ID (FND.)
      DISPLAY (IS=ON) NAME BIRTH (EM=YYYY-MM-DD)
      SALARY (1) MAKE (AL=8)
    LOOP
  DOEND
DOEND
LOOP
END

```

Ausgabe des Programms IFEX1R:

NAME	DATE OF BIRTH	ANNUAL SALARY	MAKE
BAECKER	1956-01-05	74400	BMW
***** BECKER			SALARY LT 40000
BLOEMER	1979-11-07	45200	FIAT
FALTER	1954-05-23	70800	FORD
***** FALTER			SALARY LT 40000
***** GROTHE			SALARY LT 40000
***** HEILBROCK			SALARY LT 40000
***** HESCHMANN			SALARY LT 40000
HUCH	1952-09-12	67200	MERCEDES
***** KICKSTEIN			SALARY LT 40000

```

***** KLEENE                                SALARY LT 40000
***** KRAMER                                SALARY LT 40000

```

PERFORM BREAK PROCESSING

Das folgende Beispiel wird in der `PERFORM BREAK PROCESSING`-Statement-Beschreibung referenziert.

PBPEX1R - PERFORM BREAK PROCESSING (Reporting Mode)

```

** Beispiel 'PBPEX1R': PERFORM BREAK PROCESSING (reporting mode)
*****
RESET #LINE (N2) #INDEX (N2)
*
MOVE 1 TO #LINE
FOR #INDEX 1 TO 18
  PERFORM BREAK PROCESSING
  /*
  AT BREAK OF #INDEX /1/ DO
    WRITE NOTITLE / 'PLEASE COMPLETE LINES 1-9 ABOVE' /
    MOVE 1 TO #LINE
  DOEND
  /*
  WRITE NOTITLE '_' (64) '=' #LINE
  ADD 1 TO #LINE
LOOP
END

```

Ausgabe des Programms PBPEX1R:

```

_____ #LINE: 1
_____ #LINE: 2
_____ #LINE: 3
_____ #LINE: 4
_____ #LINE: 5
_____ #LINE: 6
_____ #LINE: 7
_____ #LINE: 8
_____ #LINE: 9

PLEASE COMPLETE LINES 1-9 ABOVE

_____ #LINE: 1
_____ #LINE: 2
_____ #LINE: 3
_____ #LINE: 4
_____ #LINE: 5

```

```
_____|#LINE: 6
_____|#LINE: 7
_____|#LINE: 8
_____|#LINE: 9

PLEASE COMPLETE LINES 1-9 ABOVE
```

READ

Das folgende Beispiel wird in der [READ](#)-Statement-Beschreibung referenziert.

REAEX1R - READ (Reporting Mode)

```
** Beispiel 'REAEX1R': READ (reporting mode)
*****
LIMIT 3
*
WRITE 'READ IN PHYSICAL SEQUENCE'
READ EMPLOYEES IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOYEES BY ISN STARTING FROM 1 ENDING AT 3
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOYEES BY NAME
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'' '
READ EMPLOYEES BY NAME STARTING FROM 'M'
  DISPLAY PERSONNEL-ID NAME *ISN *COUNTER
LOOP
*
END
```

Ausgabe des Programms REAEX1R:

PERSONNEL ID	NAME	ISN	CNT

READ IN PHYSICAL SEQUENCE			
50005800	ADAM	1	1
50005600	MORENO	2	2
50005500	BLOND	3	3
READ IN ISN SEQUENCE			
50005800	ADAM	1	1
50005600	MORENO	2	2
50005500	BLOND	3	3
READ IN NAME SEQUENCE			
60008339	ABELLAN	478	1
30000231	ACHIESON	878	2
50005800	ADAM	1	3
READ IN NAME SEQUENCE STARTING FROM 'M'			
30008125	MACDONALD	923	1
20028700	MACKARNESS	765	2
40000045	MADSEN	508	3

REPEAT

Die folgenden Beispiele werden in der [REPEAT](#)-Statement-Beschreibung referenziert.

RPTEX1R - REPEAT (Reporting Mode)

```

** Beispiel 'RPTEX1R': REPEAT (reporting mode)
*****
RESET #PERS-NR (A8)
*
REPEAT
  INPUT 'ENTER A PERSONNEL NUMBER:' #PERS-NR
  IF #PERS-NR = ' '
    ESCAPE BOTTOM
  FIND EMPLOYEES WITH PERSONNEL-ID = #PERS-NR
  IF NO RECORD FOUND
    REINPUT 'NO RECORD FOUND'
  DISPLAY NOTITLE NAME
  LOOP
LOOP
*
END

```

Ausgabe des Programms RPTX1R:

```
ENTER A PERSONNEL NUMBER:
```

RPTX2R - REPEAT mit WHILE- und UNTIL-Option (Reporting Mode)

```
** Beispiel 'RPTX2R': REPEAT (with WHILE and UNTIL option)
*****
RESET #X (I1) #Y (I1)
*
*
REPEAT WHILE #X <= 5
  ADD 1 TO #X
  WRITE NOTITLE '=' #X
LOOP
*
SKIP 3
REPEAT
  ADD 1 TO #Y
  WRITE '=' #Y
  UNTIL #Y = 6
LOOP
*
END
```

Ausgabe des Programms RPTX2R:

```
#X: 1
#X: 2
#X: 3
#X: 4
#X: 5
#X: 6

#Y: 1
#Y: 2
#Y: 3
#Y: 4
#Y: 5
#Y: 6
```

SORT

Das folgende Beispiel wird in der [SORT-Statement-Beschreibung](#) referenziert.

SRTEX1R - SORT (Reporting Mode)

```

** Beispiel 'SRTEX1R': SORT (reporting mode)
*****
RESET #AVG (P11) #TOTAL-TOTAL (P11) #TOTAL-SALARY (P11)
      #AVER-PERCENT (N3.2)
*
LIMIT 3
FIND EMPLOYEES WITH CITY = 'BOSTON'
  OBTAIN SALARY(1:2)
  COMPUTE #TOTAL-SALARY = SALARY (1) + SALARY (2)
  ACCEPT IF #TOTAL-SALARY GT 0
  /*
  SORT BY PERSONNEL-ID USING #TOTAL-SALARY SALARY(*) CURR-CODE
      GIVE AVER(#TOTAL-SALARY)
  /*
  AT START OF DATA DO
    WRITE NOTITLE '*' (40)
      'AVG CUMULATIVE SALARY:' *AVER (#TOTAL-SALARY) /
    MOVE *AVER (#TOTAL-SALARY) TO #AVG
  DOEND
  COMPUTE ROUNDED #AVER-PERCENT = #TOTAL-SALARY / #AVG * 100
  ADD #TOTAL-SALARY TO #TOTAL-TOTAL
  /*
  DISPLAY NOTITLE PERSONNEL-ID SALARY (1) SALARY (2)
      #TOTAL-SALARY CURR-CODE (1)
      'PERCENT/OF/AVER' #AVER-PERCENT
  AT END OF DATA
    WRITE / '*' (40) 'TOTAL SALARIES PAID: ' #TOTAL-TOTAL
LOOP
*
END

```

Ausgabe des Programms SRTEX1R:

PERSONNEL ID	ANNUAL SALARY	ANNUAL SALARY	#TOTAL-SALARY	CURRENCY CODE	PERCENT OF AVER
***** AVG CUMULATIVE SALARY:					44633
20000100	31000	29400	60400	USD	135.30

20019200	18000	17100	35100	USD	78.60
20020400	20000	18400	38400	USD	86.00
***** TOTAL SALARIES PAID:					133900

STORE

Das folgende Beispiel wird in der [STORE-Statement-Beschreibung](#) referenziert.

STOEX1R - STORE (Reporting Mode)

```

** Beispiel 'STOEX1R': STORE (reporting mode)
**
** CAUTION: Executing this example will modify the database records!
*****
RESET #PERSONNEL-ID (A8)
      #NAME          (A20)
      #FIRST-NAME    (A15)
      #BIRTH-D       (D)
      #MAR-STAT      (A1)
      #BIRTH         (A8)
      #CITY          (A20)
      #COUNTRY       (A3)
      #CONF          (A1)
*
REPEAT
  INPUT 'ENTER A PERSONNEL ID AND NAME (OR ''END'' TO END)' //
        'PERSONNEL-ID : ' #PERSONNEL-ID //
        'NAME          : ' #NAME /
        'FIRST-NAME    : ' #FIRST-NAME
  /*
  /* VALIDATE ENTERED DATA
  /*
  IF #PERSONNEL-ID = 'END' OR #NAME = 'END'
    STOP
  IF #NAME = ' '
    REINPUT WITH TEXT 'ENTER A LAST-NAME' MARK 2 AND SOUND ALARM
  IF #FIRST-NAME = ' '
    REINPUT WITH TEXT 'ENTER A FIRST-NAME' MARK 3 AND SOUND ALARM
  /*
  /* ENSURE PERSON IS NOT ALREADY ON FILE
  /*
  FIND NUMBER EMPLOYEES WITH PERSONNEL-ID = #PERSONNEL-ID
  IF *NUMBER > 0
    REINPUT 'PERSON WITH SAME PERSONNEL-ID ALREADY EXISTS'
          MARK 1 AND SOUND ALARM
  MOVE 'N' TO #CONF
  /*

```

```

/* GET FURTHER INFORMATION
/*
INPUT
  'ADDITIONAL PERSONNEL DATA'          ////
  'PERSONNEL-ID           :' #PERSONNEL-ID (AD=IO) /
  'NAME                   :' #NAME         (AD=IO) /
  'FIRST-NAME             :' #FIRST-NAME  (AD=IO) ///
  'MARITAL STATUS        :' #MAR-STAT     /
  'DATE OF BIRTH (YYYYMMDD) :' #BIRTH     /
  'CITY                   :' #CITY        /
  'COUNTRY (3 CHARACTERS) :' #COUNTRY     //
  'ADD THIS RECORD (Y/N)  :' #CONF        (AD=M)
/*
/* ENSURE REQUIRED FIELDS CONTAIN VALID DATA
/*
IF NOT (#MAR-STAT = 'S' OR = 'M' OR = 'D' OR = 'W')
  REINPUT TEXT 'ENTER VALID MARITAL STATUS S=SINGLE ' -
              'M=MARRIED D=DIVORCED W=WIDOWED' MARK 1
IF NOT (#BIRTH = MASK(YYYYMMDD) AND #BIRTH = MASK(1582-2699))
  REINPUT TEXT 'ENTER CORRECT DATE' MARK 2
IF #CITY = ' '
  REINPUT TEXT 'ENTER A CITY NAME' MARK 3
IF #COUNTRY = ' '
  REINPUT TEXT 'ENTER A COUNTRY CODE' MARK 4
IF NOT (#CONF = 'N' OR= 'Y')
  REINPUT TEXT 'ENTER Y (YES) OR N (NO)' MARK 5
IF #CONF = 'N'
  ESCAPE TOP
/*
/* ADD THE RECORD
/*
MOVE EDITED #BIRTH TO #BIRTH-D (EM=YYYYMMDD)
/*
STORE RECORD IN EMPLOYEES
  WITH PERSONNEL-ID = #PERSONNEL-ID
      NAME           = #NAME
      FIRST-NAME     = #FIRST-NAME
      MAR-STAT       = #MAR-STAT
      BIRTH          = #BIRTH-D
      CITY           = #CITY
      COUNTRY        = #COUNTRY
END OF TRANSACTION
/*
WRITE NOTITLE 'RECORD HAS BEEN ADDED'
/*
LOOP
END

```

UPDATE

Das folgende Beispiel wird in der [UPDATE](#)-Statement-Beschreibung referenziert.

UPDEX1R - UPDATE (Reporting Mode)

```
** Beispiel 'UPDEX1R': UPDATE (reporting mode)
**
** CAUTION: Executing this example will modify the database records!
*****
RESET #NAME (A20)
*
INPUT 'ENTER A NAME:' #NAME (AD=M)
IF #NAME = ' '
  STOP
*
FIND EMPLOYEES WITH NAME = #NAME
  IF NO RECORDS FOUND
    REINPUT WITH 'NO RECORDS FOUND' MARK 1
  /*
  INPUT 'NAME:      ' NAME (AD=0) /
        'FIRST NAME:' FIRST-NAME (AD=M) /
        'CITY:      ' CITY (AD=M)
  /*
  UPDATE USING SAME RECORD
  /*
  END TRANSACTION
  /*
LOOP
*
END
```

Ausgabe des Programms UPDEX1R:

```
ENTER A NAME:
```

Beispielprogramme für Systemvariablen

Die folgenden Beispiele werden in der *OCCURRENCE-Systemvariablen-Beschreibung referenziert:

OCC1P - Systemvariable *OCCURRENCE

```
** Beispiel 'OCC1P': *OCCURRENCE
*****
DEFINE DATA LOCAL
1 #N1 (N7/1:10)
1 #N2 (N7/1:10,1:10)
1 #N3 (N7/1:10,1:10,1:10)
END-DEFINE
*
CALLNAT 'OCC1N' #N1(*) #N2(1:2,1:4) #N3(1:6,1:7,1:8)
*
END
```

Vom Programm OCC1P aufgerufenes Subprogramm OCC1N:

```
** Beispiel 'OCC1N': *OCCURRENCE (called by OCC1P)
*****
DEFINE DATA
PARAMETER
1 PARM1 (N7/1:V)
1 PARM2 (N7/1:V,1:V)
1 PARM3 (N7/1:V,1:V,1:V)
LOCAL
1 #OCC2 (I4/1:2)
1 #OCC3 (I4/1:3)
1 #OCC1 (I4)
END-DEFINE
*
MOVE *OCC(PARM1) TO #OCC1
MOVE *OCC(PARM2,*) TO #OCC2(*)
MOVE *OCC(PARM3,*) TO #OCC3(*)
*
DISPLAY #OCC1 #OCC2(*) #OCC3(*)
DISPLAY *OCC(PARM1,*) *OCC(PARM2,*) *OCC(PARM3,*)
*
NEWPAGE
*
WRITE NOHDR
      'Occurrences of 1. parameter:' *OCC(PARM1)
      / 'Occurrences of 1. parameter:' *OCC(PARM1,1)
      / 'Occurrences of 1. parameter:' *OCC(PARM1,*)
```

```

/ 'Occurrences of 2. parameter:' *OCC(PARM2,1) *OCC(PARM2,2)
/ 'Occurrences of 2. parameter:' *OCC(PARM2,*)
/ 'Occurrences of 3. parameter:' *OCC(PARM3,1) *OCC(PARM3,2)
                                *OCC(PARM3,3)
/ 'Occurrences of 3. parameter:' *OCC(PARM3,*)
*
END

```

Ausgabe des Programms OCC1P - Seite 1:

Page	1	05-01-18 10:21:30		
	#OCC1	#OCC2	#OCC3	
	10	2	6	
		4	7	
			8	
	10	2	6	
		4	7	
			8	

Ausgabe des Programms OCC1P - Seite 2:

Page	2	05-01-18 10:21:30		
Occurrences of 1. parameter:	10			
Occurrences of 1. parameter:	10			
Occurrences of 1. parameter:	10			
Occurrences of 2. parameter:	2	4		
Occurrences of 2. parameter:	2	4		
Occurrences of 3. parameter:	6	7	8	
Occurrences of 3. parameter:	6	7	8	

OCC2P - System Variable *OCCURRENCE

```

** Beispiel 'OCC2P': *OCCURRENCE
*****
DEFINE DATA LOCAL
1 #N (N7/1:10)
1 #I (I4)
END-DEFINE
*
FOR #I=1 TO 10
  MOVE #I TO #N(#I)
END-FOR
*
WRITE 'Passing occurrences 1:5'

```



```
CALLNAT 'OCC2N' #N(1:5)
*
WRITE 'Passing occurrences 5:10'
CALLNAT 'OCC2N' #N(5:10)
*
END
```

Vom Programm OCC2P aufgerufenes Subprogramm OCC2N:

```
** Beispiel 'OCC2N': *OCCURRENCE (called by OCC2P)
*****
DEFINE DATA
PARAMETER
1 #ARR (N7/1:V)
LOCAL
1 I (N7)
END-DEFINE
*
FOR I=1 TO *OCC(#ARR)
  DISPLAY #ARR(I)
END-FOR
*
END
```

Ausgabe des Programms OCC2P:

```
Page      1                                05-01-18  10:33:03

Passing occurrences 1:5
  1
  2
  3
  4
  5
Passing occurrences 5:10
  5
  6
  7
  8
  9
 10
```

