Natural for z/OS

Statements

Version 9.2.5

Oktober 2026

ADABAS & NATURAL

This document applies to Natural for z/OS Version 9.2.5 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1979-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Document ID: NATMF-NNATSTATEMENTS-925-20261013DE

Inhaltsverzeichnis

Vorwort xxi
1 Über diese Dokumentation 1
Dokumentationskonventionen 2
Online-Informationen und Support 2
Datenschutz 3
I 5
2 Syntax-Symbole und Operandentabellen 7
Syntax-Symbole 8
Operandentabelle 10
3 Statements nach Funktionen 13
Datenbankzugriffe und Datenbankänderungen 14
Arithmetische Funktionen und Datenzuweisungen 16
Schleifenverarbeitung 16
Erstellen von Ausgabe-Reports 17
Bildschirmgenerierung für interaktive Verarbeitung 17
Verarbeitung logischer Bedingungen 18
Aufrufen von Programmen und Subprogrammen 18
Functions 19
Beenden von Programmen und Sessions 19
Verarbeitung von Arbeitsdateien/PC-Dateien 19
Komponentenbasierte Programmierung 20
Speicherverwaltung für dynamische Variablen/X-Arrays 20
Natural Remote Procedure Call 20
Internet und Parsing 21
Verschiedene Statements 21
Reporting Mode-Statements 21
Statements für Predict Case und Entire DB 23
II 25
4 ACCEPT/REJECT 27
Funktion ACCEPT/REJECT 28
Syntax-Beschreibung ACCEPT/REJECT 28
Verarbeitung mehrerer ACCEPT/REJECT-Statements 29
Limit-Notation 30
Hold-Status 30
Beispiele ACCEPT/REJECT 30
5 ADD 33
Funktion ADD 34
Syntax 1 - ADD-Statement ohne GIVING-Klausel 34
Syntax 2 - ADD-Statement mit GIVING-Klausel 35
Beispiel für ADD-Statement 36
6 ASSIGN 39
7 AT BREAK 41
Funktion AT BREAK 42

Syntax-Beschreibung AT BREAK 43
Gruppenwechsel auf mehreren Ebenen 45
Beispiele AT BREAK 46
8 AT END OF DATA 51
Funktion AT END OF DATA 52
Einschränkungen 53
Syntax-Beschreibung AT END OF DATA 53
Beispiel für AT END OF DATA-Statement 54
9 AT END OF PAGE 57
Funktion AT END OF PAGE 58
Syntax-Beschreibung AT END OF PAGE 60
Beispiele AT END OF PAGE 61
10 AT START OF DATA 65
Funktion AT START OF DATA 66
Syntax-Beschreibung AT START OF DATA 67
Beispiel für AT START OF DATA-Statement 68
11 AT TOP OF PAGE 71
Funktion AT TOP OF PAGE 72
Einschränkung 73
Syntax-Beschreibung AT TOP OF PAGE 73
Beispiel für AT TOP OF PAGE-Statement 74
12 BACKOUT TRANSACTION 77
Funktion BACKOUT TRANSACTION 78
Einschränkung 79
Datenbank-spezifische Anmerkungen 79
Beispiel für BACKOUT TRANSACTION-Statement 79
13 BEFORE BREAK PROCESSING 81
Funktion BEFORE BREAK PROCESSING 82
Einschränkungen 83
Syntax-Beschreibung BEFORE BREAK PROCESSING 83
Beispiel für BEFORE BREAK PROCESSING-Statement 84
14 CALL 87
Funktion CALL 88
Syntax-Beschreibung CALL 88
Return Code 89
Registerbelegung 90
Speicherplatzausrichtung 91
Adabas-Aufrufe 91
Direktes/Dynamisches Laden 91
Linkage-Konventionen 93
Programm-Eigenschaften 95
Aufrufen eines PL/I-Programms 95
Aufruf eines C-Programms 97
INTERFACE4 101
15 CALL FILE 115

Funktion CALL FILE ... 116
Einschränkung ... 116
Syntax-Beschreibung CALL FILE ... 117
Beispiel für CALL FILE-Statement ... 118
16 CALL LOOP ... 121
Funktion CALL LOOP ... 122
Einschränkung ... 122
Syntax-Beschreibung CALL LOOP ... 123
Beispiel für CALL LOOP-Statement ... 124
17 CALLDBPROC (SQL) ... 125
Funktion CALLDBPROC (SQL) ... 126
Einschränkung ... 127
Syntax-Beschreibung CALLDBPROC (SQL) ... 127
Beispiel CALLDBPROC (SQL) ... 129
18 CALLNAT ... 131
Funktion CALLNAT ... 132
Syntax-Beschreibung CALLNAT ... 133
Übertragung von Parametern mit dynamischen Variablen ... 135
Beispiele CALLNAT ... 136
19 CLOSE CONVERSATION ... 139
Funktion CLOSE CONVERSATION ... 140
Syntax-Beschreibung CLOSE CONVERSATION ... 140
Weitere Informationen und Beispiele zu CLOSE CONVERSATION ... 141
III ... 143
20 CLOSE PC FILE ... 145
Funktion CLOSE PC FILE ... 146
Syntax-Beschreibung CLOSE PC FILE ... 146
Beispiel für CLOSE PC FILE-Statement ... 147
21 CLOSE PRINTER ... 149
Funktion CLOSE PRINTER ... 150
Syntax-Beschreibung CLOSE PRINTER ... 150
Beispiel für CLOSE PRINTER-Statement ... 151
22 CLOSE WORK FILE ... 153
Funktion CLOSE WORK FILE ... 154
Syntax-Beschreibung CLOSE WORK FILE ... 154
Beispiel für CLOSE WORK FILE-Statement ... 155
23 COMMIT (SQL) ... 157
Funktion COMMIT (SQL) ... 158
Hinweis für Nicht-Natural-Programme ... 158
Beispiel COMMIT (SQL) ... 158
24 COMPOSE ... 159
Funktion COMPOSE ... 160
Syntax-Beschreibung COMPOSE ... 161
Der Formatiervorgang ... 175
Verarbeitung im Dialog-Modus ... 176

Eingabe-/Ausgabe-Verarbeitung durch Nicht-Natural-Programme 178
Beispiele COMPOSE 179
25 COMPRESS 189
Funktion COMPRESS 190
Syntax-Beschreibung COMPRESS 190
Verarbeitung COMPRESS 196
Beispiele COMPRESS 196
26 COMPUTE 201
Funktion COMPUTE 202
Syntax-Beschreibung COMPUTE 204
Ergebnisgenauigkeit einer Division 207
Beispiele COMPUTE 207
27 CREATE OBJECT 211
Funktion CREATE OBJECT 212
Syntax-Beschreibung CREATE OBJECT 212
28 DECIDE FOR 215
Funktion DECIDE FOR 216
Syntax-Beschreibung DECIDE FOR 216
Beispiele DECIDE FOR 217
29 DECIDE ON 221
Funktion DECIDE ON 222
Syntax-Beschreibung DECIDE ON 223
Beispiele DECIDE ON 225
30 DEFINE CLASS 227
Funktion DEFINE CLASS 228
Syntax-Beschreibung DEFINE CLASS 228
IV DEFINE DATA 231
31 Funktion und generelle Syntaxregeln 233
Funktion DEFINE DATA 234
Generelle Syntaxregeln DEFINE DATA 234
Programmiermodi DEFINE DATA 234
32 Definition von Global Data 237
Funktion DEFINE DATA GLOBAL 238
Syntax-Beschreibung DEFINE DATA GLOBAL 238
33 Definition von Parameter Data 241
Funktion DEFINE DATA PARAMETER 242
Einschränkungen DEFINE DATA PARAMETER 242
Syntax-Beschreibung DEFINE DATA PARAMETER 242
34 Definition von Local Data 247
Funktion DEFINE DATA LOCAL 248
Einschränkung DEFINE DATA LOCAL 248
Syntax-Beschreibung DEFINE DATA LOCAL 248
35 Definition von anwendungsunabhängigen Variablen 253
Funktion DEFINE DATA INDEPENDENT 254
Syntax-Beschreibung DEFINE DATA INDEPENDENT 254

36 Definition von Kontext-Variablen für den Natural RPC 257
Funktion DEFINE DATA CONTEXT 258
Einschränkungen DEFINE DATA CONTEXT 259
Syntax-Beschreibung DEFINE DATA CONTEXT 259
37 Definition von NaturalX-Objekten 261
Funktion DEFINE DATA OBJECT 262
Syntax-Beschreibung DEFINE DATA OBJECT 262
38 Definition von Variablen 265
Syntax-Beschreibung variable-definition 266
39 View-Definition 269
Syntax-Beschreibung view-definition 270
40 Redefinition 275
Einschränkungen redefinition 276
Syntax-Beschreibung redefinition 276
41 Definition von Array-Dimensionen 279
Syntax-Beschreibung array-dimension-definition 280
42 Definition eines Ausgangswerts 283
Einschränkung init-definition 284
Syntax-Beschreibung init-definition 284
43 Ausgangswerte/Konstanten-Werte für ein Array 287
Einschränkung array-init-definition 288
Syntax-Beschreibung array-init-definition 289
44 Parameter EM, HD, PM für Feld/Variable 293
Syntax-Beschreibung emhdpm 294
45 Beispiele für die Benutzung des DEFINE DATA-Statements 295
Beispiel 1 - DEFINE DATA LOCAL (Lokale Daten-Definition) 296
Beispiel 2 - DEFINE DATA LOCAL (Array-Definition/Initialisierung) 296
Beispiel 3 - DEFINE DATA (View-Definition, Array-Redefinition) 300
Beispiel 4 - DEFINE DATA (Global, Parameter und Local Data Areas) 301
Beispiel 5 - DEFINE DATA (Initialisierung) 302
Beispiel 6 - DEFINE DATA (Variables Array mit (1:V)) 302
V 305
46 DEFINE FUNCTION 307
Funktion DEFINE FUNCTION 308
Syntax-Beschreibung DEFINE FUNCTION 308
Beispiele DEFINE FUNCTION 312
47 DEFINE PRINTER 315
Funktion DEFINE PRINTER 316
Syntax-Beschreibung DEFINE PRINTER 317
Druckername unter z/OS Batch, TSO und Server 320
Druckername unter CICS 324
Druckername unter Com-plete 324
Druckername unter Com-plete/SMARTS 325
Druckernamen unter Natural Advanced Facilities 325

Druckername für zusätzliche Reports und Remote-Destinationen (Ausgabeziele) ... 325
Beispiele DEFINE PRINTER ... 326
48 DEFINE PROTOTYPE ... 329
Funktion DEFINE PROTOTYPE ... 330
Syntax-Beschreibung DEFINE PROTOTYPE ... 331
Beispiele DEFINE PROTOTYPE ... 334
49 DEFINE SUBROUTINE ... 337
Funktion DEFINE SUBROUTINE ... 338
Syntax-Beschreibung DEFINE SUBROUTINE ... 339
Einschränkungen DEFINE SUBROUTINE ... 340
Beispiele DEFINE SUBROUTINE ... 341
50 DEFINE WINDOW ... 347
Funktion DEFINE WINDOW ... 348
Syntax-Beschreibung DEFINE WINDOW ... 349
Schutz von Eingabefeldern in einem Fenster ... 353
Aufrufen unterschiedlicher Fenster ... 353
Beispiel DEFINE WINDOW ... 354
51 DEFINE WORK FILE ... 357
Funktion DEFINE WORK FILE ... 358
Syntax-Beschreibung DEFINE WORK FILE ... 358
Arbeitsdateiname unter z/OS Batch, TSO und Server ... 360
Arbeitsdateiname unter CICS ... 363
Arbeitsdateiname unter Com-plete/SMARTS ... 364
VI ... 365
52 DELETE ... 367
Funktion DELETE ... 368
Einschränkung DELETE ... 368
Syntax-Beschreibung DELETE ... 368
Datenbank-spezifische Anmerkungen DELETE ... 369
Beispiele DELETE ... 369
53 DELETE (SQL) ... 371
Funktion DELETE (SQL) ... 372
Syntax 1 - Searched DELETE ... 372
Syntax 2 - Positioned DELETE ... 374
54 DISPLAY ... 377
Funktion DISPLAY ... 378
Syntax-Beschreibung DISPLAY ... 378
Standardwerte DISPLAY ... 391
Beispiele DISPLAY ... 392
55 DIVIDE ... 401
Funktion DIVIDE ... 402
Syntax 1 - DIVIDE-Statement ohne GIVING-Klausel ... 402
Syntax 2 - DIVIDE-Statement mit GIVING-Klausel ... 403
Syntax 3 - DIVIDE-Statement mit REMAINDER-Klausel ... 404

Beispiel für DIVIDE-Statement ... 406
56 DO/DOEND ... 409
Funktion DO/DOEND ... 410
Einschränkungen DO/DOEND ... 410
Beispiel DO/DOEND ... 411
57 DOWNLOAD PC FILE ... 413
Funktion DOWNLOAD PC FILE ... 414
Syntax-Beschreibung DOWNLOAD PC FILE ... 414
Externe Darstellung der Felder ... 416
Beispiele DOWNLOAD PC FILE ... 417
58 EJECT ... 421
Funktion EJECT ... 422
Syntax-Beschreibung EJECT ... 422
Verarbeitung ... 424
Beispiel EJECT ... 424
59 END ... 427
Funktion END ... 428
Syntax-Beschreibung END ... 428
Beispiele END ... 429
60 END TRANSACTION ... 431
Funktion END TRANSACTION ... 432
Einschränkung END TRANSACTION ... 433
Syntax-Beschreibung END TRANSACTION ... 433
Betroffene Datenbanken ... 433
Datenbank-spezifische Anmerkungen ... 434
Beispiele END TRANSACTION ... 434
61 ESCAPE ... 437
Funktion ESCAPE ... 438
Syntax-Beschreibung ESCAPE ... 439
Beispiel für ESCAPE-Statement ... 441
62 EXAMINE ... 443
Syntax 1 - EXAMINE ... 444
Syntax 2 - EXAMINE TRANSLATE ... 453
Syntax 3 - EXAMINE für Unicode-Grapheme ... 455
Beispiele EXAMINE ... 457
63 EXPAND ... 467
Funktion EXPAND ... 468
Syntax-Beschreibung EXPAND ... 468
VII ... 473
64 FETCH ... 475
Funktion FETCH ... 476
Syntax-Beschreibung FETCH ... 477
Beispiel für FETCH-Statement ... 478
65 FIND ... 481
Funktion FIND ... 482

Einschränkungen FIND ... 483
Syntax 1 - FIND-Statement mit Verarbeitungsschleife ... 484
Syntax 2 - FIND-Statement ohne Verarbeitungsschleife ... 484
Syntax-Beschreibung FIND ... 485
Beispiele FIND ... 510
66 FOR ... 523
Funktion FOR ... 524
Syntax-Beschreibung FOR ... 525
Beispiel für FOR-Statement ... 526
67 FORMAT ... 529
Funktion FORMAT ... 530
Syntax-Beschreibung FORMAT ... 531
Parameter beim FORMAT-Statement ... 532
Beispiel für FORMAT-Statement ... 533
68 GET ... 535
Funktion GET ... 536
Einschränkungen GET ... 537
Syntax-Beschreibung GET ... 537
Beispiel für GET-Statement ... 538
69 GET SAME ... 541
Funktion GET SAME ... 542
Einschränkungen GET SAME ... 542
Syntax-Beschreibung GET SAME ... 543
Beispiel für GET SAME-Statement ... 543
70 GET TRANSACTION DATA ... 545
Funktion GET TRANSACTION DATA ... 546
Einschränkung GET TRANSACTION DATA ... 547
Syntax-Beschreibung GET TRANSACTION DATA ... 547
Beispiel für GET TRANSACTION DATA-Statement ... 547
71 HISTOGRAM ... 549
Funktion HISTOGRAM ... 550
Einschränkungen HISTOGRAM ... 551
Syntax-Beschreibung HISTOGRAM ... 551
Bei HISTOGRAM verfügbare Systemvariablen ... 557
Beispiele HISTOGRAM ... 557
72 IF ... 563
Funktion IF ... 564
Syntax-Beschreibung IF ... 564
Beispiel für IF-Statement ... 565
73 IF SELECTION ... 567
Funktion IF SELECTION ... 568
Syntax-Beschreibung IF SELECTION ... 568
Beispiel für IF SELECTION-Statement ... 570
74 IGNORE ... 571
Funktion IGNORE ... 572

Beispiel für IGNORE-Statement ... 572
75 INCLUDE ... 573
Funktion INCLUDE ... 574
Syntax-Beschreibung INCLUDE ... 574
Beispiele INCLUDE ... 576
VIII INPUT ... 581
76 INPUT-Syntax 1 - Dynamisch generierter Eingabeschirm ... 589
INPUT Syntax 1 - Beschreibung ... 590
Beispiele - Verwendung von Syntax 1 ... 600
77 INPUT-Syntax 2 - Verwendung einer vordefinierten Eingabemaske ... 603
INPUT USING MAP ohne Parameterliste ... 604
Im Programm definierte Eingabefelder ... 605
INPUT Syntax 2 - Beschreibung ... 605
INPUT-Statement unter Nicht-Screen-Modi ... 607
Eingabedaten aus dem Natural-Stack ... 609
INPUT-Statement im Batch-Betrieb ... 610
IX ... 613
78 INSERT (SQL) ... 615
Funktion INSERT (SQL) ... 616
Syntax-Beschreibung INSERT (SQL) ... 616
Beispiel für INSERT (SQL) ... 622
79 INTERFACE ... 625
Funktion INTERFACE ... 626
Syntax-Beschreibung INTERFACE ... 627
80 LIMIT ... 633
Funktion LIMIT ... 634
Syntax-Beschreibung LIMIT ... 635
Beispiele LIMIT ... 635
81 LOOP ... 637
Funktion LOOP ... 638
Einschränkung bei LOOP ... 638
Syntax-Beschreibung LOOP ... 639
Beispiele LOOP ... 639
82 MERGE (SQL) ... 641
Funktion MERGE (SQL) ... 642
Einschränkung bei MERGE (SQL) ... 642
Syntax-Beschreibung MERGE (SQL) ... 642
Beispiele MERGE (SQL) ... 648
83 METHOD ... 651
Funktion METHOD ... 652
Syntax-Beschreibung METHOD ... 652
Beispiel für METHOD-Statement ... 653
84 MOVE ... 657
Funktion MOVE ... 658
Syntax 1 - MOVE ... 658

Syntax 2 - MOVE SUBSTRING ... 660
Syntax 3 - MOVE BY NAME / POSITION ... 662
Syntax 4 - MOVE EDITED (Editiermaske bei operand2) ... 663
Syntax 5 - MOVE EDITED (Editiermaske bei operand1) ... 665
Syntax 6 - MOVE LEFT / RIGHT JUSTIFIED ... 666
Syntax 7 - MOVE NORMALIZED ... 667
Syntax 8 - MOVE ENCODED ... 669
Syntax 9 - MOVE ALL ... 672
Beispiele MOVE ... 675
85 MOVE INDEXED ... 687
86 MULTIPLY ... 689
Funktion MULTIPLY ... 690
Syntax 1 - MULTIPLY-Statement ohne GIVING-Klausel ... 690
Syntax 2 - MULTIPLY-Statement mit GIVING-Klausel ... 691
Beispiel für MULTIPLY-Statement ... 692
87 NEWPAGE ... 695
Funktion NEWPAGE ... 696
Syntax-Beschreibung NEWPAGE ... 696
Beispiel für NEWPAGE-Statement ... 697
88 OBTAIN ... 701
Funktion OBTAIN ... 702
Einschränkung bei OBTAIN ... 703
Syntax-Beschreibung OBTAIN ... 703
Beispiele OBTAIN ... 707
89 ON ERROR ... 711
Funktion ON ERROR ... 712
Einschränkung bei ON ERROR ... 712
Syntax-Beschreibung ON ERROR ... 713
ON ERROR-Verarbeitung in Subprogrammen ... 713
Systemvariablen *ERROR-NR und *ERROR-LINE ... 714
Beispiel für ON ERROR-Statement ... 714
90 OPEN CONVERSATION ... 717
Funktion OPEN CONVERSATION ... 718
Syntax-Beschreibung OPEN CONVERSATION ... 718
Weitere Informationen und Beispiele zu OPEN CONVERSATION ... 719
91 OPTIONS ... 721
Funktion OPTIONS ... 722
Verarbeitung von mehreren OPTIONS-Statements ... 722
X ... 723
92 PARSE JSON ... 725
Funktion PARSE JSON ... 726
Syntax-Beschreibung PARSE JSON ... 727
Beispiele PARSE JSON ... 730
93 PARSE XML ... 737
Funktion PARSE XML ... 738

Syntax-Beschreibung PARSE XML ... 740
Beispiele PARSE XML ... 743
94 PASSW ... 751
Funktion PASSW ... 752
Einschränkung bei PASSW ... 753
Syntax-Beschreibung PASSW ... 753
Beispiel für PASSW-Statement ... 754
95 PERFORM ... 755
Funktion PERFORM ... 756
Syntax-Beschreibung PERFORM ... 756
Beispiele PERFORM ... 759
96 PERFORM BREAK PROCESSING ... 763
Funktion PERFORM BREAK PROCESSING ... 764
Syntax-Beschreibung PERFORM BREAK PROCESSING ... 764
Beispiel für PERFORM BREAK PROCESSING ... 765
97 PRINT ... 767
Funktion PRINT ... 768
Syntax-Beschreibung PRINT ... 769
Beispiel für PRINT ... 774
98 PROCESS ... 777
Funktion PROCESS ... 778
Einschränkung bei PROCESS ... 778
Syntax-Beschreibung PROCESS ... 779
99 PROCESS COMMAND ... 781
Funktion PROCESS COMMAND ... 783
Für die Kommandoverarbeitung verwendetes Datendefinitionsmodul (DDM) ... 783
Security-Aspekte ... 784
Syntax-Beschreibung PROCESS COMMAND ... 784
Beispiele PROCESS COMMAND ... 796
100 PROCESS PAGE ... 799
Funktion PROCESS PAGE ... 800
Syntax 1 - PROCESS PAGE ... 800
Syntax 2 - PROCESS PAGE USING ... 803
Syntax 3 - PROCESS PAGE UPDATE ... 806
Syntax 4 - PROCESS PAGE MODAL ... 810
Beispiele ... 812
101 PROCESS SQL (SQL) ... 813
Funktion PROCESS SQL ... 814
Syntax-Beschreibung PROCESS SQL ... 814
Beispiele PROCESS SQL ... 815
102 PROPERTY ... 817
Funktion PROPERTY ... 818
Syntax-Beschreibung PROPERTY ... 818
Beispiel für PROPERTY-Statement ... 819

XI 821
103 READ 823
Funktion READ 824
Syntax-Beschreibung READ 825
Bei READ verfügbare Systemvariablen 839
Beispiele READ 840
104 READLOB 849
Funktion READLOB 850
Einschränkungen bei READLOB 850
Syntax-Beschreibung READLOB 851
Bei READLOB verfügbare Systemvariablen 853
Funktionstechnische Überlegungen bei READLOB 854
Beispiele READLOB 855
105 READ RESULT SET (SQL) 857
Funktion READ RESULT SET (SQL) 858
Einschränkung bei READ RESULT SET (SQL) 859
Syntax-Beschreibung READ RESULT SET (SQL) 859
Beispiel für READ RESULT SET (SQL) 861
106 READ WORK FILE 863
Funktion READ WORK FILE 864
Syntax 1 - READ WORK FILE mit Verarbeitungsschleife 864
Syntax 2 - READ WORK FILE ohne Verarbeitungsschleife 865
Syntax-Beschreibung READ WORK FILE 866
Feldlängen bei READ WORK FILE 869
Variabler Index-Bereich bei READ WORK FILE 869
Verarbeitung dynamischer Variablen bei READ WORK FILE 870
Verarbeitung von X-Arrays bei READ WORK FILE 870
Beispiele READ WORK FILE 870
107 REDEFINE 877
Funktion REDEFINE 878
Einschränkung bei REDEFINE 878
Syntax-Beschreibung REDEFINE 878
Beispiele REDEFINE 879
108 REDUCE 883
Funktion REDUCE 884
Syntax-Beschreibung REDUCE 884
109 REINPUT 889
Funktion REINPUT 890
Syntax-Beschreibung REINPUT 891
Beispiele REINPUT 898
110 REJECT 903
111 RELEASE 905
Funktion RELEASE 906
Syntax-Beschreibung RELEASE 906
Beispiel für RELEASE 907

112 REPEAT ... 909
Funktion REPEAT ... 910
Syntax-Beschreibung REPEAT ... 910
Beispiele REPEAT ... 911
113 REQUEST DOCUMENT ... 915
Funktion REQUEST DOCUMENT ... 916
Syntax-Beschreibung REQUEST DOCUMENT ... 917
Automatisch erzeugte Header ... 923
URL-Kodierung bei Sonderzeichen ... 924
HTTP/HTTPS Status Codes: weitergeleitet und verweigert ... 926
Beispiele REQUEST DOCUMENT ... 926
114 RESET ... 933
Funktion RESET ... 934
Syntax-Beschreibung RESET ... 934
Beispiel für RESET-Statement ... 936
115 RESIZE ... 939
Funktion RESIZE ... 940
Syntax-Beschreibung RESIZE ... 940
116 RETRY ... 945
Funktion RETRY ... 946
Einschränkung bei RETRY ... 946
Beispiel für RETRY-Statement ... 946
117 ROLLBACK (SQL) ... 949
Funktion ROLLBACK (SQL) ... 950
Hinweis für Nicht-Natural-Programme ... 950
Beispiel für ROLLBACK (SQL) ... 951
118 RUN ... 953
Funktion RUN ... 954
Syntax-Beschreibung RUN ... 954
Dynamische Quellcode-Generierung und -Ausführung ... 955
Beispiel für RUN-Statement ... 956
XII ... 959
119 SELECT (SQL) ... 961
Funktion SELECT (SQL) ... 962
Syntax 1 - Cursor-orientierte Auswahl ... 962
Syntax 2 - Nicht cursor-orientierte Auswahl ... 964
Syntax-Element-Beschreibung SELECT (SQL) ... 964
Verknüpfungsabfragen (Join Query) ... 981
120 SEND METHOD ... 983
Funktion SEND METHOD ... 984
Syntax-Beschreibung SEND METHOD ... 984
Beispiel für SEND METHOD-Statement ... 987
121 SEPARATE ... 995
Funktion SEPARATE ... 996
Syntax-Beschreibung SEPARATE ... 996

Regeln und operative Aspekte ... 1000
Beispiele SEPARATE ... 1002
122 SET CONTROL ... 1009
Funktion SET CONTROL ... 1010
Syntax-Beschreibung SET CONTROL ... 1010
Beispiele SET CONTROL ... 1011
123 SET GLOBALS ... 1013
Funktion SET GLOBALS ... 1014
Parameterangabe bei SET GLOBALS ... 1014
Beispiel für SET GLOBALS-Statement ... 1016
124 SET KEY ... 1017
Funktion SET KEY ... 1018
Syntax-Beschreibung SET KEY ... 1018
Tasten programm-sensitiv machen und deaktivieren ... 1020
Kommandos/Programme einer Taste zuweisen ... 1021
Eingabedaten einer Taste zuweisen (DATA) ... 1022
Tastenfunktion vorübergehend deaktivieren ... 1022
Helproutine zuweisen (HELP) ... 1023
Dynamische Funktionszuweisung (DYNAMIC) ... 1023
GUI-Element-Zuweisung deaktivieren (DISABLED) ... 1024
SET KEY-Statements auf verschiedenen Programmebenen ... 1024
Namen zuweisen ... 1026
Beispiel für SET KEY-Statement ... 1027
125 SET TIME ... 1029
Funktion SET TIME ... 1030
Beispiel für SET TIME ... 1030
126 SET WINDOW ... 1033
Funktion SET WINDOW ... 1034
Syntax-Beschreibung SET WINDOW ... 1034
Beispiel für SET WINDOW ... 1035
127 SKIP ... 1037
Funktion SKIP ... 1038
Syntax-Beschreibung SKIP ... 1038
Beispiel für SKIP-Statement ... 1039
128 SORT ... 1041
Funktion SORT ... 1042
Einschränkungen beim SORT-Statement ... 1043
Syntax-Beschreibung SORT ... 1043
Phasen der SORT-Verarbeitung ... 1046
Beispiel für SORT-Statement ... 1047
Benutzung externer Sortierprogramme ... 1052
129 STACK ... 1053
Funktion STACK ... 1054
Syntax-Beschreibung STACK ... 1054
Beispiel für STACK-Statement ... 1057

130 STOP 1059
Funktion STOP 1060
Beispiel für STOP-Statement 1060
XIII 1063
131 STORE 1065
Funktion STORE 1066
Datenbankspezifische Anmerkungen zu STORE 1067
Syntax-Beschreibung STORE 1067
Beispiel für STORE-Statement 1069
132 SUBTRACT 1073
Funktion SUBTRACT 1074
Syntax 1 - SUBTRACT-Statement ohne GIVING-Klausel 1074
Syntax 2 - SUBTRACT-Statement mit GIVING-Klausel 1075
Beispiel für SUBTRACT-Statement 1076
133 SUSPEND IDENTICAL SUPPRESS 1079
Funktion SUSPEND IDENTICAL SUPPRESS 1080
Syntax-Beschreibung SUSPEND IDENTICAL SUPPRESS 1080
Beispiele SUSPEND IDENTICAL SUPPRESS 1081
134 TERMINATE 1085
Funktion TERMINATE 1086
Syntax-Beschreibung TERMINATE 1086
Kontrollübergabe nach Abbruch 1087
Beispiel für TERMINATE-Statement 1087
135 UPDATE 1089
Funktion UPDATE 1090
Einschränkungen bei UPDATE 1091
Datenbankspezifische Anmerkungen zu UPDATE 1091
Syntax-Beschreibung UPDATE 1091
Beispiel für UPDATE-Statement 1093
136 UPDATE (SQL) 1095
Funktion UPDATE (SQL) 1096
Syntax 1 - Searched UPDATE 1096
Syntax 2 - Positioned UPDATE 1099
Beispiele UPDATE (SQL) 1100
137 UPDATELOB 1103
Funktion UPDATELOB 1104
Einschränkungen bei UPDATELOB 1104
Syntax-Beschreibung UPDATELOB 1105
Bei UPDATELOB verfügbare Systemvariable 1106
Funktionstechnische Überlegungen zu UPDATELOB 1107
Beispiele UPDATELOB 1107
138 UPLOAD PC FILE 1111
Funktion UPLOAD PC FILE 1113
Syntax-Beschreibung UPLOAD PC FILE 1113
Beispiel für UPLOAD PC FILE-Statement 1114

139 WRITE ... 1117
Funktion WRITE ... 1118
Syntax 1 - Dynamische Formatierung ... 1119
Syntax 1 - Beschreibung ... 1119
Syntax 2 - Vordefinierte Form/Map benutzen ... 1127
Syntax 2 - Beschreibung ... 1128
Beispiele WRITE ... 1129
140 WRITE TITLE ... 1135
Funktion WRITE TITLE ... 1136
Einschränkungen bei WRITE TITLE ... 1137
Syntax-Beschreibung WRITE TITLE ... 1137
Beispiel für WRITE TITLE ... 1141
141 WRITE TRAILER ... 1143
Funktion WRITE TRAILER ... 1144
Einschränkungen bei WRITE TRAILER ... 1145
Syntax-Beschreibung WRITE TRAILER ... 1145
Beispiel für WRITE TRAILER ... 1150
142 WRITE WORK FILE ... 1153
Funktion WRITE WORK FILE ... 1154
Syntax-Beschreibung WRITE WORK FILE ... 1154
Externe Darstellung der Felder ... 1157
Besonderheiten bei Systemfunktionen ... 1158
Verarbeitung großer und dynamischer Variablen ... 1158
Beispiele WRITE WORK FILE ... 1158
XIV Natural-SQL-Statements benutzen ... 1161
143 Common Set und Extended Set ... 1163
144 Grundlegende Syntaxbestandteile ... 1165
Konstanten ... 1166
Namen ... 1167
Parameter ... 1170
Include Columns-Klausel ... 1174
Period-Klausel ... 1175
Natural-Formate und SQL-Datentypen ... 1176
145 Das Natural-View-Konzept ... 1179
146 Skalar-Ausdrücke ... 1181
scalar-expression ... 1182
scalar-operator ... 1182
factor ... 1183
row-value-expression ... 1194
147 Suchbedingungen ... 1195
search-condition ... 1196
predicate ... 1196
148 SELECT-Ausdrücke ... 1203
selection ... 1204
table-expression ... 1205

149 Flexible SQL 1217
Flexible SQL benutzen 1218
Textvariablen in flexibler SQL angeben 1219
XV Referenzierte Beispielprogramme 1223
150 Referenzierte Beispielprogramme 1225
ASSIGN 1226
AT BREAK 1227
AT END OF DATA 1229
AT END OF PAGE 1230
AT START OF DATA 1231
AT TOP OF PAGE 1232
DEFINE SUBROUTINE 1233
FIND 1234
FOR 1236
HISTOGRAM 1237
IF 1238
PERFORM BREAK PROCESSING 1239
READ 1240
REPEAT 1241
SORT 1243
STORE 1244
UPDATE 1246
Beispielprogramme für Systemvariablen 1247
Stichwortverzeichnis 1251

Vorwort

Diese Dokumentation beschreibt die nativen Natural-Statements (DML) und Natural-SQL-Statements. Sie ist in die folgenden Abschnitte untergliedert:

Statements nach Funktionen	Übersicht über die nach Funktionen eingeteilten Statements.
Syntax-Symbole und Operandentabellen	Informationen zu den Symbolen, die in den Diagrammen verwendet werden, in denen die Syntax und Operanden-Definitionstabellen dargestellt sind.
Natural-SQL-Statements benutzen	Spezifische Regeln zur Benutzung der Natural Natural-SQL-Statements.
Referenzierte Beispielprogramme	Zusätzliche Beispielprogramme, die in der *Statements*- und *Systemvariablen*-Dokumentation referenziert werden.

Verwandte Themen:

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

Statements in alphabetischer Reihenfolge:

A - C	D - F	G - O	P - R	S - Z
ACCEPT/REJECT	DECIDE FOR	GET	PARSE JSON	SELECT (SQL)
ADD	DECIDE ON	GET SAME	PARSE XML	SEND METHOD
ASSIGN	DEFINE CLASS	GET	PASSW	SEPARATE
AT BREAK	DEFINE DATA	TRANSACTION	PERFORM	SET CONTROL
AT END OF DATA	DEFINE FUNCTION	DATA	PERFORM BREAK	SET GLOBALS
AT END OF PAGE	DEFINE PRINTER	HISTOGRAM	PROCESSING	SET KEY
AT START OF DATA	DEFINE PROTOTYPE	IF	PRINT	SET TIME
AT TOP OF PAGE	DEFINE	IF SELECTION	PROCESS	SET WINDOW
BACKOUT	SUBROUTINE	IGNORE	PROCESS COMMAND	SKIP
TRANSACTION	DEFINE WINDOW	INCLUDE	PROCESS PAGE	SORT
BEFORE BREAK	DEFINE WORK FILE	INPUT	PROCESS	STACK
PROCESSING	DELETE	INSERT (SQL)	SQL (SQL)	STOP
CALL	DELETE (SQL)	INTERFACE	PROPERTY	STORE
CALL FILE	DISPLAY	LIMIT	READ	SUBTRACT
CALL LOOP	DIVIDE	LOOP	READ RESULT SET	SUSPEND
CALLDBPROC (SQL)	DO/DOEND	MERGE (SQL)	(SQL)	IDENTICAL
CALLNAT	DOWNLOAD PC FILE	METHOD	READ WORK FILE	SUPPRESS
CLOSE	EJECT	MOVE	READLOB	TERMINATE
CONVERSATION	END	MOVE INDEXED	REDEFINE	UPDATE

A - C	D - F	G - O	P - R	S - Z
CLOSE PC FILE	END TRANSACTION	MULTIPLY	REDUCE	UPDATE (SQL)
CLOSE PRINTER	ESCAPE	NEWPAGE	REINPUT	UPDATELOB
CLOSE WORK FILE	EXAMINE	OBTAIN	REJECT	UPLOAD PC FILE
COMMIT (SQL)	EXPAND	ON ERROR	RELEASE	WRITE
COMPOSE	FETCH	OPEN	REPEAT	WRITE TITLE
COMPRESS	FIND	CONVERSATION	REQUEST DOCUMENT	WRITE TRAILER
COMPUTE	FOR	OPTIONS	RESET	WRITE WORK FILE
CREATE OBJECT	FORMAT		RESIZE	
			RETRY	
			ROLLBACK (SQL)	
			RUN	

1 Über diese Dokumentation

- Dokumentationskonventionen 2
- Online-Informationen und Support 2
- Datenschutz 3

Dokumentationskonventionen

Konvention	Beschreibung
Fettschrift	>Kennzeichnet Elemente auf einem Bildschirm.
`Nichtproportionale Schrift`	Kennzeichnet Namen und Orte von Diensten im Format *`Ordner.Unterordner.Dienst`*, Programmierschnittstellen (APIs), Namen von Klassen, Methoden und Properties in Java.
`Kursivschrift`	Kennzeichnet: Variablen, für die Sie situations- oder umgebungsspezifische Werte angeben müssen. Neue Begriffe, wenn sie erstmals im Text auftreten. Verweise auf andere Dokumentationsquellen.
`Nichtproportionale Schrift`	Kennzeichnet: Text, den Sie eingeben müssen. Meldungen, die vom System angezeigt werden. Programmcode.
{ }	Zeigt eine Reihe von Auswahlmöglichkeiten an, von denen Sie eine auswählen müssen. Geben Sie nur die innerhalb der geschweiften Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole { } ein.
|	Trennt zwei sich gegenseitig ausschließende Auswahlmöglichkeiten in einer Syntaxzeile voneinander ab. Geben Sie eine der Auswahlmöglichkeiten ein. Geben Sie nicht das Symbol | ein.
[ ]	Zeigt eine oder mehrere Optionen an. Geben Sie nur die innerhalb der eckigen Klammern vorhandenen Informationen ein. Geben Sie nicht die Klammersymbole [ ] ein.
...	Zeigt an, dass Sie mehrere Auswahlmöglichkeiten desselben Typs eingeben können. Geben Sie nur die Informationen ein. Geben Sie nicht die drei Auslassungspunkte (...) ein.

Online-Informationen und Support

Produktdokumentation

Sie finden die Produktdokumentation auf unserer Dokumentationswebsite unter **https://documentation.softwareag.com**.

Zusätzlich können Sie auch über **https://www.softwareag.cloud** auf die Dokumentation für die Cloud-Produkte zugreifen. Navigieren Sie zum gewünschten Produkt und gehen Sie dann, je nach Produkt, zu „Developer Center“, „User Center“ oder „Documentation“.

Produktschulungen

Sie finden hilfreiches Produktschulungsmaterial auf unserem Lernportal unter **https://knowledge.softwareag.com**.

Tech Community

Auf der Website unserer Tech Community unter **https://techcommunity.softwareag.com** können Sie mit Experten der Software AG zusammenarbeiten. Von hier aus können Sie zum Beispiel:

- Unsere umfangreiche Wissensdatenbank durchsuchen.
- In unseren Diskussionsforen Fragen stellen und Antworten finden.
- Die neuesten Nachrichten und Ankündigungen der Software AG lesen.
- Unsere Communities erkunden.
- Unsere öffentlichen Repositories auf GitHub and Docker unter **https://github.com/softwareag** und **https://hub.docker.com/publishers/softwareag** besuchen und weitere Ressourcen der Software AG entdecken.

Produktsupport

Support für die Produkte der Software AG steht lizenzierten Kunden über unser Empower-Portal unter **https://empower.softwareag.com** zur Verfügung. Für viele Dienstleistungen auf diesem Portal benötigen Sie ein Konto. Wenn Sie noch keines haben, dann können Sie es unter **https://empower.softwareag.com/register** beantragen. Sobald Sie ein Konto haben, können Sie zum Beispiel:

- Produkte, Aktualisierungen und Programmkorrekturen herunterladen.
- Das Knowledge Center nach technischen Informationen und Tipps durchsuchen.
- Frühwarnungen und kritische Alarme abonnieren.
- Supportfälle öffnen und aktualisieren.
- Anfragen für neue Produktmerkmale einreichen.

Datenschutz

Die Produkte der Software AG stellen Funktionen zur Verarbeitung von personenbezogenen Daten gemäß der Datenschutz-Grundverordnung (DSGVO) der Europäischen Union zur Verfügung. Gegebenenfalls sind in der betreffenden Systemverwaltungsdokumentation entsprechende Schritte dokumentiert.

I

- 2 Syntax-Symbole und Operandentabellen 7
- 3 Statements nach Funktionen 13

2 Syntax-Symbole und Operandentabellen

- Syntax-Symbole 8
- Operandentabelle 10

Dieser Abschnitt 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.
ABCDEF	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.
ABCDEF	Ist ein Teil eines Wortes in Großbuchstaben unterstrichen (kein Hyperlink!), kann der unterstrichene Teil als Abkürzung für das jeweilige Wort verwendet werden.
abcdef	Elemente, die in Kleinbuchstaben und kursiv dargestellt sind, sind variable Informationen, an deren Stelle Sie die gewünschten Angaben machen. **Anmerkung:** Anstelle von *statement* oder *statements* 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 ein 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.
Sonstige Symbole (außer `[ ] { } | . . . , . . . : . . .`)	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`, `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
operand1	C	S	A	G	N/M	E	A	U	N	P	I	F	B	D	T	L	C	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 *modifizierbar* ist, steht in der *Systemvariablen*-Dokumentation.
E	Arithmetische Ausdrücke.	

Mögliche Formate

Gibt an, welche Formate der Operand haben darf:

A	Alphanumerisch (EBCDIC-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
O	Objekt-Handle (`HANDLE OF OBJECT`)

Referenzierung erlaubt

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

Siehe auch *Referenzieren mit Statement-Labels* und *Referenzieren mit Quellcode-Zeilennummern* im Kapitel *Benutzervariablen* im *Leitfaden zur Programmierung*.

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 14
- Arithmetische Funktionen und Datenzuweisungen 16
- Schleifenverarbeitung 16
- Erstellen von Ausgabe-Reports 17
- Bildschirmgenerierung für interaktive Verarbeitung 17
- Verarbeitung logischer Bedingungen 18
- Aufrufen von Programmen und Subprogrammen 18
- Functions 19
- Beenden von Programmen und Sessions 19
- Verarbeitung von Arbeitsdateien/PC-Dateien 19
- Komponentenbasierte Programmierung 20
- Speicherverwaltung für dynamische Variablen/X-Arrays 20
- Natural Remote Procedure Call 20
- Internet und Parsing 21
- Verschiedene Statements 21
- Reporting Mode-Statements 21
- Statements für Predict Case und Entire DB 23

Dieser Abschnitt 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:

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

AT START OF DATA	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 Zeilen aus einer Tabelle, ohne einen Cursor zu verwenden („searched“ DELETE), oder löscht Zeilen aus einer Tabelle, auf die der Cursor zeigt („positioned“ DELETE).
INSERT	Fügt einer Tabelle eine oder mehrere neue Reihen hinzu.
MERGE	Aktualisiert eine Tabelle mit den angebenen Eingabedaten. Diejenigen Zeilen in der Zieltabelle, die zu den Eingabedaten passen, werden gemäß den Angaben aktualisiert, und Zeilen, welche in der Zieltabelle nicht vorhanden sind, werden eingefügt.
PROCESS SQL	Dient dazu, mit SQL-Statements auf eine Datenbank zuzugreifen.
READ RESULT SET	Liest eine Ergebnismenge, die von einer mit einem vorhergehenden CALLDBPROC-Statement aufgerufenen „Stored Procedure“ erzeugt wurde.
ROLLBACK	Entspricht dem Natural-DML-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 Tabellenzeilen gelesen werden kann, als auch das nicht cursor-orientierte „singleton“ SELECT, das maximal eine Zeile liest.
UPDATE	Führt UPDATE-Operationen auf Zeilen in einer Tabelle aus, ohne einen Cursor zu verwenden („searched“ UPDATE), oder auf Spalten in einer Zeile, 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 eins einen 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 Subprogrammen

Die folgenden Statements werden zum Aufrufen von Programmen und Subprogrammen 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 Subprogramms.
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.

Functions

Folgende Natural-Statements dienen zum Erstellen von Functions:

DEFINE FUNCTION	Dient zum Erstellen neuer benutzerdefinierter Functions, die anstelle von Operanden in Natural-Statements aufgerufen werden können. Functions werden in Objekten des Typs Function definiert.
DEFINE PROTOTYPE	Dient zum Angeben der Eigenschaften, die bei einem Function Call verwendet werden sollen.
Function Call	Dient zum Aufrufen von Natural-Objekten des Typs 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/PC-Dateien

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 in eine Arbeitsdatei.
DOWNLOAD PC FILE	Ermöglicht die Übertragung von Daten auf den PC.
READ WORK FILE	Liest Daten von einer Arbeitsdatei.
UPLOAD PC FILE	Ermöglicht die Übertragung von Daten von einem PC auf eine z/OS- oder Linux-Plattform.
CLOSE WORK FILE	Schließt eine Arbeitsdatei.
CLOSE PC FILE	Schließt eine spezifische PC-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 Subprogramm als Implementierung einer Methode zu.
PROPERTY	Weist außerhalb einer Schnittstellendefinition eine Objektdaten-Variable als Implementierung einer Eigenschaft zu.

Speicherverwaltung für dynamische Variablen/X-Arrays

EXPAND	Erweitert den zugewiesenen Speicher für dynamische Variablen auf eine gegebene Größe.
REDUCE	Reduziert die Größe einer dynamischen Variablen.
RESIZE	Passt die Größe einer dynamischen Variablen an.

Natural Remote Procedure Call

OPEN CONVERSATION	Ermöglicht es dem Client, eine Konversation zu öffnen und die Remote-Subprogramme anzugeben, die an der Konversation beteiligt sein sollen.
CLOSE CONVERSATION	Ermöglicht es 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 entfernte Subprogramme innerhalb einer Konversation zur Verfügung stehen sollen, ohne dass Sie die Variablen explizit als Parameter mit den entsprechenden CALLNAT-Statements übergeben müssen.

Siehe auch Verwendete Natural Statementsin Natural RPC (Remote Procedure Call)-Dokumentation.

Internet und Parsing

PARSE JSON	Gestattet es, JSON-Dokumente von einem Natural-Programm aus zu parsen.
PARSE XML	Gestattet es, XML-Dokumente von einem Natural-Programm aus zu parsen.
REQUEST DOCUMENT	Ermöglicht den Zugang zu einem externen System.

Verschiedene Statements

DEFINE DATA	Definiert die Datenelemente, die in einem Natural-Programm oder -Subprogramm verwendet werden sollen.
END	Zeigt das Ende des Quellcodes eines Natural-Programms bzw. -Subprogramms 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.
RELEASE	Löschen aller auf dem Natural-Stack abgelegten 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 auf dem 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 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.
UPLOAD PC FILE	Ermöglicht die Übertragung von Daten von einem PC auf eine z/OS- oder Linux-Plattform.

Statements für Predict Case und Entire DB

Die folgenden Natural-Statements können in Verbindung mit Predict Case und Entire DB Engine verwendet werden:

- DLOGOFF/DLOGON
- IMPORT
- EXPORT
- SHOW

Weitere Informationen zu diesen Statements siehe *Predict Case* Documentation.

II

- 4 ACCEPT/REJECT 27
- 5 ADD 33
- 6 ASSIGN 39
- 7 AT BREAK 41
- 8 AT END OF DATA 51
- 9 AT END OF PAGE 57
- 10 AT START OF DATA 65
- 11 AT TOP OF PAGE 71
- 12 BACKOUT TRANSACTION 77
- 13 BEFORE BREAK PROCESSING 81
- 14 CALL 87
- 15 CALL FILE 115
- 16 CALL LOOP 121
- 17 CALLDBPROC (SQL) 125
- 18 CALLNAT 131
- 19 CLOSE CONVERSATION 139

4 ACCEPT/REJECT

- Funktion ACCEPT/REJECT 28
- Syntax-Beschreibung ACCEPT/REJECT 28
- Verarbeitung mehrerer ACCEPT/REJECT-Statements 29
- Limit-Notation 30
- Hold-Status 30
- Beispiele ACCEPT/REJECT 30

```
{ ACCEPT }
{ REJECT } [IF] logical-condition
```

Dieser Abschnitt 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 ACCEPT/REJECT

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 ACCEPT/REJECT

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
IF	**IF-KLausel:** 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.
logical-condition	**Logische Bedingung:** Das Basis-Kriterium ist ein relationaler Ausdruck. Mehrere relationale Ausdrücke können mit logischen Operatoren zu komplexen Kriterien kombiniert werden (AND, OR). Arithmetische Ausdrücke können auch zur Bildung eines relationalen Ausdrucks benutzt werden. Felder können Datenbankfelder oder Benutzervariablen sein. Weitere Informationen zu logischen Bedingungen, siehe *Logische Bedingungen* im *Leitfaden zur Programmierung*. Wenn ACCEPT/REJECT mit einem HISTOGRAM-Statement benutzt wird, kann nur das im HISTOGRAM-Statement angegebene Datenbankfeld als logische Bedingung verwendet werden.

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.

Hold-Status

Eine ACCEPT/REJECT-Verarbeitung hat keinen Einfluss darauf, ob ein im „Hold“ gehaltener Datensatz freigegeben wird - es sei denn, der Profilparameter RI ist auf ON gesetzt.

Beispiele ACCEPT/REJECT

- 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 ADD 34
- Syntax 1 - ADD-Statement ohne GIVING-Klausel 34
- Syntax 2 - ADD-Statement mit GIVING-Klausel 35
- Beispiel für ADD-Statement 36

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: COMPRESS | COMPUTE | DIVIDE | EXAMINE | MOVE | MOVE ALL | MULTIPLY | RESET | SEPARATE | SUBTRACT

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion ADD

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

Dieses Statement hat zwei verschiedene Syntax-Strukturen.

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 Abschnitt *Arithmetische Operationen mit Arrays* im *Leitfaden zur Programmierung*.
3. Zum Format der Operanden siehe auch Abschnitt *Formatwahl im Hinblick auf die Verarbeitungszeit* im *Leitfaden zur Programmierung*.

Syntax 1 - ADD-Statement ohne GIVING-Klausel



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
operand1	C	S	A		N			N	P	I	F		D	T				ja	nein
operand2		S	A		M			N	P	I	F		D	T				ja	ja

Syntax-Element-Beschreibung:

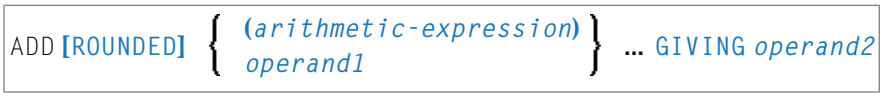
Syntax-Element	Beschreibung
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 TO *operand2*	**Operanden:** *operand1* und *operand2* sind Summanden. Das Ergebnis wird in *operand2* (Ergebnisfeld) gespeichert. Somit ist das Statement gleichbedeutend mit: `operand2 := operand2 + operand1 + ...`
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

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-Statement mit GIVING-Klausel



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
operand1	C	S	A		N				N	P	I	F		D	T				ja	nein
operand2		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:

Syntax-Element	Beschreibung:
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 GIVING *operand2*	**Operands:** *operand1* ist ein Summand. *operand2* wird nur benutzt, um das Ergebnis der Operation aufzunehmen und wird *nicht* in die Addition einbezogen. Somit ist das Statement gleichbedeutend mit: `operand2 := operand1 + ...`
ROUNDED	**ROUNDED-Option:** Wenn das Schlüsselwort ROUNDED angegeben wird, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.



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 für ADD-Statement

```
** 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 AT BREAK 42
- Syntax-Beschreibung AT BREAK 43
- Gruppenwechsel auf mehreren Ebenen 45
- Beispiele AT BREAK 46

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 }
```

Dieser Abschnitt 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 AT BREAK

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 AT BREAK

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(*r*)	**Statement-Referenzierung:** Ein AT BREAK-Statement wird zum letztenmal ausgeführt, wenn eine mit FIND, READ, READ WORK FILE, HISTOGRAM oder SORT initiierte Verarbeitungsschleife beendet wird. Normalerweise bezieht sich das AT BREAK-Statement hierbei auf die äußerste aktive Schleife. Wollen Sie, dass sich die abschließende AT BREAK-Verarbeitung auf eine andere offene Schleife bezieht (die Schleife, in der das AT BREAK-Statement steht, oder eine äußere Schleife), so verwenden Sie hierzu die Notation (*r*), wobei *r* das Statement-Label bzw. die Quellcode-Zeilennummer des betreffenden schleifeninitiierenden Statements ist. **Beispiel:**

Syntax-Element	Beschreibung
	``` 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 ... ``` 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. Soll eine ganze Hierarchie von AT BREAK-Statements sich auf eine andere als die aktive Schleife beziehen, so müssen Sie die Notation (*r*) bei dem ersten AT BREAK-Statement angeben; sie bezieht sich dann auch auf alle innerhalb der Hierarchie folgenden AT BREAK-Statements.
operand1	**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.
/n/	**Notation /*n*/** Sie haben auch die Möglichkeit, einen Teil eines Feldes zum Kontrollfeld zu machen: Mit der Notation */n/* geben Sie an, dass nur die ersten *n* 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 *n* Stellen sich ändert. Diese Möglichkeit besteht allerdings nur bei Feldern, die das Format A, B, N oder P haben. 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.
statement	**Bei Eintreten der BREAK-Bedingung auszuführende(s) Statement(s):** Im Structured Mode muss anstelle von *statement* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe ***Beispiele*** weiter unten.
END-BREAK *statement*	**Ende des AT BREAK-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-BREAK zum Beenden des AT BREAK-Statements benutzt werden.

Syntax-Element	Beschreibung
DO *statement* ... DOEND	Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das AT BREAK-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

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:

```
FIND ...
   AT BREAK
   ...
   END-BREAK
   AT BREAK
   ...
   END-BREAK
   AT BREAK
   ...
   END-BREAK
END-FIND
...
```

Reporting Mode:

```
FIND ...
   AT BREAK
      DO
      ...
      DOEND
   AT BREAK
      DO
      ...
      DOEND
...
```

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 AT BREAK

- Beispiel 1 - AT BREAK
- Beispiel 2 - AT BREAK mit der Notation /n/
- Beispiel 3 - AT BREAK mit Gruppenwechseln 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         NAME         #LEAVE-DUE-L
                        CODE
-------------------- ---------- -------------------- ------------

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 AT END OF DATA 52
- Einschränkungen 53
- Syntax-Beschreibung AT END OF DATA 53
- Beispiel für AT END OF DATA-Statement 54

Structured Mode-Syntax

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

Reporting Mode-Syntax

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

Dieser Abschnitt 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 AT END OF DATA

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.

Dieser 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 AT END OF DATA

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(r)`	**Referenzierungsnotation:** 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

Syntax-Element	Beschreibung
	Quellcode-Zeilennummer des Statements ist, welches die gewünschte Schleife initiiert.
statement ...	**Bei Eintreten der Bedingung auszuführende(s) Statement(s):** Im Structured Mode muss anstelle von *statement* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe ***Beispiel*** weiter unten.
END-ENDDATA	**Ende des AT END OF DATA-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-ENDDATA zum Beenden des AT END OF DATA-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das AT END OF DATA-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.
statement DO *statement ...* DOEND	

Beispiel für AT END OF DATA-Statement

```
** 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         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     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 AT END OF PAGE 58
- Syntax-Beschreibung AT END OF PAGE 60
- Beispiele AT END OF PAGE 61

Structured Mode-Syntax

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

Reporting Mode-Syntax

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

Dieser Abschnitt 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 AT END OF PAGE

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 AT END OF PAGE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(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 `(rep)` nicht angegeben wird, bezieht sich das `AT END OF PAGE`-Statement auf den ersten Report (Report 0). Informationen zum Steuern des Formats eines mit Natural erzeugten Ausgabe-Reports siehe *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
`statement`	**Bei Eintreten der Bedingung auszuführende(s) Statement(s):** Im Structured Mode muss anstelle von *`statement`* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe ***Beispiel*** weiter unten.
`END-ENDPAGE`	**Ende des AT END OF PAGE-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-ENDPAGE` zum Beenden des `AT END OF PAGE`-Statements benutzt werden. Im Reporting Mode werden die Statements `DO ... DOEND` benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das `AT END OF PAGE`-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements `DO ... DOEND` weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.
`statement` `DO` *`statement ...`* `DOEND`	

Beispiele AT END OF PAGE

- 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            SALARY   CURRENCY
                            POSITION                       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
*
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           CITY
                                           ADDRESS
-------------------- -------------------- ---------- --------------------

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 AT START OF DATA 66
- Syntax-Beschreibung AT START OF DATA 67
- Beispiel für AT START OF DATA-Statement 68

Structured Mode-Syntax

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

Reporting Mode-Syntax

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

Dieser Abschnitt 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 AT START OF DATA

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 schleifeninitiierende 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 AT START OF DATA

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(r)`	**Referenzieren einer bestimmten Verarbeitungsgsschleife:** 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 `(r)`, wobei *r* das Statement-Label oder die Quellcode-Zeilennummer des Statements ist, welches die gewünschte Schleife initiiert.
statement ...	**Bei Eintreten der Bedingung auszuführende(s) Statement(s):** Im Structured Mode muss anstelle von *statement* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe ***Beispiel*** weiter unten.
END-START	**Ende des AT START OF DATA-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-START zum Beenden des AT START OF DATA-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das AT START OF DATA-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.
statement DO *statement ...* DOEND	

Beispiel für AT START OF DATA-Statement

```
** 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
      WRITE NOTITLE NOHDR 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-NOREC
    /*
   AT START OF DATA
      INPUT (AD=O) '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 AT TOP OF PAGE 72
- Einschränkung 73
- Syntax-Beschreibung AT TOP OF PAGE 73
- Beispiel für AT TOP OF PAGE-Statement 74

Structured Mode-Syntax

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

Reporting Mode-Syntax

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

Dieser Abschnitt 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 AT TOP OF PAGE

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 AT TOP OF PAGE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(*rep*)	**Report-Spezifikation:** Mit der Notation (*rep*) kann ein bestimmter anderer Report angegeben werden, auf den sich das AT TOP OF PAGE-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 (*rep*) nicht angegeben wird, bezieht sich das AT TOP OF PAGE-Statement auf den ersten Report (Report 0). Weitere Informationen zur Steuerung des Formats eines mit Natural erzeugten Ausgabe-Reports siehe *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
statement ...	**Bei Eintreten der Bedingung auszuführende(s) Statement(s):** Im Structured Mode muss anstelle von *statement* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe ***Beispiel*** weiter unten.
END-TOPPAGE	**Ende des AT TOP OF PAGE-Statements:**

Syntax-Element	Beschreibung
statement ... DO *statement* ... DOEND	Im Structured Mode muss das für Natural reservierte Schlüsselwort END-TOPPAGE zum Beenden des AT TOP OF PAGE-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das AT TOP OF PAGE-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Beispiel für AT TOP OF PAGE-Statement

```
** 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               VEMMELEV             SYSA02
------------------------------------------------------------------------------

ENDING NAME:    LARSEN
```

Äquivalentes Reporting-Mode-Beispiel: ATPEX1R.

12 BACKOUT TRANSACTION

- Funktion BACKOUT TRANSACTION 78
- Einschränkung 79
- Datenbank-spezifische Anmerkungen 79
- Beispiel für BACKOUT TRANSACTION-Statement 79

```
BACKOUT [TRANSACTION]
```

Dieser Abschnitt 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 BACKOUT TRANSACTION

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
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.

Beispiel für BACKOUT TRANSACTION-Statement

```
** 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
*
FIND EMPLOY-VIEW WITH DEPT = #DEPT
  IF NO RECORDS FOUND
    REINPUT 'NO RECORDS FOUND'
  END-NOREC
  INPUT 'NAME:       ' NAME (AD=O) /
        '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 BEFORE BREAK PROCESSING 82
- Einschränkungen 83
- Syntax-Beschreibung BEFORE BREAK PROCESSING 83
- Beispiel für BEFORE BREAK PROCESSING-Statement 84

Structured Mode-Syntax

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

Reporting Mode-Syntax

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

Dieser Abschnitt 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 BEFORE BREAK PROCESSING

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 BEFORE BREAK PROCESSING

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
statement...	**Statement(s) für die Break-Verarbeitung:** Im Structured Mode muss anstelle von *statement* je nach Situation eines oder mehrere passende Statements angeben werden. Siehe **Beispiel** weiter unten Wird keine an einen Gruppenwechsel geknüpfte Verarbeitung ausgeführt (d.h. wenn die betreffende Verarbeitungsschleife kein AT BREAK-Statement enthält), dann wird der BEFORE BREAK PROCESSING-Statement-Block *nicht* ausgeführt.
END-BEFORE	**Ende des BEFORE BREAK PROCESSING-Statements:** Im Structured Mode muss das reservierte Natural-Wort END-BEFORE zum Beenden des BEFORE BREAK PROCESSING-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um das BEFORE BREAK PROCESSING-Statement zu beenden. Falls Sie nur ein
statement ... DO *statement* ... DOEND	

Syntax-Element	Beschreibung
	einziges Statement angeben, können Sie die Statements `DO ... DOEND` weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Beispiel für BEFORE BREAK PROCESSING-Statement

```
** 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 CALL 88
- Syntax-Beschreibung CALL 88
- Return Code 89
- Registerbelegung 90
- Speicherplatzausrichtung 91
- Adabas-Aufrufe 91
- Direktes/Dynamisches Laden 91
- Linkage-Konventionen 93
- Programm-Eigenschaften 95
- Aufrufen eines PL/I-Programms 95
- Aufruf eines C-Programms 97
- INTERFACE4 101

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

Dieser Abschnitt 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 Subprogrammen***

Funktion CALL

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

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

Ein CALL-Statement kann auch in einem Programm, das unter Kontrolle eines TP-Monitors ausgeführt werden soll, angegeben werden, vorausgesetzt der TP-Monitor unterstützt eine CALL-Schnittstelle.

Syntax-Beschreibung CALL

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

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.
operand1	**Programmname:** Der Name des aufgerufenen Programms (*operand1*) 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.
[USING] *operand2*	**Parameter:** Das CALL-Statement kann bis zu 128 Parameter (*operand2*) enthalten, es sei denn, die INTERFACE4-Option wird verwendet. In diesem Fall wird die Anzahl der Parameter durch die Größe des katalogisierten Objekts begrenzt. Abhängig von allen anderen Statements im Natural-Objekt dürfen bis zu 16370 Parameter benutzt werden. Hierbei gelten die Standard-Linkage-Registerkonventionen. Für jedes angegebene Parameterfeld wird in der Parameterliste eine Adresse übergeben. 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. Die interne Darstellung der positiven Vorzeichen gepackter Zahlen ändert sich auf den vom PSIGNF-Parameter des NTCMPO-Makros (Kompilieroptionen) angegebenen Wert, *bevor* die Kontrolle an das externe Programm übergeben wird.

Return Code

Um den Condition Code eines aufgerufenen Programms (Inhalt von Register 15 beim Rücksprung zum Natural-Programm) 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
...
```

Registerbelegung

Register	Inhalt
R1	Adresse der Parameteradressenliste.
R2	Adresse der Feld-(Parameter-)Beschreibungsliste. Die Feldbeschreibungsliste enthält Informationen über die ersten 128 in der Parameterliste übergebenen Felder. Jede Beschreibung ist ein 4 Byte langer Eintrag, der sich wie folgt zusammensetzt: ■ 1. Byte: Variablentyp (A, B, ...) ■ Überschreitet eine Variable vom Typ A die Länge von 32767 Bytes, wird sie als Typ Y übergeben. ■ Überschreitet eine Variable vom Typ B die Länge von 32767 Bytes, wird sie als Typ X übergeben. Die Typen X und Y wurden zur Unterstützung langer alphanumerischer und binärer Variablen für die standardmäßige `CALL`-Schnittstelle eingeführt. Falls das Feld vom Typ N oder P ist: ■ 2. Byte: Anzahl der Stellen insgesamt. ■ 3. Byte: Anzahl der Stellen vor dem Komma (Dezimalpunkt). ■ 4. Byte: Anzahl der Stellen nach dem Komma (Dezimalpunkt). Falls das Feld vom Typ X oder Y ist: ■ 2. Byte: ungenutzt. ■ 3.-4. Byte: enthält Null. ■ Länge des Feldes wird über R4 übergeben. Bei allen anderen Feldtypen: ■ 2. Byte: ungenutzt. ■ 3.-4. Byte: Länge des Feldes.
R3	Adresse der Feldlängen-Liste. Diese Liste enthält die Längen der in der Parameterliste übergebenen Felder. Bei einem Array ergibt sich die Länge aus der Summe der Längen der einzelnen Ausprägungen.
R4	Nur bei Typ X und Y: Ein vier Byte langer Eintrag für jede Variable des Typs A oder B, die die Größe von 32767 Bytes überschreitet.
R13	Adresse des 18-Wort-Speicherbereiches.
R14	Rücksprungadresse.
R15	Eingangsadresse/Return Code.

Speicherplatzausrichtung

Siehe Abschnitt *Speicherplatzausrichtung* im *Leitfaden zur Programmierung*.

Adabas-Aufrufe

Ein aufgerufenes Programm darf einen Adabas-Aufruf beinhalten. Das aufgerufene Programm darf kein Adabas-Open- oder -Close-Kommando absetzen. Adabas öffnet alle angesprochenen Dateien.

Soll Adabas-EXU-Modus (EXclusive Update) verwendet werden, muss zum Öffnen aller angesprochenen Dateien der Natural-Profilparameter `OPRB` (Öffnen und Schließen der Datenbank) benutzt werden; bevor Sie den EXU-Update-Modus einsetzen, sollten Sie in jedem Fall Ihren Natural-Administrator konsultieren.

Wenn ein aufgerufenes Programm Adabas-Kommandos absetzt, die eine Transaktion beginnen oder beenden, kann Natural nicht die Änderung des Transaktions-Status erkennen.

Aufrufe an Adabas müssen den Aufruf-Konventionen für die Adabas-Anwendungsprogrammierschnittstelle (API) für den/das entsprechende/n TP-Monitor oder Betriebssystem entsprechen. Dies gilt auch, wenn Natural als Server agiert, z.B. unter z/OS oder SMARTS.

Direktes/Dynamisches Laden

Das aufgerufene Programm kann entweder direkt an den Natural-Nukleus gelinkt werden (indem es mit dem `CSTATIC`-Parameter im Natural-Parametermodul angegeben wird; vgl. *Operations*-Dokumentation), oder es kann dynamisch geladen werden, wenn es zum erstenmal aufgerufen wird.

Soll es dynamisch geladen werden, so muss die Lademodul-Library, die das aufgerufene Programm enthält, mit der Natural-Lade-Library verkettet werden, und zwar entweder in der Natural-Ausführungs-JCL oder in der entsprechenden Programm-Library des TP-Monitors. Weitere Einzelheiten erfragen Sie bitte bei Ihrem Natural-Administrator.

Beispiel:

Das Beispiel auf der nächsten Seite zeigt ein Natural-Programm, welches das COBOL-Programm `TABSUB` aufruft, und zwar zu dem Zweck, Ländercodes (`COUNTRY-CODE`) in die entsprechenden Ländernamen (`COUNTRY-NAME`) umzusetzen. Das Natural-Programm übergibt zwei Parameter an das COBOL-Programm:

- der erste Parameter ist der Ländercode, so wie er von der Datenbank gelesen wird;

- der zweite Parameter dient dazu, den Ländernamen zurückzugeben.

Aufrufendes Natural-Programm:

```
** Example 'CALEX1': CALL PROGRAM 'TABSUB'
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
  2 COUNTRY
*
1 #COUNTRY      (A3)
1 #COUNTRY-NAME (A15)
1 #FIND-FROM    (D)
1 #FIND-TO      (D)
END-DEFINE
*
MOVE EDITED '19550701' TO #FIND-FROM (EM=YYYYMMDD)
MOVE EDITED '19550731' TO #FIND-TO   (EM=YYYYMMDD)
*
FIND EMPLOY-VIEW WITH BIRTH = #FIND-FROM THRU #FIND-TO
  MOVE COUNTRY TO #COUNTRY
  /*
  CALL 'TABSUB' #COUNTRY #COUNTRY-NAME
  /*
  DISPLAY NAME BIRTH (EM=YYYY-MM-DD) #COUNTRY-NAME
END-FIND
END
```

Aufgerufenes COBOL-Programm `TABSUB`:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSUB.
REMARKS. THIS PROGRAM PROVIDES THE COUNTRY NAME
        FOR A GIVEN COUNTRY CODE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 COUNTRY-CODE  PIC X(3).
01 COUNTRY-NAME  PIC X(15).
PROCEDURE DIVISION USING COUNTRY-CODE COUNTRY-NAME.
P-CONVERT.
   MOVE SPACES TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'BLG' MOVE 'BELGIUM' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'DEN' MOVE 'DENMARK' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'FRA' MOVE 'FRANCE' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'GER' MOVE 'GERMANY' TO COUNTRY-NAME.
```

```
   IF COUNTRY-CODE = 'HOL' MOVE 'HOLLAND' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'ITA' MOVE 'ITALY' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'SPA' MOVE 'SPAIN' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'UK'  MOVE 'UNITED KINGDOM' TO COUNTRY-NAME.
P-RETURN.
GOBACK.
```

Linkage-Konventionen

Im Batch-Betrieb wird Standard-Linkage-Register-Notation verwendet. Für jeden TP-Monitor gelten hierbei andere Konventionen. Diese Konventionen müssen unbedingt befolgt werden, da es sonst zu unvorhersehbaren Ergebnissen kommen kann.

Die für die einzelnen unterstützten TP-Monitore gültigen Konventionen sind im folgenden beschrieben.

- CALL unter Com-plete
- CALL unter CICS

CALL unter Com-plete

Das aufgerufene Programm muss sich in der Com-plete-Online-Lade-Library befinden, damit Com-plete es dynamisch laden kann. Zum Katalogisieren des Programms kann die Com-plete-Utility `ULIB` verwendet werden.

CALL unter CICS

Das aufgerufene Nicht-Natural-Subprogramm muss sich in einer der Load Libraries der `DFHRPL`-Library-Verkettung der CICS JCL befinden. Das Subprogramm muss einen Verarbeitungsprogrammtabellen (PPT)-Eintrag in der aktiven PPT haben, damit CICS das Subprogramm finden und laden kann.

Der Parameter `CALLRPL` im Macro `NTCICSP` steuert, wo und wie die Parameterlistenadressen an das externe Subroutinenprogramm übergeben werden.

Wenn Sie statt der Adressen der Adressenliste die Parameterwerte selbst in eine CICS COMMAREA (oder einen Container) übergeben wollen, müssen Sie vor dem Aufruf das Natural-Terminalkommando (Call-Optionen) `%P=C` (oder `%P=CC`) absetzen. Alternativ können Sie den Profilparameter `PGP` benutzen, um die Call-Optionen zu definieren.

Wenn ein Natural-Programm ein Nicht-Natural-Subprogramm unter CICS aufruft, erfolgt der Aufruf durch eine `EXEC CICS LINK`-Anforderung. Dies gilt nicht bei LE-Subprogrammen. Informationen bezüglich der Unterstützung von IBM Language Environment (LE)-Subprogrammen durch Natural siehe Abschnitt *LE-Subprogramme* in der *Operations*-Dokumentation.

Wenn Sie für den Aufruf stattdessen Standard-Linkage-Konventionen verwenden (direkte Verzweigung mittels einer BASR-Anweisung), müssen Sie das Terminalkommando `%P=S` benutzen oder beim Profilparameter `PGP` den Eigenschaftswert `STDLQ` angeben. In diesem Fall muss das aufgerufene Programm Standard-Linkage-Konventionen mit Standard-Register-Benutzung befolgen. Wenn keine `EXEC CICS`-Kommandos ausgeführt werden sollen, brauchen die von CICS bereitgestellten Stub-Routinen (z.B. `DFHELII`) nicht mit dem aufgerufenen Subprogramm verlinkt zu sein.

Setzen Sie das Terminalkommando `%P=SQ` ab oder geben Sie beim Profilparameter `PGP` die Eigenschaft `STDLQ` an, wenn die folgenden Bedingungen beide zutreffen:

- Das mittels Standard-Linkage-Konventionen aufgerufene Subprogramm ist nur quasi-reentrant (aber nicht thread-sicher und voll reentrant), d.h., es ist mit dem CICS-PPT-Attribut `CONCURRENCY(QUASIRENT)` definiert.
- Natural ist mit dem CICS-PPT-Attribut `CONCURRENCY(REQUIRED)` definiert.

Dadurch wird das aufgerufene Subprogramm unter dem CICS QR TCB ausgeführt.

Weitere Informationen siehe *Threadsafe Considerations* in der *TP Monitor Interfaces*-Dokumentation.

Wenn ein mit `AMODE=24` gelinktes Programm in einer 31-Bit-Modus-Umgebung aufgerufen wird und die Threads im Speicherbereich oberhalb 16 MB zugewiesen werden, wird automatisch ein Wertaufruf (Call by Value) durchgeführt, d.h. die angegebenen Parameter, die an das aufgerufene Programm übergeben werden sollen, werden in den Speicherbereich unterhalb 16 MB kopiert.

Return Codes unter CICS

CICS selbst unterstützt zwar keine Return Codes für einen Aufruf mit CICS-Konventionen (`EXEC CICS LINK`), außer beim Aufruf von C/C++-Programmen, wobei von der `exit()`-Funktion oder dem `return()`-Statement übergebene Werte im `EIBRESP2`-Feld gespeichert werden. Aber das Natural/CICS-Interface unterstützt Return Codes für das `CALL`-Statement: Wenn die Kontrolle vom aufgerufenen Programm zurückgegeben wird, überprüft Natural zuerst das `EIBRESP2`-Feld auf einen Return Code ungleich Null.

Dann überprüft Natural, ob sich das erste Vollwort der COMMAREA geändert hat (nur wenn die COMMAREA für die Parameteradressliste benutzt wurde). Ist dies der Fall, wird dessen neuer Inhalt als Return Code genommen. Hat es sich nicht geändert, wird das erste Vollwort der TWA geprüft (nur wenn die TWA für die Parameteradressliste benutzt wurde) und dessen neuer Inhalt als Return Code genommen. Hat sich keins der beiden Vollwörter geändert, ist der Return Code `0`.

Anmerkung: Wenn die Parameterwerte in der COMMAREA übergeben werden (`%P=C`), wird nur das Feld `EIBRESP2` auf einen Returncode überprüft, d.h. bei Nicht-C/C++-Programmen ist der Return Code immer `0`.

Programm-Eigenschaften

To define properties permanently for external programs to be called, use the profile parameter PGP. To define temporary properties for external programs to be called, use the terminal command %P=.

Um Eigenschaften für externe Programme permanent festzulegen, können Sie den Profilparameter `PGP` benutzen. Um temporäre Eigenschaften für aufzurufende externe Programme festzulegen, benutzen Sie das Terminalkommando `%P=`.

Aufrufen eines PL/I-Programms

Ist das aufgerufene Programm in PL/I geschrieben, erfordert dies zusätzlich Folgendes:

Da die Parameterliste eine Standardliste und keine von einem anderen PL/I-Programm übergebene Argumentliste ist, zeigen die übergebenen Adressen nicht auf einen `LOCATOR DESCRIPTOR`. Dieses Problem löst man, indem man die Parameterfelder als arithmetische Variablen definiert. Dies bewirkt, dass PL/I die Parameterliste als Adressen von Daten und nicht als Adressen von `LOCATOR DESCRIPTOR`-Kontrollblöcken behandelt.

Es wird empfohlen, die Parameterfelder folgendem Beispiel entsprechend zu definieren:

```
PLIPROG: PROC(INPUT_PARM_1, INPUT_PARM_2) OPTIONS(MAIN);
      DECLARE (INPUT_PARM_1, INPUT_PARM_2) FIXED;
      PTR_PARM_1 = ADDR(INPUT_PARM_1);
      PTR_PARM_2 = ADDR(INPUT_PARM_2);
      DECLARE FIRST_PARM        PIC '99'   BASED (PTR_PARM_1);
      DECLARE SECOND_PARM       CHAR(12)   BASED (PTR_PARM_2);
```

Jeder Parameter der Input-Liste sollte als eindeutiges Element behandelt werden. Die Anzahl der Eingabeparameter sollte mit der Zahl der vom Natural-Programm übergebenen genau übereinstimmen. Die Eingabeparameter und ihre Attribute müssen den Natural-Definitionen entsprechen, andernfalls kann es zu unvorhersehbaren Ergebnissen kommen. Weitere Informationen zur Übergabe von Parametern an PL/I-Programme finden Sie in der entsprechenden IBM PL/I-Dokumentation.

Siehe auch die folgenden Beispiele.

- Beispiel für den Aufruf eines PL/I-Programms

- Beispiel für den Aufruf eines PL/I-Programms, das unter CICS abläuft

Beispiel für den Aufruf eines PL/I-Programms

```
** Example 'CALEX2': CALL PROGRAM 'NATPLI'
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 AREA-CODE
  2 REDEFINE AREA-CODE
    3 #AC         (N1)
*
1 #INPUT-NUMBER   (N2)
1 #OUTPUT-COMMENT (A15)
END-DEFINE
*
READ EMPLOY-VIEW IN LOGICAL SEQUENCE BY NAME
                 STARTING FROM 'WAGNER'
  MOVE ' ' TO #OUTPUT-COMMENT
  MOVE #AC TO #INPUT-NUMBER
  /*
  CALL 'NATPLI' #INPUT-NUMBER #OUTPUT-COMMENT
  /*
END-READ
*
END
```

Aufgerufenes PL/I-Programm `NATPLI`:

```
NATPLI:  PROC(PARM_COUNT, PARM_COMMENT) OPTIONS(MAIN);
    /*                                              */
    /* THIS PROGRAM ACCEPTS AN INPUT NUMBER         */
    /* AND TRANSLATES IT TO AN OUTPUT CHARACTER     */
    /* STRING FOR PLACEMENT ON THE FINAL            */
    /* NATURAL REPORT                               */
    /*                                              */
    /*                                              */
   DECLARE (PARM_COUNT, PARM_COMMENT) FIXED;
   DECLARE ADDR BUILTIN;
   COUNT_PTR = ADDR(PARM_COUNT);
   COMMENT_PTR = ADDR(PARM_COMMENT);
   DECLARE  INPUT_NUMBER    PIC '99' BASED (COUNT_PTR);
   DECLARE  OUTPUT_COMMENT  CHAR(15) BASED (COMMENT_PTR);
   DECLARE COMMENT_TABLE(9) CHAR(15) STATIC INITIAL
      ('COMMENT1   ',
       'COMMENT2   ',
       'COMMENT3   ',
       'COMMENT4   ',
       'COMMENT5   ',
```

```
       'COMMENT6   ',
       'COMMENT7   ',
       'COMMENT8   ',
       'COMMENT9   ');
    OUTPUT_COMMENT = COMMENT_TABLE(INPUT_NUMBER);
    RETURN;
END NATPLI;
```

Beispiel für den Aufruf eines PL/I-Programms, das unter CICS abläuft

```
** Example 'CALEX3': CALL PROGRAM 'CICSP'
************************************************************************
DEFINE DATA LOCAL
1 #MESSAGE (A10) INIT <' '>
END-DEFINE
*
CALL 'CICSP' #MESSAGE
DISPLAY #MESSAGE
*
END
```

Aufgerufenes PL/I-Programm `CICSP`:

```
CICSP: PROCEDURE OPTIONS (MAIN REENTRANT);
       DCL  TWA_ADDRESS    BASED(TWA_POINTER);
       DCL  LIST_ADDRESS   POINTER;
       DCL  PTR_TO_LIST    POINTER  BASED(LIST_ADDRESS);
       DCL  PARM_01        POINTER;
       DCL  MESSAGE        CHAR(10) BASED(PARM_01);
       EXEC CICS ADDRESS TWA(TWA_POINTER);
       MESSAGE='SUCCESS'; EXEC CICS RETURN; END CICSP;
```

Aufruf eines C-Programms

Bevor Sie ein C-Programm benutzen können, müssen Sie es zuerst kompilieren und linken.

- Zum Erstellen des ausführbaren Moduls benutzen Sie z.B. den C Compiler von IBM. Da der C Compiler von IBM LE-Code erzeugt, ist das Muster nur in einer LE-Umgebung ausführbar. Um LE-Programme auszuführen, muss das Natural-Frontend mit LE-Funktionalität installiert werden.
- Wenn Sie einen anderen C Compiler wie z.B. Dignus oder SASC benutzen möchten, ist es erforderlich, dass Sie ein Modul erstellen, das von einer Nicht-C-Umgebung aufrufbar ist. Weitere Informationen entnehmen Sie der betreffenden Compiler-Dokumentation.
- Die Include-Datei `NATUSER` muss im C-Programm mit enthalten sein.

Für INTERFACE4 geschriebene C-Programme können sowohl auf Großrechner-Systemen als auch auf Linux- oder Windows-Systemen verwendet werden. Dagegen sind für die standardmäßige CALL-Schnittstelle geschriebene C-Programme plattformabhängig.

Wenn das C-Programm über CALL INTERFACE4 aufgerufen werden soll oder wenn ein Natural-Subprogramm vom C-Programm aufgerufen wird, muss NATXCAL4 an das ausführbare Modul gelinkt werden. Benutzen Sie eine der INTERFACE4-Rückruffunktionen, um die Parameter-Beschreibung und Parameter-Werte einzulesen. Die Rückruf-Funktionen sind im folgenden beschrieben.

Benutzen Sie die Funktion ncxr_if4_callnat, um ein Natural-Subprogramm vom C-Programm aus auszuführen.

Prototyp:

```
int ncxr_if4_callnat ( char *natpgm, int parmnum, struct parameter_description ↩
*descr );
```

Parameter-Beschreibung:

Parameter	Beschreibung	
natpgm	Name des aufzurufenden Natural-Subprogramms.	
parmnum	Anzahl der an das Subprogramm zu übergebenden Parameter-Felder.	
descr	Addresse einer struct parameter_description. Siehe ***Operanden-Struktur für Interface4***.	
return	Rückgabewert:	Informationen:
	0	OK Tritt ein Natural-Fehler auf, während das Subprogramm ausgeführt wird, werden Informationen über diesen Fehler in der Variable natpgm in der Form *NAT*nnnn* zurückgegeben, wobei *nnnn* die betreffende Natural-Fehlernummer ist.
	-1	Ungültige Parameter-Nummer.
	-2	Interner Fehler.

Siehe auch die folgenden Beispiele.

- Beispiel zum Aufruf eines C-Programms über einen Standard-CALL

- Beispiel zum Aufruf eines C-Programms über CALL INTERFACE4

Beispiel zum Aufruf eines C-Programms über einen Standard-CALL

```
** Example 'CALEX4': CALL PROGRAM 'ADD'
************************************************************************
DEFINE DATA LOCAL
1 #OP1  (I4)
1 #OP2  (I4)
1 #SUM  (I4)
END-DEFINE
*
CALL 'ADD' #OP1 #OP2 #SUM
DISPLAY #SUM
*
END
```

Aufgerufenes C-Programm `ADD`:

```
/*
**  Example C Program ADD.c
*/
NATFCT ADD (int *op1, int *op2, int *sum)
{
*sum = *op1 + *op2;    /* add opperands */

return 0;              /* return successfully */
} /* ADD */
```

Beispiel zum Aufruf eines C-Programms über CALL INTERFACE4

```
** Example 'CALEX5': CALL PROGRAM 'ADD4'
************************************************************************
DEFINE DATA LOCAL
1 #OP1  (I4)
1 #OP2  (I4)
1 #SUM  (I4)
END-DEFINE
*
CALL INTERFACE4 'ADD4' #OP1 #OP2 #SUM
DISPLAY #SUM
*
END
```

Aufgerufenes C-Programm `ADD4`:

```
NATFCT ADD4 NATARGDEF(numparm, parmhandle, parmdec)
{
NATTYP_I4 op1, op2, sum;               /* local integers */
int i;                                 /* loop counter */
struct parameter_description desc;
int rc;                                /* return code access Funktions */

/*
** test number of arguments
*/
if (numparm != 3) return 1;

/*
** test types of arguments
*/
for (i = 0; i < (int) numparm; i++)
{
       rc = ncxr_get_parm_info( i, parmhandle, &desc );
      if ( rc ) return rc;

     if ( desc.format != 'I' || desc.length != sizeof(NATTYP_I4) || desc.dimensions ↩
!= 0 )
      {
               return 2;          /* invalid parameter */
      }
}

/*
** get arguments
*/
rc = ncxr_get_parm( 0, parmhandle, sizeof op1, (void *)&op1 );
if ( rc ) return rc;

rc = ncxr_get_parm( 1, parmhandle, sizeof op2, (void *)&op2 );
if ( rc ) return rc;

/*
** perform the addition
*/
sum = op1 + op2;

/*
** move the result back to operand 3
*/
rc = ncxr_put_parm( 2, parmhandle, sizeof sum, (void *)&sum );
if ( rc ) return rc;

/*
** all ok, return success to the caller
*/
```

```
return 0;
} /* ADD4 */
```

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 INTERFACE4	CALL-Statement mit Schlüsselwort INTERFACE4
Anzahl der möglichen Parameter	128	16370 oder weniger
Maximale Länge eines Parameters	keine Beschränkung	1 GB
Array-Informationen einlesen	nein	ja
Unterstützung großer und dynamischer Operanden	voller Lesezugriff, Schreiben ohne Ändern der Größe des Operanden	ja
Parameter-Zugriff über API	direkt	über API

Die maximale Anzahl der Parameter wird durch die maximale Größe des generierten Programms (GP) und durch die maximale Größe eines Statements beschränkt. 16370 Parameter sind möglich, wenn das Programm nur das CALL-Statement enthält. Die maximale Anzahl ist niedriger, wenn andere Statements benutzt werden.

INTERFACE4 - Externe 3GL-Programmierschnittstelle

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

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

`USR_WORD`	`numparm;`	16 Bit umfassender Kurzwert ohne Vorzeichen, der die Gesamtzahl der übertragenen Operanden (*operand2*) 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, `realloc()` und `free()` sind nicht zulässig.	
`int`	`format`	Felddatentyp: `NCXR_TYPE_ALPHA`, usw. (*natuser.h*).	
`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 `IF4_MAX_DIM`)	
`int`	`dimensions`	Anzahl Dimensionen (`0` bis `IF4_MAX_DIM`).	
`int`	`length_all`	Gesamtlänge des Arrays in Bytes.	
`int`	`flags`	Mehrere Flag-Bits, durch `OR` bitweise miteinander kombiniert, Bedeutung:	
		`IF4_FLG_PROTECTED`	Parameter ist schreibgeschützt.
		`IF4_FLG_DYNAMIC`	Parameter ist dynamische Variable.
		`IF4_FLG_NOT_CONTIGUOUS`	Array-Elemente berühren sich nicht (es steht ein Leerzeichen zwischen ihnen).
		`IF4_FLG_AIV`	Der Parameter ist eine anwendungsunabhängige Variable.
		`IF4_FLG_DYNVAR`	Parameter ist dynamische Variable.
		`IF4_FLG_XARRAY`	Parameter ist ein X-Array.

		IF4_FLG_LBVAR_0	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:

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 `struct parameter_description`	
`return`	Rückgabewert:	Informationen:
	`0`	OK
	`-1`	Fehlerhafte Parameter-Nummer
	`-2`	Interner Fehler

Parameter	Beschreibung	
	-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:

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:

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=O).
	-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.

Parameter	Beschreibung	
	-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:

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:

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:

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:

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:

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:

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.

Parameter	Beschreibung	
	-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:

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 CALL FILE 116
- Einschränkung 116
- Syntax-Beschreibung CALL FILE 117
- Beispiel für CALL FILE-Statement 118

Structured Mode-Syntax

```
CALL FILE'program-name' operand1 operand2
  statement ...
END-FILE [(r)]
```

Reporting Mode-Syntax

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

Dieser Abschnitt 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 Subprogrammen***

Funktion CALL FILE

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 CALL FILE

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
'*program-name*'	Der Name des aufzurufenden Nicht-Natural-Programmes.
operand1	Kontrollfeld: *operand1* dient dazu, Kontrollinformationen zu liefern.
operand2	*operand2* definiert den Datensatz-Bereich. Das Format des zu lesenden Datensatzes kann mit Felddefinitionseinträgen (oder `FILLER` *nX*), 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.
statement ...	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` (*r*) `LOOP` (*r*)	**Ende des CALL FILE-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-FILE` zum Beenden des `CALL FILE`-Statements benutzt werden. Im Structured Mode können Sie bei `END-FILE` Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement `LOOP` zum Beenden des `CALL FILE`-Statements benutzt. Im Reporting Mode können Sie bei `LOOP` Labels oder Zeilennummern angeben.

Beispiel für CALL FILE-Statement

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 xxxxxxxxxx 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 CALL LOOP 122
- Einschränkung 122
- Syntax-Beschreibung CALL LOOP 123
- Beispiel für CALL LOOP-Statement 124

Structured Mode-Syntax

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

Reporting Mode-Syntax

```
CALL LOOP operand1 [operand2] ...40
  statement ...
LOOP [(r)]
```

Dieser Abschnitt 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 Subprogrammen***

Funktion CALL LOOP

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 CALL LOOP

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

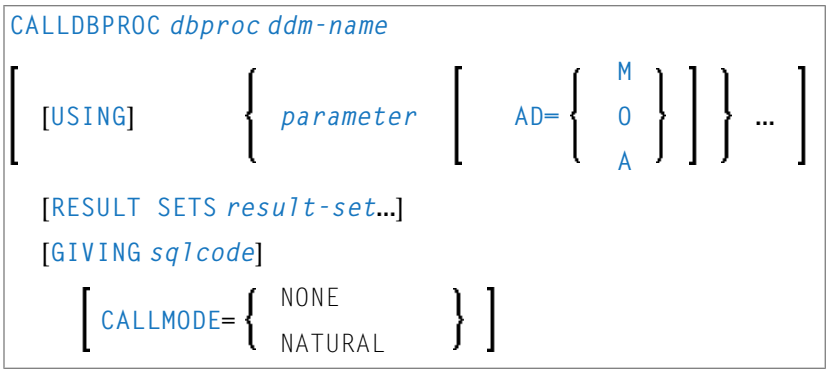
Syntax-Element	Beschreibung
operand1	Der Name des aufgerufenen Nicht-Natural-Programms (*operand1*) kann entweder als Konstante angegeben werden oder - falls je nach Programmlogik verschiedene Programme aufgerufen werden sollen - als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname muss linksbündig in der Variablen stehen.
operand2	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.
statement ...	Die mit CALL LOOP initiierte Verarbeitungsschleife muss mit einem ESCAPE-Statement beendet werden.
END-LOOP [(*r*)]	**Ende des CALL LOOP-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-LOOP zum Beenden des CALL LOOP-Statements benutzt werden. Im Structured Mode können Sie bei END-LOOP Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des CALL LOOP-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.
LOOP [(*r*)]	

Beispiel für CALL LOOP-Statement

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

17 CALLDBPROC (SQL)

- Funktion CALLDBPROC (SQL) 126
- Einschränkung 127
- Syntax-Beschreibung CALLDBPROC (SQL) 127
- Beispiel CALLDBPROC (SQL) 129



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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *CALLDBPROC - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

Funktion CALLDBPROC (SQL)

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 zurückzugeben, 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.

Einschränkung

Dieses Statement steht nur bei Natural for Db2 zur Verfügung.

Syntax-Beschreibung CALLDBPROC (SQL)

Syntax-Element	Beschreibung	
dbproc	**Aufzurufende Stored Procedure:** Als *dbproc* geben Sie den Namen der Stored Procedure an, die aufgerufen werden soll. Der Name kann entweder als alphanumerische Variable oder, mit Apostrophen ('), als Konstante angegeben werden. Der Name muss den Regeln für Stored-Procedure-Namen des Ziel-Datenbanksystems entsprechen. Falls die Stored Procedure ein Natural-Subprogramm ist, darf der eigentliche Procedure-Name nicht länger als 8 Stellen sein.	
ddm-name	**Name eines Natural-Datendefinitionsmoduls (DDM):** Der Name eines DDM muss angegeben werden, um die „Adresse“ der Datenbank, welche die Stored Procedure ausführt, bereitzustellen. Weitere Informationen siehe *ddm-name*.	
[USING] *parameter*	**Zu übergebende(r) Parameter:** Hier können Sie einen oder mehrere Parameter angeben, die vom aufrufenden Objekt an die Stored Procedure übergeben werden sollen. Als *parameter* können Sie Folgendes angeben: ■ eine *host-variable* (optional mit INDICATOR- und LINDICATOR-Klauseln) ■ eine Konstante oder ■ das Schlüsselwort NULL. Siehe weitere Informationen zu *host-variable*.	
AD=	**Attribut-Definition:** Wenn es sich beim *parameter* um eine *host-variable* handelt, können Sie sie mit einem der folgenden Attribute versehen:	
	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.

Syntax-Element	Beschreibung	
		(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`.)
	Wenn *`parameter`* eine Konstante ist, kann `AD` nicht explizit angegeben werden. Für Konstanten gilt immer `AD=O`.	
`RESULT SETS` *`result-set`*	**Feld für Result-Set-Locator-Variable:** Als *`result-set`* geben Sie ein Feld an, in das der Result-Set-Locator zurückgegeben werden soll. Ein Result-Set muss eine Variable mit Format/Länge I4 sein. Der Wert einer Result-Set-Variablen ist lediglich eine Zahl, die den Result Set identifiziert und die in einem nachfolgenden `READ RESULT SET`-Statement referenziert werden kann. Die Reihenfolge der Result-Set-Werte entspricht der Reihenfolge der von der Stored Procedure zurückgegebenen Result Sets. Der Inhalt der Result Sets kann von einem nachfolgenden `READ RESULT SET`-Statement verarbeitet werden. Wenn kein Result Set zurückgegeben wird, enthält die entsprechende Result-Set-Variable den Wert Null (`0`). Es können mehrere Result Sets angegeben werden. Siehe auch *Result Sets* (im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation).	
`GIVING` *`sqlcode`*	**GIVING *`sqlcode`*-Option:** Diese Option können Sie benutzen, um den SQLCODE des SQL `CALL`-Statements zu erhalten, das die Stored Procedure aufruft. Wenn Sie diese Option angeben und der SQLCODE der Stored Procedure ist ungleich Null (`0`), wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muss die als Reaktion auf den SQLCODE-Wert auszuführende Maßnahme im aufrufenden Natural-Objekt programmiert werden. Das *`sqlcode`*-Feld muss eine Variable von Format/Länge I4 sein. Wenn Sie die Option `GIVING` *`sqlcode`* nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQLCODE der Stored Procedure ungleich Null (`0`) ist.	
`CALLMODE=`	**CALLMODE-Parameter:** Mögliche Angaben:	

Syntax-Element	Beschreibung	
	CALLMODE=NATURAL	Diese Angabe gilt, wenn die Stored Procedure ein Natural-Subprogramm ist, das mit PARAMETER STYLE GENERAL oder PARAMETER STYLE GENERAL WITH NULL definiert ist, sonst geben Sie NONE (Standardwert) an. Diese Angabe wirkt sich auch auf interne Parameter aus, die an die bzw. von der Stored Procedure übergeben werden; siehe *CALLMODE=NATURAL* (im Abschnitt *CALLDBPROC* der Natural for Db2-Dokumentation).
	CALLMODE=NONE	Dies ist die Standardeinstellung.

Beispiel CALLDBPROC (SQL)

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 abzurufen.

Drei Parameter-Felder werden an DEMO_PROC übergeben: der erste und zweite Parameter übergeben jeweils Start- und Endewerte 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 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=O) #END (AD=O) #NAME (AD=A)
    RESULT SETS #RESULT
    GIVING #RESPONSE

READ RESULT SET #RESULT INTO #NAME FROM DEMO-PERSON
    GIVING #RESPONSE
  DISPLAY #NAME
```

```
END-RESULT

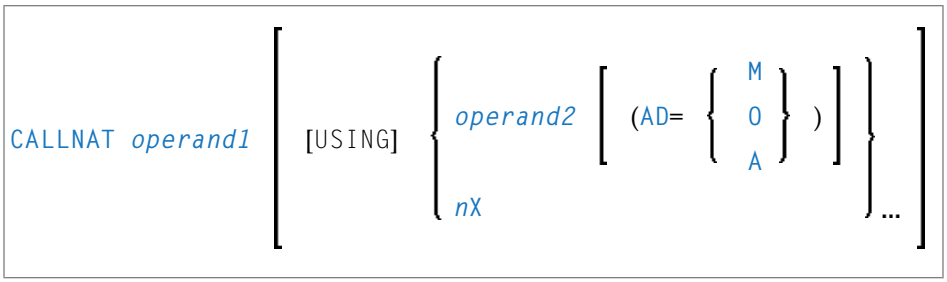
...

END
```

Weitere Beispiele siehe *Example of CALLDBPROC/READ RESULT SET* im Abschnitt *CALLDBPROC* in der *Natural for Db2*-Dokumentation.

18 CALLNAT

- Funktion CALLNAT 132
- Syntax-Beschreibung CALLNAT 133
- Übertragung von Parametern mit dynamischen Variablen 135
- Beispiele CALLNAT 136



Dieser Abschnitt 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 Subprogrammen***

Funktion CALLNAT

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 CALLNAT

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Subprogramm-Name:** Als *operand1* 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. Die Groß-/Kleinschreibung des Namens wird nicht verändert. 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.
operand2	**Parameter:** Werden Parameter an das Subprogramm übergeben, muss die Struktur der Parameterliste in einem DEFINE DATA PARAMETER-Statement definiert werden. Die mit dem CALLNAT-Statement angegebenen Parameter sind die einzigen Daten, die dem Subprogramm vom aufrufenden Objekt zur Verfügung stehen. 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 DEFINE DATA PARAMETER-Statement des Subprogramms mit der Option BY VALUE bzw. BY VALUE RESULT (siehe *parameter-data-definition* in der Beschreibung des DEFINE DATA-Statements). ■ 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 des Subprogramms entsprechen. Die Namen der Variablen im aufrufenden Objekt und im aufgerufenen Subprogramm können unterschiedlich sein.

Syntax-Element	Beschreibung
	■ Für die Übergabe der Parameterwerte selbst („By Value“) gilt: die Reihenfolge der Parameter im aufrufenden Objekt muss der Reihenfolge im DEFINE DATA PARAMETER-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 *operand2* eine Variable ist, können Sie sie wie folgt kennzeichnen: **AD=O** — Nicht modifizierbar, siehe Session-Parameter AD=O.

Syntax-Element	Beschreibung	
		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 *operand2* eine Konstante ist, kann der Session-Parameter AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.	
*n*X	Mit der Notation *n*X können Sie angeben, dass die nächsten *n* 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 *n* Parameter keine Werte an das Subprogramm übergeben werden. Der mögliche Wertebereich für *n* ist 1 - 4096. 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.	

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

operand2 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)

operand2 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 CALLNAT

- 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=O) #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 #FIELDC (A10)
1 #FIELDD (A7)
END-DEFINE
*
*
#FIELDA := 4711
*
#FIELDB := 'HALLO'
*
#FIELDC := 'ABC'
*
WRITE '=' #FIELDA '=' #FIELDB '=' #FIELDC '=' #FIELDD
*
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
```

19 CLOSE CONVERSATION

- Funktion CLOSE CONVERSATION 140
- Syntax-Beschreibung CLOSE CONVERSATION 140
- Weitere Informationen und Beispiele zu CLOSE CONVERSATION 141

```
                    ┌ {operand1} ... ┐
CLOSE CONVERSATION  { *CONVID       }
                    └ ALL            ┘
```

Dieser Abschnitt 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 CLOSE CONVERSATION

Das Statement `CLOSE CONVERSATION` wird im Zusammenhang mit dem Natural RPC (Remote Procedure Call) verwendet. Damit ist es möglich, Konversationen von der Client-Seite aus zu schließen. Sie können die aktuelle Konversation, eine andere offene Konversation oder alle offenen Konversationen schließen.

Anmerkung: Eine Anmeldung (Logon) bei einer anderen Library bewirkt kein automatisches Schließen von Konversationen.

Syntax-Beschreibung CLOSE CONVERSATION

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1		S	A						I							ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Zu schließende Konversation:** Um eine bestimmte offene Konversation zu schließen, geben Sie die Konversationskennung als *operand1* an. *operand1* muss eine Variable mit Format/Länge I4 sein.
*CONVID	**Aktuelle Konversation schließen:**

Syntax-Element	Beschreibung
	Um die aktuelle Konversation zu schließen, geben Sie `*CONVID` an. Die ID der aktuellen Konversation wird bestimmt durch den Wert der Systemvariablen `*CONVID`.
`ALL`	**Alle offenen Konversationen schließen:** Um alle offenen Konversationen zu schließen, geben Sie `ALL` an.

Weitere Informationen und Beispiele zu CLOSE CONVERSATION

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

- *Natural RPC-Betrieb im konversationellen Modus*
- *Verwendung des Natural RPC im konversationellen Modus*

III

- 20 CLOSE PC FILE 145
- 21 CLOSE PRINTER 149
- 22 CLOSE WORK FILE 153
- 23 COMMIT (SQL) 157
- 24 COMPOSE 159
- 25 COMPRESS 189
- 26 COMPUTE 201
- 27 CREATE OBJECT 211
- 28 DECIDE FOR 215
- 29 DECIDE ON 221
- 30 DEFINE CLASS 227

20 CLOSE PC FILE

- Funktion CLOSE PC FILE 146
- Syntax-Beschreibung CLOSE PC FILE 146
- Beispiel für CLOSE PC FILE-Statement 147

```
CLOSE { PC   } [FILE] work-file-number
      { WORK }
```

Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements: `DOWNLOAD PC FILE` | `UPLOAD PC FILE`

Funktion CLOSE PC FILE

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

Eine Arbeitsdatei wird auch automatisch geschlossen, wenn der Kommandomodus erreicht ist.

Es gelten die Einstellungen im `NTWORK`-Makro.

Weitere Einzelheiten siehe *Natural Connection*- und *Entire Connection*-Dokumentation.

Gehört zur Funktionsgruppe: ***Verarbeitung von Arbeitsdateien***

Syntax-Beschreibung CLOSE PC FILE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	Als *work-file-number* geben Sie die Nummer der zu schließenden PC-Arbeitsdatei an. Diese Nummer muss einer der Arbeitsdateinummern für den PC entsprechen (wie für Natural definiert).

Beispiel für CLOSE PC FILE-Statement

Das folgende Programm veranschaulicht die Benutzung des `CLOSE PC FILE`-Statements.

```
** Example 'PCCLEX1': CLOSE PC FILE
**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
01 W-DAT   (A40)
01 REC-NUM (N3)
01 I       (P3)
END-DEFINE
*
REPEAT
UPLOAD PC FILE 7 ONCE W-DAT                /* Data upload
  AT END OF FILE
    ESCAPE BOTTOM
  END-ENDFILE
  INPUT 'Processing file' W-DAT (AD=O)
    /   'Enter record number to display' REC-NUM
  IF REC-NUM = 0
    STOP
  END-IF
  FOR I = 1 TO REC-NUM
    UPLOAD PC FILE 7 ONCE W-DAT
    AT END OF FILE
      WRITE 'Max. record number reached, last record is'
      ESCAPE BOTTOM
    END-ENDFILE
  END-FOR
  I := I - 1
  WRITE 'Record' I ':' W-DAT
CLOSE PC FILE 7                            /* Close PC file 7
END-REPEAT
END
```

Ausgabe des Programms `PCCLEX1`:

Wenn Sie das Programm starten, erscheint ein Fenster, in dem Sie den Namen der PC-Datei angeben, von der die Daten hochgeladen werden sollen. Die Daten werden dann vom PC hochgeladen. Am Ende jeder Schleife wird die PC-Datei geschlossen.

21 CLOSE PRINTER

- Funktion CLOSE PRINTER 150
- Syntax-Beschreibung CLOSE PRINTER 150
- Beispiel für CLOSE PRINTER-Statement 151

```
CLOSE PRINTER  { (logical-printer-name) }
               { (printer-number)       }
```

Dieser Abschnitt 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 CLOSE PRINTER

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.

Wenn ein Drucker geschlossen wird, dann wird das zu dem Drucker gehörende Profil (siehe PROFILE**-Klausel** des DEFINE PRINTER-Statements) gelöscht, d.h., dies betrifft alle weiteren Schreibaktionen auf diesem Drucker.

Syntax-Beschreibung CLOSE PRINTER

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
logical-printer-name	**Logischer Drucker-Name:** Mit *logical-printer-name* geben Sie den Drucker an, der geschlossen werden soll. Der Name entspricht den Angaben in dem DEFINE PRINTER-Statement, in dem Sie den betreffenden Drucker definiert haben.

Syntax-Element	Beschreibung
	Für den *logical-printer-name* gelten die gleichen Namenskonventionen wie für Benutzervariablen; siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.
printer-number	**Drucker-Nummer:** Außer dem *logical-printer-name* können Sie auch die *printer-number* 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 für CLOSE PRINTER-Statement

```
** 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
```

22 CLOSE WORK FILE

- Funktion CLOSE WORK FILE 154
- Syntax-Beschreibung CLOSE WORK FILE 154
- Beispiel für CLOSE WORK FILE-Statement 155

```
CLOSE WORK [FILE] work-file-number
```

Dieser Abschnitt 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 CLOSE WORK FILE

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 ein anderes Dataset zuweist;
- entsprechend dem Schlüsselwortparameter `CLOSE` des Profilparameters `WORK`.

`CLOSE WORK FILE` wird für Arbeitsdateien ignoriert, für die im Profilparameter `WORK CLOSE=FIN` angegeben ist.

Syntax-Beschreibung CLOSE WORK FILE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Arbeitsdateinummer:** Gibt die für Natural definierte Nummer der Arbeitsdatei an, die gelesen werden soll. Die Nummer der Arbeitsdatei ist entweder ■ eine numerische Konstante im Wertebereich 1:32 oder

Syntax-Element	Beschreibung
	■ eine mit der Klausel CONSTANT definierte numerische Variable im Format B/N/P/I, die einen Wert im Bereich 1:32 zuweist. Dezimalstellen bei den Formaten (N/P) sind nicht erlaubt.

Beispiel für CLOSE WORK FILE-Statement

```
** 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=O)
      / '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
```

23 COMMIT (SQL)

- Funktion COMMIT (SQL) 158
- Hinweis für Nicht-Natural-Programme 158
- Beispiel COMMIT (SQL) 158

```
COMMIT
```

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Siehe auch *COMMIT - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion COMMIT (SQL)

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 datenbankändernden Verarbeitungsschleife stehen, sondern muss außerhalb einer solchen stehen (bzw. bei geschachtelten Schleifen nach der äußersten Schleife).

Hinweis für Nicht-Natural-Programme

Wenn ein Natural-Programm ein externes Nicht-Natural-Programm aufruft, sollte das aufgerufene Programm kein eigenes `COMMIT`-Statement enthalten, falls das aufrufende Natural-Programm selbst auch Datenbankaufrufe durchführt. In diesem Falle sollte das Natural-Programm das `COMMIT`-Statement für das externe Nicht-Natural-Programm enthalten.

Beispiel COMMIT (SQL)

```
...
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'
COMMIT
...
```

24 COMPOSE

- Funktion COMPOSE 160
- Syntax-Beschreibung COMPOSE 161
- Der Formatiervorgang 175
- Verarbeitung im Dialog-Modus 176
- Eingabe-/Ausgabe-Verarbeitung durch Nicht-Natural-Programme 178
- Beispiele COMPOSE 179

Dieses Statement kann nur verwendet werden, wenn das Bürosystem Con-nect und das Textformatierungssystem Con-form installiert sind.

```
COMPOSE
  [RESETTING-clause]
  [MOVING-clause]
  [ASSIGNING-clause]
  [FORMATTING-clause]
  [EXTRACTING-clause]
```

Wenn Sie mehr als eine Klausel (*clause*) angeben, werden diese in der oben abgebildeten Reihenfolge verarbeitet.

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

Funktion COMPOSE

Mit dem Statement `COMPOSE` können Sie die Textformatierung durch Con-form (das von Con-nect verwendete Textformatiersystem) direkt von einem Natural-Programm aus auslösen.

Der zu formatierende Text kann entweder mittels Variablen zur Verfügung gestellt werden oder aus einem Con-nect-Textblock (einem Dokument, das Con-form-Formatierbefehle enthält) abgerufen werden.

Die Inhalte von Natural-Variablen können als Variablen an Con-form übergeben werden und dynamisch in den formatierten Text eingefügt werden.

Die Werte einer Con-form-Variablen können auch vom Textformatiersystem wieder an das Natural-Programm übergeben werden.

Wenn die Con-form-Anweisungen ausgeführt worden sind (d.h. wenn ein formatiertes Dokument erstellt wurde), wird die Ausgabe an eines der folgenden Ziele geleitet:

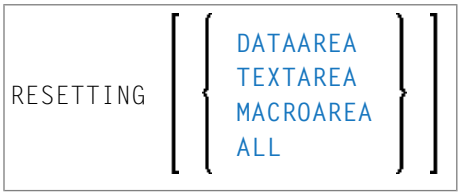
- einen Natural-Report,
- ein Dokument in der Con-nect-Systemdatei,
- Variablen in dem Natural-Programm, welches das `COMPOSE`-Statement ausführt,
- ein Nicht-Natural-Programm.

Syntax-Beschreibung COMPOSE

- RESETTING-Klausel
- MOVING-Klausel
- ASSIGNING-Klausel
- FORMATTING-Klausel
- EXTRACTING-Klausel

RESETTING-Klausel

Mit dieser Klausel können Sie Informationen aus dem Textformatierbereich des Con-form-Buffer löschen und Speicherplatz im Con-form-Buffer freigeben, welcher mit dem Profilparameter `CSIZE` im Natural-Parametermodul zugewiesen wird.



Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`DATAAREA`	Löscht alle aktiven Textvariablen.
`TEXTAREA`	Löscht alle Texteingabedaten. **Anmerkung:** Aus Kompatibilitätsgründen bezieht sich das Wort `TEXTAREA` auf `DATAAREA`, so wie es in der `MOVING`-Klausel verwendet wird.
`MACROAREA`	Löscht alle Textmakros.
`ALL`	Löscht alle aktiven Textvariablen, Texteingabedaten und Textmakros.

Siehe auch *Beispiel 1* und *Beispiel 2*.

MOVING-Klausel

Mit dieser Klausel können Sie Textzeilen in den Textformatierbereich des Con-form-Buffer oder direkt in das Formatierungssystem übertragen und die formatierte Textausgabe vom Con-form-Buffer abrufen.

Mit dieser Klausel können ein oder mehrere Textwerte in den Textformatierbereich übertragen werden (siehe *Syntax 1*). Dieser Bereich kann als Eingabequelle für Formatieroperationen verwendet werden.

Wartet das Textformatierungssystem gerade auf Eingaben (siehe *Verarbeitung im Dialog-Modus*), wird der Text direkt an es übergeben, ohne im Con-form-Buffer zwischengespeichert zu werden (siehe *Syntax 1* und *Syntax 2*). Die Source-Eingabe wird mit der `LAST`-Option beendet.

Wartet der formatierte Text gerade darauf, ausgegeben zu werden (siehe *Verarbeitung im Dialog-Modus*), dann dient *Syntax 3* der `MOVING`-Klausel dazu, die Kontrolle vom Natural-Programm wieder an das Textformatierungssystem zu übergeben.

Zur Beschreibung der Statusvariablen siehe *FORMATTING-Klausel*.

Je nach Status der Dialog-Modus-Verarbeitung können Sie eine der folgenden Formen der `MOVING`-Klausel verwenden:

Syntax 1 - Dialogmodus für Eingabe

Syntax 1 der `MOVING`-Klausel gilt, wenn die Formatierung noch nicht begonnen hat oder das Textformatierungssystem im Dialog-Modus für Eingabe ist und auf Eingabedaten wartet (`TERM` in der Statusvariablen **„State“**).

```
MOVING [operand1] ... 37  [TO DATAAREA] [LAST]
[STATUS [TO] operand2  [operand3 [operand4  [operand5]]]]
```

Syntax 2 - Dialogmodus für Ein- und Ausgabe

Syntax 2 der `MOVING`-Klausel gilt, wenn das Textformatierungssystem im Dialog-Modus für Ein- und Ausgabe ist und auf weitere Eingabedaten wartet (Status `TERM` in der Statusvariablen **„State“**). In diesem Modus akzeptiert der Formatierer jeweils nur eine Eingabezeile.

Der Ausführungskontext kann sich zwischen einer Reihe ausgeführter `COMPOSE`-Statements ändern. Daher müssen die Ausgabevariablen erneut angegeben werden, selbst wenn der Formatierer auf Eingabedaten wartet.

```
MOVING [ { operand1 [TO DATAAREA] } ]  [OUTPUT] TO VARIABLES
       [ { LAST                   } ]  operand6 ... 20
[STATUS [TO] operand2 [operand3 [operand4 [operand5]]]]
```

Syntax 3 - Dialogmodus für Ausgabe

Syntax 3 der `MOVING`-Klausel gilt, wenn das Textformatierungssystem im Dialog-Modus für Ausgabe (und möglicherweise gleichzeitig für Eingabe) ist und die Ausgabe an das Natural-Programm übergibt (Status `STRG` in der Statusvariablen **„State“**).

```
MOVING OUTPUT [TO VARIABLES] operand6 ... 20
[STATUS [TO] operand2 [operand3 [operand4 [operand5]]]]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A			A	N	P								ja	nein
operand2		S				A										ja	ja
operand3		S								B						ja	ja
operand4		S								B						ja	ja
operand5		S								B						ja	ja
operand6		S	A			A										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	Enthält die (unformatierten) Eingabe-Textzeilen. Format/Länge: (A*n*), dabei ist *n* maximal 253; (N*n*) oder (P*n*), dabei ist *n* maximal 29
operand2	Enthält die Statusvariable **„State“** (Status der Formatierung). Format/Länge: (A4)
operand3	Enthält die Statusvariable **„Position“** (Seitenzahl). Format/Länge: (B4)
operand4	Enthält die Statusvariable **„Position“** (Zeilennummer). Format/Länge: (B4)
operand5	Enthält die Statusvariable **„Amount of Output Data“** (Menge der Ausgabedaten, d.h. Anzahl der Zeilen). Format/Länge: (B4)

Syntax-Element	Beschreibung
operand6	Enthält die (formatierten) Ausgabe-Textzeilen. Format/Länge: (A*n*), dabei ist *n* maximal 253

ASSIGNING-Klausel

Mit dieser Klausel können Sie Werte von Natural-Variablen zu Con-form-Textvariablen zuweisen. Anschließende Formatiervorgänge können sich auf diese Textvariablen beziehen.

```
ASSIGNING [TEXTVARIABLE] {operand1=operand2}, ... 19
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A										ja	nein
operand2	C	S				A	N	P								ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	*operand1* ist der Name der Con-form-Textvariablen. Die Namen der Textvariablen müssen in Großbuchstaben angegeben werden. Format/Länge: (A*n*), dabei ist *n* maximal 253
operand2	*operand2* ist der Name einer gegebenen Natural-Variablen. Format/Länge: (A*n*), dabei ist *n* maximal 253; (N*n*) oder (P*n*), dabei ist *n* maximal 29

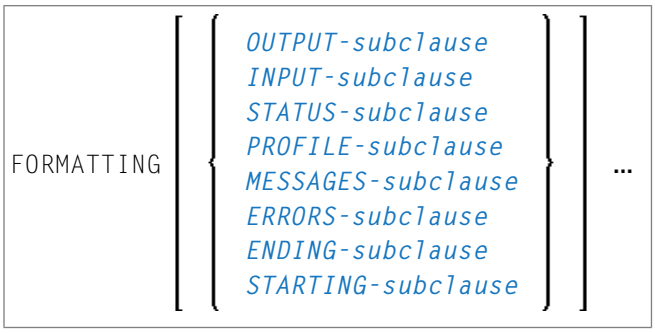
Siehe auch ***Beispiel 3*** und ***Beispiel 4***.

FORMATTING-Klausel

Mit dieser Klausel können Sie Con-form dazu veranlassen, eine formatierte Ausgabe zu erzeugen. Diese Klausel dient dazu, Text in der endgültig formatierten Form zu erstellen, d.h. mit korrekten Zeilen- und Seitenumbrüchen, unter Verwendung von Eingabedaten, die aus einer Kombination von Text und Con-form-Statements bestehen können.

Die Formatier-Optionen werden in einer oder mehreren Subklauseln (*subclauses*) angegeben. Falls keine Subklauseln angegeben werden, nimmt Con-form die gültigen Standard-Formatieroptionen.

Die Statusvariable wird im Dialog-Modus verwendet.



Syntax-Element-Beschreibung:

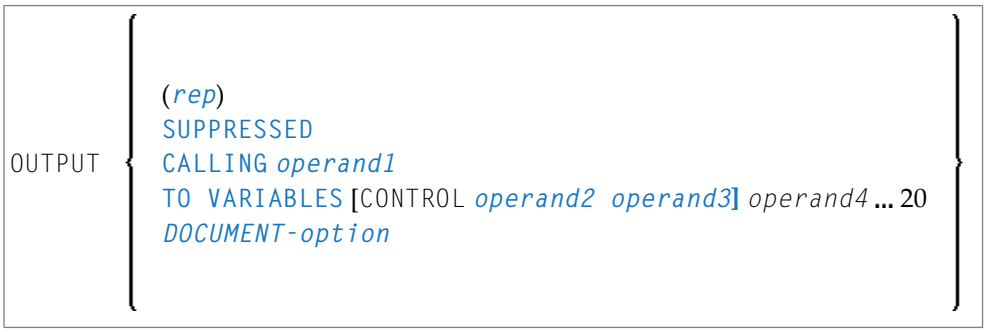
Syntax-Element	Beschreibung	
OUTPUT-subclause	Das Ausgabemedium. Dies kann ein Natural-Report, ein Dokument in einem Con-nect-Büro, eine oder mehrere Natural-Variablen (oder ein Array von Natural-Variablen) oder ein Nicht-Natural-Programm sein. Siehe ***Output-Subklausel***.	
INPUT-subclause	Das Eingabemedium. Dies kann ein Con-nect-Dokument, sowie auch der Textformatierbereich des Con-form-Buffer (siehe `DATAAREA` in der ***MOVING-Klausel***), die Umgebung des Natural-Programms (auch mehrere), welches die `COMPOSE`-Statements ausführt (siehe auch ***MOVING-Klausel***), ein Nicht-Natural-Programm oder eine Kombination dieser vier Möglichkeiten sein.	
STATUS-subclause	Der Status des Formatiervorgangs. Der Formatiervorgang kann die mehrfache Ausführung eines `COMPOSE`-Statements umfassen (bei ***Verarbeitung im Dialog-Modus***). Zum Beispiel wird die Eingabe von einem Natural-Programm in den Textformatierbereich gestellt, und die Ausgabe wird vom Textformatierbereich an die Umgebung eines Natural-Programms (d.h. an eine oder mehrere Natural-Variablen) übergeben. Deshalb muss das Natural-Programm über den Status der Formatierung informiert werden. Folgende Variablen werden während der Formatierung an das Natural-Programm übergeben:	
	State	Status:
		`TERM` - wenn der Dialog-Modus für die Eingabe bereit ist.
		`STRG` - wenn der Dialog-Modus für die Ausgabe bereit ist.
		`END` - wenn die Formatierung erfolgreich beendet wurde.
		`ENDX` - wenn die Formatierung nicht erfolgreich beendet wurde.
	Position	Seiten- und Zeilennummer des Dokuments, das gerade formatiert wird. Diese Angaben werden in zwei separaten Variablen (Seitenposition und Zeilenposition) gehalten.

Syntax-Element	Beschreibung	
	Amount of Output Data	Die Anzahl formatierter Ausgabezeilen, die an das Natural- Programm übergeben werden. Der Formatierer benutzt diese Zahl als Zeiger auf die nächste zu füllende Ausgabevariable. Der Wert erhöht sich um 1, bevor die Ausgabezeile ausgegeben wird. Liegt der aktuelle Wert außerhalb des Bereichs, wird der Wert auf 1 gesetzt.
PROFILE-subclause	Der Textblock, der vor den Eingaben verarbeitet wird. Siehe ***PROFILE-Subklausel***.	
MESSAGES-subclause *ERRORS-subclause*	Steuert die Ausgabe von Warnmeldungen und statistischen Informationen und die Fehlerverarbeitung. Siehe ***MESSAGES-Subklausel*** und ***ERRORS-Subklausel***.	
ENDING-subclause	Dient zur Festlegung der Seite, bis zu der die Ausgabe des formatierten Textes erfolgen soll. Siehe ***ENDING-Subklausel***.	
STARTING-subclause	Dient zur Festlegung der Seite, ab der die Ausgabe des formatierten Textes erfolgen soll. Siehe ***STARTING-Subklausel***.	

OUTPUT-Subklausel

Mit dieser Subklausel können Sie den formatierten Con-form-Text an eine bestimmte Ausgabestelle schicken.

Wird diese Subklausel weggelassen, dann wird die Ausgabe standardmäßig an den Natural-Hauptdrucker geschickt.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A												ja	nein
operand2	C	S				A												ja	nein
operand3	C	S				A												ja	nein
operand4		S	A			A												ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`OUTPUT (`*rep*`)`	Wenn die Ausgabe an einen Drucker geschickt wird (d.h. wenn die Reportnummer nicht 0 ist) und ein Con-nect-Druckerprofil (mit der Con-nect-API-Funktion `Z-DRIVER`) geladen wurde, werden die Optionen zur Hervorhebung von Teilen des formatierten Ausgabetexts über die Einstellungen in diesem Profil gesteuert. Wenn ein Druckerprofil aktiv ist und keine logische Seitenvorschubsteuerung angegeben wurde, werden Seitenvorschübe über die entsprechenden internen Natural-Nukleusfunktionen eingefügt. Weitere Texthervorhebungsoptionen, die im gerade aktiven Con-nect-Druckerprofil nicht definiert sind, werden ignoriert. **Anmerkung:** Bei der Ausführung eines `COMPOSE RESETTING ALL`-Statements oder eines `COMPOSE FORMATTING`-Statements, bei dem die Ausgabe nicht an einen Report geschickt wird, wird ein Druckerprofil aus dem Textformatierbereich entladen. Wird die Ausgabe an Report 0 geschickt oder ist kein Druckerprofil aktiv, übergibt Con-nect die Verantwortung für die Ausgabeverarbeitung an die Natural-Nukleus-Routinen. In diesem Falle werden nur die Texthervorhebungsoptionen Fettdruck, Unterstreichung und Kursivschrift berücksichtigt. **Anmerkung:** Ein Report, der in einem Statement `DEFINE PRINTER (`*n*`) OUTPUT 'CONNECT'` referenziert wird, darf nicht als Ausgabeziel in einem `COMPOSE FORMATTING`-Statement angegeben werden.
`OUTPUT SUPPRESSED`	Mit dieser Option können Sie die Ausgabe unterdrücken.
`OUTPUT CALLING` *operand1*	*operand1* ist der Name des Programms, das aufgerufen wird. Format/Länge: (A*n*), dabei ist *n* maximal 8 Siehe Abschnitt *Eingabe-/Ausgabe-Verarbeitung durch Nicht-Natural-Programme*.
`OUTPUT TO VARIABLES [CONTROL` *operand2 operand3*`]` *operand4*	Normalerweise wird der formatierte Text in seiner endgültigen Form an ein Array von Natural-Variablen übergeben. Jede Zeile füllt eine Variable (wenn eine Zeile nicht ganz in eine Variable passt, wird sie abgeschnitten). Texthervorhebungsoptionen werden ignoriert, mit Ausnahme angegebener `CONTROL`-Variablen, die dazu dienen, Textteile hervorzuheben (d.h. fett oder unterstrichen auszugeben).

Syntax-Element	Beschreibung
	Wenn die CONTROL-Variablen I und N angegeben werden, wird formatierter Text in einem Zwischenformat erzeugt (d.h. durchsetzt mit logischen Steuersequenzen). *operand2* ist das Anfang-Zeichen. Format/Länge: (A1). *operand3* ist das Ende-Zeichen. Format/Länge: (A1). Beispiel mit spitzen Klammern als Anfang- und Ende-Zeichen: `<ABC...XYZ>` *operand4* ist das Ausgabefeld. Format/Länge: (A*n*), dabei ist *n* maximal 253 Weitere Informationen siehe Abschnitt ***Verarbeitung im Dialog-Modus* und speziell Abschnitt *Dialog-Modus für Ausgabe***.
OUTPUT *DOCUMENT-option*	Siehe ***DOCUMENT-Option***.

DOCUMENT-Option

Mit der DOCUMENT-Option der OUTPUT-Subklausel können Sie den formatierten Text von Con-form in fertig formatierter Form (Txt) oder im Zwischenformat (Int) an ein Con-nect-Büro schicken. Der Text im Zwischenformat (Int) kann nicht verändert werden.

```
DOCUMENT { INTO [ { FINAL        } ] [CABINET] operand1 [PASSW=operand2] }
         {      [ { INTERMEDIATE } ]                                     }
         { [GIVING] { operand3 [operand4] }                              }
         {          { operand4 [operand3] }                              }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A											ja	nein
operand2		S				A											ja	nein
operand3		S								B							ja	ja
operand4		S								B							ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
DOCUMENT INTO FINAL | INTERMEDIATE CABINET	Wenn Sie das Schlüsselwort FINAL angeben, wird das Dokument in fertig formatierter Form erstellt. In diesem Fall werden spezifische Texthervorhebungsoptionen, z.B. Fettdruck oder Kursivschrift) ignoriert Wenn Sie das Schlüsselwort INTERMEDIATE angeben, wird das Dokument ohne Namen in das Fach COMPOSE gestellt. In der Beschreibungszeile steht der Name des Programms, welches das COMPOSE FORMATTING-Statement ausgeführt hat, sowie Datum und Uhrzeit der Ausführung.
CABINET *operand1*	Für *operand1* können Sie ein spezielles Büro angeben. Format/Länge: (A*n*), dabei ist *n* maximal 8 Wenn Sie *operand1* nicht benutzen, wird das Dokument in das aktuelle Benutzerbüro gestellt (d.h. das Büro, dessen ID der des gerade aktiven Natural-Benutzers entspricht). Con-form erzwingt, dass Con-nect-Zugriffsbeschränkungen eingehalten werden, und akzeptiert nur Büro-IDs, die in Con-nect definiert sind. **Anmerkung:** Büro-IDs müssen in Großbuchstaben angegeben werden.
PASSW=*operand2*	Ein Passwort müssen Sie dann angeben, wenn Sie ein Dokument in einem Büro speichern wollen, auf das Sie als angenommener aktueller Benutzer keinen Zugriff haben. Format/Länge: (A*n*), dabei ist *n* maximal 8
operand3	*operand3* wird vom Textformatierungssystem verwendet, um einen eindeutigen Schlüssel vom Dokument an das Natural-Programm zurückzugeben. Er wird nur aus Kompatibilitätsgründen unterstützt. Format/Länge: (B10)
operand4	*operand4* wird vom Textformatierungssystem verwendet, um eine ISN, die auf das formatierte Ausgabedokument zeigt, an das Natural-Programm zurückzugeben. Diese ISN kann bei der Referenzierung eines Dokuments in aufeinanderfolgenden Aufrufen von Con-nect-APIs nützlich sein. Format/Länge: (B4)

INPUT-Subklausel

Mit dieser Subklausel können Sie angeben, woher die Eingaben für das Textformatierungssystem kommen..

Anmerkungen:

1. Ohne diese Subklausel wird standardmäßig DATAAREA (Textformatierbereich von Con-form) verarbeitet.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A											ja	nein
operand2	C	S				A											ja	nein
operand3		S				A											ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
DATAAREA	Die Eingaben können aus dem Textformatierbereich kommen (eine Kombination aus Text aus dem Textformatierbereich und aus dem **Dialog-Modus** ist ebenfalls möglich), der mit einer oder mehreren MOVING-Operationen gefüllt werden muss. Siehe*MOVING-Klausel*.
operand1	Alternativ können die Eingaben auch von einem Textblock genommen werden. Der Name des Textblocks wird mit *operand1* angegeben. **Anmerkung:** Textblock-IDs müssen in Großbuchstaben angegeben werden. Format/Länge: (A*n*), dabei ist *n* maximal 253 Der Textblock kann in einem Con-nect-Büro enthalten sein oder über ein **Nicht-Natural-Programm** zur Verfügung gestellt werden. Beim Aufrufen eines solchen Programms gelten die für das CALL-Statement gültigen Konventionen. Es kann eine Hierarchie von Con-nect-Büros oder Nicht-Natural-Programmen angegeben werden, von denen jede nacheinander nach dem in *operand1* benannten Textblock abgesucht wird.
CABINET *operand2*	Die Eingaben (ein mit *operand1* angegebener Textblock) können aus einem bestimmten Con-nect-Büro entmommen werden. *operand2* ist der Name des Con-nect-Büros. **Anmerkung:** Büro-IDs müssen in Großbuchstaben angegeben werden. Format/Länge: (A*n*), dabei ist *n* maximal 8
EXIT *operand2*	*operand2* ist der Name des Exit. Format/Länge: (A*n*), dabei ist *n* maximal 8

Syntax-Element	Beschreibung
PASSW=*operand3*	Ein Passwort muss angegeben werden, wenn ein Dokument in einem Büro gespeichert wird, auf das unter der aktuellen Benutzerkennung kein Zugriff erlaubt ist. Format/Länge: (A*n*), dabei ist *n* maximal 8 Con-form erzwingt, dass Con-nect-Zugriffsbeschränkungen eingehalten werden, und akzeptiert nur Büro-IDs, die in Con-nect definiert sind.

Siehe auch ***Beispiel 4***.

STATUS-Subklausel

Die in der FORMATTING-Klausel verwendete STATUS-Subklausel entspricht der STATUS-Subklausel, die in der MOVING-Klausel vorhanden ist. Sie sollten sie verwenden, um sicherzustellen, dass das der Formatierungsvorgang immer den passenden Status für den gegebenen Verarbeitungsschritt hat.

```
[STATUS operand1 [operand2 [operand3 [operand4]]]]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
Operand1		S				A										ja	nein
Operand2		S								B						ja	nein
Operand3		S								B						ja	nein
Operand4		S								B						ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	Enthält die Statusvariable **„State“** (Status der Formatierung). Format/Länge: (A4)
operand2	Enthält die Statusvariable **„Position“** (Seitenzahl). Format/Länge: (B4)
operand3	Enthält die Statusvariable **„Position“** (Zeilennummer). Format/Länge: (B4)
operand4	Enthält die Statusvariable **„Amount of Output Data“** (Menge der Ausgabedaten, d.h. Anzahl der Zeilen). Format/Länge: (B4)

PROFILE-Subklausel

Diese Subklausel bewirkt, dass der Inhalt des angegebenen Textblocks verarbeitet wird, bevor in der `INPUT`-**Subklausel** angegebene Eingaben verarbeitet werden (standardmäßig wird ein Textblock nicht als Profil verarbeitet).

```
PROFILE operand1
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	*operand1* ist der Name des Textblocks(z.B. `FPROFILE` - Formatierungsprofil). Format/Länge: (A*n*), dabei ist *n* maximal 32

MESSAGES-Subklausel

Mit dieser Subklausel können Sie bestimmen, ob bei Beendigung der Formatierung die Ausgabe von Warnmeldungen und statistische Informationen erfolgt oder die Ausgabe unterdrückt wird (der Fehler wird ignoriert).

```
MESSAGES { [LISTED] [ON] (rep) }
         { SUPPRESSED          }
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(`*rep*`)`	Mit der `(`*rep*`)`-Notation geben Sie den Report an, für den die `MESSAGES`-Subklausel gültig ist. Sie können einen Wert im Bereich von `0 - 31` angeben. Der Wert für `(`*rep*`)` kann auch ein logischer Name sein, der mittels des `DEFINE PRINTER`-Statements zugewiesen wurde Wie Sie das Format eines mit Natural erstellten Report steuern können ist im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung* beschrieben.
`SUPPRESSED`	Wenn Sie das Schlüsselwort `SUPPRESSED` angeben, werden keine Meldungen ausgegeben und Fehler werden ignoriert.

ERRORS-Subklausel

Mit dieser Subklausel können Sie bestimmen, was im Falle eines Formatierungsfehlers geschehen soll. Der Fehler kann von einer Natural-Standard-Fehlerroutine verarbeitet oder in einem bestimmten Natural-Report (*rep*) aufgelistet werden.

```
ERRORS { [LISTED] [ON] (rep) }
       { INTERCEPTED         }
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(*rep*)	Mit der (*rep*)-Notation geben Sie den Report an, für den die ERRORS-Subklausel gültig ist. Sie können einen Wert im Bereich von 0 - 31 angeben. Der Wert für (*rep*) kann auch ein logischer Name sein, der mittels des DEFINE PRINTER-Statements zugewiesen wurde Wie Sie das Format eines mit Natural erstellten Report steuern können ist im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung* beschrieben.
INTERCEPTED	Durch Angabe des Schlüsselwortes INTERCEPTED veranlassen Sie, dass Fehler durch die Standard-Fehlerroutine von Natural verarbeitet werden.

Anmerkung: Fehler und Meldungen schließen einander aus. Bei manchen Fehlern wird die Natural-Fehlerroutine aufgerufen, selbst wenn eine andere Option angegeben wurde. Fehler und Meldungen dürfen nicht an einen Report geschickt werden, der über das Statement DEFINE PRINTER (*n*) OUTPUT 'CONNECT' an Con-nect geleitet wird.

ENDING-Subklausel

Diese Subklausel bewirkt, dass die Ausgabe des formatierten Textes nach einer bestimmten Seite (Seitennummer) unterdrückt wird bzw. die Ausgabe sich auf eine festgelegte Anzahl von Seiten beschränken soll.

```
ENDING { [AT] [PAGE] operand1          }
       { AFTER operand1 [PAGES]        }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S					N	P								ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`ENDING [AT] [PAGE] operand1`	Bewirkt, dass die Ausgabe des formatierten Textes *nach* einer bestimmten Seite unterdrückt wird. Die Angabe dieser Seite erfolgt in *operand1*. Format/Länge: (N*n*) oder (P*n*), dabei ist *n* maximal 5
`ENDING AFTER operand1 [PAGES]`	Gibt an, dass die Ausgabe sich auf eine in *operand1* festgelegte Anzahl von Seiten beschränken soll.

STARTING-Subklausel

Die Subklausel bewirkt, dass die Ausgabe des formatierten Textes solange unterdrückt wird, bis die angegebene Seite erreicht ist.

```
STARTING [FROM] [PAGE] operand1
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S					N	P								ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	Die Ausgabe des formatierten Textes wird solange unterdrückt wird, bis die in *operand1* angegebene Seite erreicht ist. Format/Länge: (N*n*) oder (P*n*), dabei ist *n* maximal 5

EXTRACTING-Klausel

Mit dieser Klausel können Sie Natural-Variablen die Werte von Con-form-Textvariablen zuweisen. Die aktuellen Textvariablenwerte können das Ergebnis von vorhergehenden Formatiervorgängen sein.

```
EXTRACTING [TEXTVARIABLE] {operand1=operand2},... 19
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1		S				A	N	P								ja	ja
operand2	C	S				A										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	*operand1* ist der Name einer gegebenen Natural-Variablen. Format/Länge: (A*n*), dabei ist *n* maximal 253; (N) oder (P*n*), dabei ist *n* maximal 29
operand2	*operand2* ist der Name einer Con-form-Textvariablen. Der Name muss in Großbuchstaben angegeben werden. Format/Länge: (A*n*), dabei ist *n* maximal 253

Siehe auch ***Beispiel 6***.

Der Formatiervorgang

Der Formatiervorgang beginnt, wenn die FORMATTING-Klausel des COMPOSE-Statements ausgeführt wird (unter Umständen auch dann, wenn bei beabsichtigter Texteingabe durch die MOVING-Klausel noch keine solche Eingabe vorliegt). Solange der Formatiervorgang aktiv ist, werden Texteingaben, die aus der Ausführung des COMPOSE MOVING-Statements resultieren, direkt zur Formatierung übertragen (und können in späteren Formatiervorgängen nicht wiederverwendet werden). Ist der Formatiervorgang nicht aktiv, werden Texteingaben im Textformatierbereich des Con-form-Buffer zwischengespeichert. Dadurch ist es möglich, die Eingabe für mehrere Formatiervorgänge wiederzuverwenden.

Da der Con-form-Buffer am Ende des Natural-Programms nicht gelöscht wird, brauchen die betreffenden COMPOSE-Statements nicht alle in demselben Natural-Programm zu stehen; sie können auch in mehreren nacheinander aufgerufenen Programmen stehen.

Die Ausführung einer RESETTING- oder FORMATTING-Klausel oder das Auftreten eines schwerwiegenden Formatierfehlers beenden den aktiven Formatierdurchlauf.

Das Ende der Eingabedaten wird durch die LAST-Subklausel der MOVING-Klausel bestimmt.

Wenn ein Con-nect-Dokument als Eingabequelle verwendet wird, wird davon ausgegangen, dass das Ende der Eingabedaten am Ende des Dokuments erreicht ist.



Anmerkung: Es empfiehlt sich, die STATUS-Subklausel der FORMATTING- bzw. MOVING-Klausel zu verwenden, um sicherzustellen, dass sich der Formatiervorgang jeweils im richtigen Status für den betreffenden Verarbeitungsschritt befindet.

Verarbeitung im Dialog-Modus

Bei der Verarbeitung im Dialog-Modus erfolgt eine Reihe von Interaktionen zwischen dem Benutzerprogramm und dem Textformatierungssystem, die während der Formatierung der Eingabe und der Erzeugung der Ausgabe stattfinden.

Der Dialog-Modus erlaubt es dem Benutzerprogramm, auf jeder Stufe der Eingabehierarchie dem Textformatierungssystem Rohtext als Eingabe zu liefern. In diesem Modus werden auch formatierte Ausgaben direkt aus der aktuellen Programmumgebung angenommen.

Der Dialog kommt dadurch zustande, dass der Formatiervorgang in eine Reihe von Schritten unterteilt wird, von denen jeder einzelne durch ein eigenes COMPOSE-Statement aufgerufen wird:

- Dialog-Modus für Eingabe
- Dialog-Modus für Ausgabe
- Dialog-Modus für Ein- und Ausgabe
- Ausführung von COMPOSE-Statements im Dialog-Modus

Dialog-Modus für Eingabe

Der Dialog-Modus für Eingabe wird aktiviert, wenn DATAAREA (Textformatierbereich) die Quelle des Eingabetextes ist, oder wenn die Con-form-Anweisung .TE ON auftaucht, und der Textformatierbereich keinen weiteren zu verarbeitenden Text mehr enthält. Der Dialog-Modus für Eingabe wird durch das Wort TERM in der Statusvariablen **„State“** angezeigt.

Das Benutzerprogramm sollte daraufhin die erforderlichen Eingaben liefern, indem es die MOVING-Klausel eines anschließend verarbeiteten COMPOSE-Statements aufruft. Das Benutzerprogramm kann die Terminaleingaben beenden, indem es die LAST-Option der MOVING-Klausel oder die Formatieranweisung .TE OFF verwendet, falls der Dialog-Modus durch .TE ON begonnen wurde, und zwar als Text mittels der MOVING-Klausel. Das Ende der Formatierung wird durch END (bzw. ENDX im Falle eines Fehlers) in der Statusvariablen **„State“** angezeigt.

Siehe auch ***Beispiel 7***.

Dialog-Modus für Ausgabe

Der Dialog-Modus für Ausgabe wird aktiviert, wenn das Ausgabeziel `TO VARIABLES` ist. Das Textformatierungssystem gibt die Kontrolle an das Natural-Programm zurück, sobald die angegebenen Natural-Variablen gefüllt sind oder ein Seitenende erreicht ist (je nachdem, was zuerst der Fall ist). Der Dialog-Modus für Ausgabe wird durch `STRG` in der `STATUS`-Variablen **„State“** angezeigt.

Das Benutzerprogramm sollte daraufhin die soeben in die Natural-Variablen gestellte formatierte Ausgabe übernehmen und einen weiteren Satz Natural-Variablen als Ausgabeziel in einem anschließend verarbeiteten `COMPOSE MOVING`-Statement bereitstellen. Das Ende der Formatierung wird durch `END` (bzw. `ENDX` im Falle eines Fehlers) in der Statusvariablen **„State“** angezeigt.



Anmerkung: Bei Verwendung des Dialog-Modus (siehe `INPUT`- und `OUTPUT`-Subklauseln) erstreckt sich der Formatiervorgang in der Regel über mehrere Ausführungen eines `COMPOSE`-Statements.

Dialog-Modus für Ein- und Ausgabe

Der Dialog-Modus kann auch für kombinierte Ein- und Ausgabeverarbeitung verwendet werden. Wenn das Textformatierungssystem weitere Eingabedaten anfordert (angezeigt durch den Status `TERM` in der Statusvariablen **„State“**) oder eine Ausgabe erzeugt hat (angezeigt durch den Status `STRG`), muss das Natural-Programm entsprechend reagieren.

Im Dialog-Modus für kombinierte Ein- und Ausgabeverarbeitung kann das Textformatierungssystem jeweils nur eine Eingabezeile annehmen. Lediglich im reinen Eingabe-Modus kann es mehrere Zeilen gleichzeitig annehmen.

Ausführung von COMPOSE-Statements im Dialog-Modus

Wie bereits erwähnt, wird der Dialog-Modus über ein `COMPOSE FORMATTING`-Statement aktiviert, welches eine Reihe von `COMPOSE MOVING`-Ausführungen umfasst. Beachten Sie dabei bitte folgendes:

- `COMPOSE ASSIGNING`- und `COMPOSE EXTRACTING`-Statements sind nur bei aktivem Dialog-Modus gültig.
- `COMPOSE RESETTING`- und `COMPOSE FORMATTING`-Statements erzwingen die sofortige Beendigung aller Formatierungsarbeiten.

Eingabe-/Ausgabe-Verarbeitung durch Nicht-Natural-Programme

In Abhängigkeit von den in der FORMATTING-Klausel angegebenen Parametern ist es möglich, Ein- und Ausgaben mit Nicht-Natural-Programmen zu verarbeiten. Solche Programme werden über den gleichen Mechanismus wie beim Natural-CALL-Statement aufgerufen.

Das COMPOSE-Statement tauscht mit diesen Programmen Parameter aus, und zwar unter Verwendung von **Standard-Linkage-Konventionen** (dynamisches Laden ist in einer CICS-Umgebung nicht möglich).



Anmerkung: Verarbeitung der Ein-/Ausgaben durch Nicht-Natural-Programme ist nur auf Großrechnern möglich; auf anderen Plattformen werden die entsprechenden Teile des COMPOSE-Statements ignoriert.

In Abhängigkeit vom Status des Formatiervorgangs werden zwei oder drei Parameter zwischen dem Formatierer und den Nicht-Natural-Programmen übergeben.

Parameter 1 (Format/Länge A1)	Der Funktionscode wird vom Formatierer an Nicht-Natural-Programme übergeben. Mögliche Werte:	
	I	Initiieren (Eingabe, Ausgabe)
	O	Dokument öffnen (Eingabe)
	R	Eine Zeile des Dokuments lesen (Eingabe)
	W	Eine Ausgabezeile schreiben (Ausgabe)
	C	Dokument schließen (Eingabe)
	T	Beenden (Eingabe, Ausgabe).
Parameter 2 (Format/Länge B1)	Der Response Code wird von Nicht-Natural-Programmen an das Formatierungssystem übergeben. Mögliche Werte:	
	X'00'	Funktion erfolgreich ausgeführt.
	X'01'	Bei Funktion O: Dokument nicht gefunden.
		Bei Funktion R: Ende des Dokuments erreicht.
	X'FF'	Funktion nicht beendet.
Parameter 3 (Format A1/256)	Bei den Funktionen O und W werden diese Parameter vom Formatierungssystem an Nicht-Natural-Programme übergeben. Dagegen werden Parameter von der Funktion R von Nicht-Natural-Programmen an das Formatierungssystem übergeben.	
	Bytes 1 - 2	Geben die Länge *n* dieses Parameters an.
	Bytes 3 - 4	Leer.
	Bytes 5 - *n*	Funktion O: Name des Dokuments.

		Funktion `R`: Vom Nicht-Natural-Programm gelesene Zeile.
		Funktion `W`: Ausgabezeile vom Formatierer.
	Ist ein Seitenvorschub erforderlich, steht vor der Ausgabe ein `N`, andernfalls eine `1`. Angaben zur Hervorhebung von Text (z.B. Fettdruck oder Kursivschrift) werden ignoriert, wenn die Ausgabe an ein Nicht-Natural-Programm übergeben wird.	

Beispiele COMPOSE

- Beispiel 1
- Beispiel 2
- Beispiel 3
- Beispiel 4
- Beispiel 5
- Beispiel 6
- Beispiel 7

Beispiel 1

Das folgende `COMPOSE`-Statement bewirkt, dass der formatierte Text des im Con-nect-Büro `TLIB` gespeicherten Textblocks `TEXT` auf Report 1 ausgegeben wird. Fehler und statistische Angaben werden auf Report 0 (dem Standarddrucker) ausgegeben.

```
COMPOSE RESETTING ALL
        FORMATTING INPUT 'TEXT' FROM CABINET 'TLIB'
        OUTPUT (1)
        MESSAGES LISTED ON (0)
```

Beispiel 2

Die folgenden `COMPOSE`-Statements bewirken eine formatierte Textausgabe an Report 0 (Standarddrucker).

```
COMPOSE RESETTING ALL
COMPOSE MOVING '.FI ON' 'This is an example'
COMPOSE MOVING 'for use of Con-form from'
               'within Natural applications' LAST
COMPOSE FORMATTING
```

Beispiel 3

Das folgende COMPOSE-Statement bewirkt die Zuordnung von Werten zu den Con-form-Textvariablen &VAR1 und &VAR2 in einer Con-nect-Prozedur.

```
COMPOSE ASSIGNING 'VAR1' = 'Text1', 'VAR2' = 540  ↩
```

Beispiel 4

Textblock XYZ in Büro XYLIB:

```
.FI ON
Dear Mr &name.,
.IL
I am pleased to invite you to a presentation of our new product &prod..  ↩
```

Natural-Programm:

```
...
INPUT #NAME (A32) #PROD (A32)
COMPOSE ASSIGNING 'NAME' = #NAME, 'PROD' = #PROD
        FORMATTING INPUT 'XYZ' FROM CABINET 'XYLIB'
        OUTPUT (1) MESSAGES SUPPRESSED
...
```

Vom Programm erzeugte Eingabe-Map:

```
  #NAME Davenport
  #PROD NaturalONE
```

Erzeugte Ausgabe:

```
  Dear Mr Davenport,

  I am pleased to invite you to a presentation of our new product NaturalONE.
```

Beispiel 5

Dies ist ein Beispiel für die Formatierung im Dialog-Modus mit kombinierter Eingabe-/Ausgabe-Verarbeitung. Das Beispielprogramm initiiert Con-forms zeilenorientierten Formatiermodus, übergibt einige Kommandos/Variablen an Con-form und führt eine Subroutine aus, die Statusinformationen und die formatierten Ausgabezeilen auf dem Schirm anzeigt.

```
DEFINE DATA LOCAL
01 #LINES_PER_PERFORM(P5)   /* counts repeat-loops per PERFORM CNF_OUT
01 #TRACE(A1) INIT<'N'>     /* if 'Y' displays additional trace-infos
01 #OUT_FORM(A1) INIT<'F'>  /* output format
01 #START_PAGE (P3) INIT<1> /* beginning of display
01 #CNTR (P5)               /* Loop counter
01 #STATI                   /* Status information
 02 #STATUS   (A4)          /* can be STRG TERM END or ENDX
 02 #PAGE     (B4)          /* current page number
 02 #LINE     (B4)          /* current line number on page (not .tt/.bt)
 02 #NO_LINES (B4)          /* number of lines returned
 02 REDEFINE #NO_LINES
  03 #NO_LINES_I (I4)
01 #OUTPUT(A30/4)           /* output of formatted line
01 #INDEX (P3)              /* index as pointer to output line
END-DEFINE
*
SET KEY ALL
SET CONTROL 'M9'
INPUT
   008/008 'Demonstration of formatted output to variable'(I)
   /   08X 'Enter page to start display    :' #START_PAGE(AD=MIL)
   /   08X 'Display additional trace data ?:' #TRACE(AD=MIT)
   /   08X 'Please select the output format:' #OUT_FORM(AD=MIT)
           '(F=Final without BP/US-marks'
   /   44X 'M=Final with BP/US marks "<>"'
   /   44X 'I=Intermediate)'
   /   50X 'PF3=Exit'(I)
*
IF *PF-KEY EQ 'PF3'
   SET CONTROL 'MB'
   STOP
END-IF
*
IF NOT (#OUT_FORM EQ 'F' OR EQ 'M' OR EQ 'I')
   REINPUT ' Please enter valid code!' MARK *#OUT_FORM ALARM
END-IF
*
WRITE TITLE LEFT
     'Stat * Page     * Line     * No.Lines >> Formatted Output'(I)
   / '-'(79)(I)
*
SET CONTROL 'MB'
COMPOSE RESETTING ALL /* clear all text format area of Con-form buffer
RESET #NO_LINES
*
* start line-oriented formatting-mode here
* starting from 0
DECIDE ON FIRST VALUE OF #OUT_FORM
   VALUE 'F'
      COMPOSE FORMATTING
           OUTPUT TO VARIABLES #OUTPUT (1:4)      /* to Output
```

```
           STATUS #STATUS #PAGE #LINE #NO_LINES   /* get Status
   VALUE 'M'
      COMPOSE FORMATTING
           OUTPUT TO VARIABLES CONTROL '<' '>'
                               #OUTPUT (1:4)      /* to output
           STATUS #STATUS #PAGE #LINE #NO_LINES   /* get Status
   VALUE 'I'
      COMPOSE FORMATTING
           OUTPUT TO VARIABLES CONTROL 'I' 'N'
                               #OUTPUT (1:4)      /* to output
           STATUS #STATUS #PAGE #LINE #NO_LINES   /* get Status
   NONE
      STOP
END-DECIDE
*
RESET #NO_LINES
*
* Put some commands to Con-form to see something
*
COMPOSE MOVING
     '.pl 16;.hs 2;.tt 1Formatting in Variable//;.tt 2//' /* Cmd
     OUTPUT TO VARIABLES #OUTPUT (1:4)           /* to Output
     STATUS #STATUS #PAGE #LINE #NO_LINES        /* get Status
*
COMPOSE MOVING
     '.fs 1;.bt Page End #//;.fi on;.tb *=15' /* Commands
     OUTPUT TO VARIABLES #OUTPUT (1:4)           /* to Output
     STATUS #STATUS #PAGE #LINE #NO_LINES        /* get Status
*
*
* loop 40-times, send commands to con-form and display output
*
COMPOSE ASSIGNING 'Value' = '1-20' /* Assign value to variable &Value
*
FOR #CNTR 1 40                       /* Loop some time
   IF #STATUS NE 'TERM' /* no wait-for-input => error!!!!
      IF #STATUS EQ 'STRG'
         IGNORE
      ELSE
         WRITE 'Unexpected Status-code' #STATUS(AD=OI) 'found!'
             / 'Execution has stopped....'
         STOP
      END-IF
   END-IF
*
   IF #CNTR EQ 21
      COMPOSE ASSIGNING 'Value' = '21-40' /* Assign a variable-value
   END-IF
   COMPOSE ASSIGNING 'CNTR' = #CNTR /* Again assignment
   COMPOSE MOVING
        '.BP;&Value *Pass &CNTR;.BR'      /* Commands
        OUTPUT TO VARIABLES #OUTPUT (1:4)     /* to output
```

```
        STATUS #STATUS #PAGE #LINE #NO_LINES  /* get status
   PERFORM CNF-OUT                            /* show result
END-FOR
COMPOSE MOVING
     LAST                                  /* End of processing
     OUTPUT TO VARIABLES #OUTPUT (1:4)     /* to output
     STATUS #STATUS #PAGE #LINE #NO_LINES  /* get status
*
IF #TRACE EQ 'Y'
   WRITE 'End of processing...'(I)
END-IF
*
* Subroutines
*
PERFORM CNF-OUT
*
* Subroutine to display any waiting output from Con-form
*
DEFINE SUBROUTINE CNF-OUT
   RESET #LINES_PER_PERFORM
   REPEAT UNTIL #STATUS EQ 'TERM' /* TERM = input waiting
      PERFORM BREAK               /* do some break processing
      AT BREAK OF #PAGE
         IF #PAGE GT #START_PAGE
            WRITE '-'(79)(I)
         END-IF
         IF #TRACE EQ 'Y'
            WRITE 'End of this page...'(I)
         END-IF
         NEWPAGE
      END-BREAK
      IF #PAGE GE #START_PAGE     /* show line of output
         IF #NO_LINES_I GT 0
            FOR #INDEX 1 #NO_LINES_I
               ADD 1 TO #LINES_PER_PERFORM /* count loops
               WRITE NOTIT NOHDR #STATUS '*' #PAGE '*' #LINE
                                         '*' #NO_LINES
                                         '>>' #OUTPUT (#INDEX)
            END-FOR
         END-IF
      END-IF
      IF #STATUS NE 'STRG'         /* if no wait on output
         ESCAPE BOTTOM
      END-IF
      RESET #NO_LINES
      COMPOSE MOVING
           OUTPUT TO VARIABLES #OUTPUT (1:4)    /* get output
           STATUS #STATUS #PAGE #LINE #NO_LINES /* Status
   END-REPEAT
*
   IF #TRACE EQ 'Y'
    WRITE 'Count of lines per PERFORM was'(I) #LINES_PER_PERFORM(AD=OI)
```

```
   END-IF
*
 END-SUBROUTINE
 SET CONTROL 'MB'
END
```

Beispiel 6

Textblock `'ABC'` in Büro `'ZLIB'`:

```
.op das=6
.CV c=(&A+&B+&D)*&A/12345678901234567.89
```

Natural-Programm:

```
DEFINE DATA LOCAL
 . . .
01 NA       (P14.1) INIT <-12345678.1>
01 NB       (N15.1) INIT <1234567890.1>
01 ND       (P15.1) INIT <1122334455.1>
01 NC       (N03.6)
01 NCOMP    (N03.6)
. . .
END-DEFINE
COMPOSE RESETTING ALL
COMPOSE ASSIGNING 'A'=NA,'B'=NB,'D'=ND
COMPOSE FORMATTING INPUT 'ABC' FROM CABINET 'ZLIB'
COMPOSE EXTRACTING NC='C'
COMPUTE ROUNDED NCOMP=(NA+NB+ND) * NA /12345678901234567.89
WRITE (0) 'CONFORM C =' NC / 'NATURAL C =' NCOMP
END
```

Erzeugte Ausgabe:

```
CONFORM C =   -2.344557
NATURAL C =   -2.344557
```

Beispiel 7

Dieses Beispiel erläutert die Benutzung der Formatieranweisung `.TE ON/OFF` im Dialogmodus für Eingabe. Ein Natural-Programm ruft das Con-form-Dokument `LETTER`, das diese Anweisung enthält, aus dem Büro `XYLIB`.

Natural-Programm:



Anmerkung: Dieses etwas vereinfachte Natural-Programm dient lediglich zur Demonstration; z.B. erfolgt keine Überprüfung der Pflichtfelder `SALUTATION`, `LASTNAME`, `STREET`, `CITY`.

```
DEFINE DATA LOCAL
01 #ENTER (A15) INIT <'Special offer:'>
01 SLINE  (A15) INIT <'.SL 1'>
01 #TEXT   (A60/1:4)
01 SALUTATION (A30)
01 LASTNAME   (A30)
01 STREET     (A30)
01 CITY       (A30)
01 #STATUS (A4)
01 #PAGE   (B4)
01 #LINE   (B4)
01 #NUMBER (B4)
END-DEFINE
COMPOSE RESETTING ALL
INPUT 25X 'Advertising letter'
        /  '-' (75)
        /  'Salutation: ' SALUTATION (AD='_')
        /  'Lastname  : ' LASTNAME (AD='_')
        /  'Street    : ' STREET (AD='_')
        /  'City      : ' CITY (AD='_')
        /  '-' (75)
       //  '-' #ENTER (AD=OI)
        /  '-' #TEXT(1) (AD='_')
        /  '-' #TEXT(2) (AD='_')
        /  '-' #TEXT(3) (AD='_')
        /  '-' #TEXT(4) (AD='_')
COMPOSE ASSIGNING   'SALUT' = SALUTATION,
                    'NAME'  = LASTNAME,
                    'STREET'= STREET,
                    'TOWN'  = CITY
COMPOSE FORMATTING INPUT 'LETTER' FROM CABINET 'XYLIB   '
DECIDE FOR FIRST CONDITION
  WHEN #TEXT(4) NE ' '
       COMPOSE MOVING SLINE #TEXT(1) #TEXT(2) #TEXT(3) #TEXT(4)
               '.TE OFF' STATUS #STATUS #PAGE  #LINE  #NUMBER
  WHEN #TEXT(3) NE ' '
       COMPOSE MOVING SLINE #TEXT(1) #TEXT(2) #TEXT(3)
               '.TE OFF' STATUS #STATUS #PAGE  #LINE  #NUMBER
  WHEN #TEXT(2) NE ' '
       COMPOSE MOVING SLINE #TEXT(1) #TEXT(2)
               '.TE OFF' STATUS #STATUS #PAGE  #LINE  #NUMBER
  WHEN #TEXT(1) NE ' '
       COMPOSE MOVING SLINE #TEXT(1)
               '.TE OFF' STATUS #STATUS #PAGE  #LINE  #NUMBER
  WHEN NONE
       COMPOSE MOVING
               '.TE OFF' STATUS #STATUS #PAGE  #LINE  #NUMBER
END-DECIDE
END
```

Con-form-Dokument `LETTER`:

```
.PL 22;.LM 0;.RM 60;.HS 0;.HM 0;.FM 0;.FS 0
&salut &name
&street
&town
.SL 2
Dear &salut &name.,
.SL
.LM 0;.JU OFF;.FI ON
Your subscription with MAGNIFICENT WILDLIFE magazine will soon expire.
If you act now and renew your subscription for one full year, you will
receive a 40% discount - a savings of $25.00 off the newsstand price!$
.TE ON
.SL 2
Sincerely,$
J. Baker$
Vice President of Sales
```

Bildschirm vor Eingabe:

```
                          Advertising letter
------------------------------------------------------------------------------
Salutation:  _____________________________
Lastname  :  _____________________________
Street    :  _____________________________
City      :  _____________________________
------------------------------------------------------------------------------

Special offer:
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
```

Bildschirm nach Eingabe:

```
                          Advertising letter
------------------------------------------------------------------------------
Salutation:  Mister_______________________
Lastname  :  Poe__________________________
Street    :  203 North Amity Street_______
City      :  Baltimore, Maryland__________
------------------------------------------------------------------------------

Special offer:
Take $500 off a trip to one of the world's premier wildlife-
viewing destinations through our travel partner._____________
_____________________________________________________________
_____________________________________________________________
```

Resultierender Brief nach erfolgter Formatierung:

```
MISTER POE
203 NORTH AMITY STREET
BALTIMORE, MARYLAND


Dear MISTER POE,

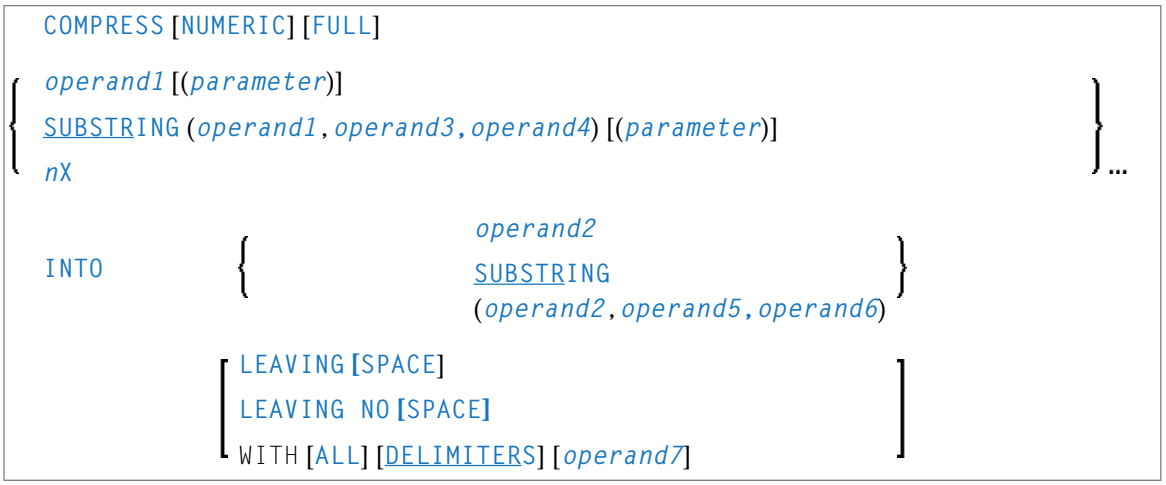
Your subscription with MAGNIFICENT WILDLIFE magazine will
soon expire. If you act now and renew your subscription for
one full year, you will receive a 40% discount - a savings
of $25.00 off the newsstand price!

TAKE $500 OFF A TRIP TO ONE OF THE WORLD'S PREMIER WILDLIFE-
VIEWING DESTINATIONS THROUGH OUR TRAVEL PARTNER.


Sincerely,
J. Baker
Vice President of Sales
```

25 COMPRESS

- Funktion COMPRESS 190
- Syntax-Beschreibung COMPRESS 190
- Verarbeitung COMPRESS 196
- Beispiele COMPRESS 196



Dieser Abschnitt 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 COMPRESS

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

Syntax-Beschreibung COMPRESS

Operanden-Definitionstabelle:

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

Syntax-Element	Beschreibung

NUMERIC	**Behandlung von Vor- und Dezimalzeichen:**
	
	Diese Option bestimmt, wie Vorzeichen und Dezimalzeichen behandelt werden:
	
	Ohne NUMERIC werden Dezimalkommas und Vorzeichen bei numerischen Ursprungswerten unterdrückt, bevor die Werte in das Zielfeld übertragen werden. Zum Beispiel:
	
	`COMPRESS -123  1.23 INTO #TARGET WITH DELIMITER '*'`
	`Content of #TARGET is: 123*123`
	
	Mit NUMERIC werden Dezimalkommas und Vorzeichen aus numerischen Ursprungswerten ebenfalls mit in das Zielfeld übertragen.
	
	Für Gleitkomma-Ursprungswerte werden Dezimalzeichen und Vorzeichen übertragen, ungeachtet der Tatsache, ob NUMERIC angegeben wurde oder nicht.
	
	Beispiel 1:
	
	`COMPRESS NUMERIC -123  1.23 INTO #TARGET WITH DELIMITER '*'`
	`Content of #TARGET is: -123*1.23`
	
	Beispiel 2:
	
	`COMPRESS NUMERIC 'ABC' -0056.00 -0056.10 -0056.01 INTO #TARGET WITH ↩`
	`DELIMITER '*'`
	`Content of #TARGET is: ABC*-56*-56.1*-56.01`
	
	Beispiel 3:
	
	`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`

FULL	**Behandlung von Ursprungsfeldwerten:** Ohne FULL werden folgende Zeichen aus den Ursprungsfeldern entfernt, bevor die Werte übertragen werden: ■ vorangestellte Nullen vor dem Komma oder Dezimalpunkt für Felder vom Format N, P oder I,

Syntax-Element	Beschreibung

	■ 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
	
	Enthält ein numerisches Ursprungsfeld lauter Nullen, wird eine Null (0) übertragen. Zum Beispiel:
	
	`COMPRESS 'ABC '  001 INTO #TARGET WITH DELIMITER '*'`
	`Content of #TARGET is: ABC*1`
	
	Mit `FULL` werden die Werte der Ursprungsfelder 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:
	
	■ 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
	
	Beispiel:
	
	`COMPRESS FULL 'ABC '  001 INTO #TARGET WITH DELIMITER '*'`
	`Content of #TARGET is: ABC *001`

operand1	**Ursprungsfelder:** Als *operand1* geben Sie die Felder an, deren Inhalt übertragen werden soll. **Anmerkung:** Wenn *operand1* nicht Format A oder B hat, wird sein Inhalt in alphanumerische Darstellung konvertiert, bevor er übertragen wird. Wenn erforderlich, wird die alphanumerische Darstellung abgeschnitten. Bei Benutzung von *operand1* ohne explizite Editiermaske: ■ wird eine Zeitvariable (Format T) nur mit der Zeitkomponente des Variableninhalts, aber ohne die Datumskomponente übertragen, ■ wird eine logische Variable (Format L) mit dem Wert `<false>` durch ein Leerzeichen und der Wert `<true>` durch das Zeichen `X` dargestellt.
n`X`	**Einfügen eines Abstandelements:**

Syntax-Element	Beschreibung

	Mit der Notation *nX* können Sie festlegen, dass an dieser Stelle *n* Leerzeichen eingefügt werden. Das Abstandselement ist eine Leerzeichenkonstante, die immer in das Zielfeld verschoben wird, wobei jedoch keine Leerzeichenverdichtung erfolgt.
	
	Bei Verwendung von `LEAVING NO SPACE` wird nur die angegebene Anzahl der mit *nX* definierten Leerzeichen an dieser Stelle eingefügt.
	
	Bei Verwendung von `LEAVING SPACE` oder `WITH DELIMITER` wird ein zusätzliches Begrenzungszeichen vor und nach dem diesem Abstandselement eingefügt.
	
	Die maximal zulässige Länge beträgt `40X`.
	
	Bei Verwendung von `LEAVING NO SPACE`, aber nicht `FULL`, werden die Daten aller Verdichtungselemente lückenlos ohne Trennzeichen aneinandergehängt. Sollte dennoch eine Lücke zwischen zwei Feldern erforderlich sein, kann ein Abstandselement zwischen ihnen eingefügt werden.
	
	Beispiele:
	
	`COMPRESS 'A' 2X 'B' INTO #T WITH DELIMITER ','`
	`-> #T = 'A,  ,B'`
	`COMPRESS 'A' 2X 'B' INTO #T LEAVING NO SPACE`
	`-> #T = 'A  B'`
	`COMPRESS 'Error' 1X #ERR 1X 'came from DB file (' #DBID '/' #FNR ')' ↩`
	`INTO #T LEAVING NO SPACE`
	`-> #T = 'Error 1234 came from DB file (10/32)'`
	
	Siehe auch **Beispiel 5 - COMPRESS-Statement mit Abstandselement**.

operand2	**Zielfeld** Als *operand2* geben Sie das Feld an, das die Werte aus den Ursprungsfeldern aufnehmen soll. Wenn das Zielfeld das Format U (Unicode) hat, und wenn es sich um ein Ursprungsfeld mit Format B handelt, muss die Länge des sendenden Binärfeldes gleich sein.
`LEAVING SPACE`	**Trennung der Werte im Zielfeld durch Leerzeichen:** Wenn Sie das `COMPRESS`-Statement ohne weitere Optionen verwenden oder `LEAVING SPACE` (gilt auch standardmäßig) angeben, werden die Werte im Zielfeld jeweils durch ein Leerzeichen voneinander getrennt.
`LEAVING NO SPACE`	**Keine Trennung der Werte im Zielfeld durch Leerzeichen:** Wenn Sie `LEAVING NO SPACE` angeben, werden die Werte im Zielfeld weder durch ein Leerzeichen noch durch ein anderes Zeichen voneinander getrennt.
parameter	**Parameter für Druckmodus, Datumsformat, Editiermaske:** Als *parameter* können Sie die Session-Parameter `PM`, `DF`, `EM` oder `EMU` angeben:
	PM=I — Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft,

Syntax-Element	Beschreibung	

		können Sie die Option PM=I angeben, um den Wert von *operand1* invers (d.h. von rechts nach links) in *operand2* zu übertragen.
		
		Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld #B den Inhalt ZYXABC:
		
		`MOVE 'XYZ' TO #A`
		`COMPRESS #A (PM=I) 'ABC'`
		`INTO #B LEAVING NO SPACE`
		
		Nachfolgende Leerzeichen in *operand1* werden entfernt, außer wenn FULL angegeben ist, dann wird der Wert Zeichen für Zeichen umgedreht und anschließend in *operand2* übertragen.

	DF	Wenn *operand1* eine Datumsvariable ist, können Sie den Session-Parameter DF als *parameter* für diese Variable angeben.
	EM	**Editiermaske:** Einzelheiten zu Editiermasken siehe Session-Parameter EM in der *Parameter-Referenz*-Dokumentation. Der Session-Parameter EM kann nicht angewendet werden bei Gruppenoperanden oder wenn die SUBSTRING-Option benutzt wird.
	EMU	**Unicode-Editiermaske:** Einzelheiten zu Editiermasken siehe Session-Parameter EMU in der *Parameter-Referenz*-Dokumentation. Der Session-Parameter EMU kann nicht angewendet werden bei Gruppenoperanden oder wenn die SUBSTRING-Option benutzt wird.
SUBSTRING (*operand1*, *operand3*, *operand4*)	**SUBSTRING-Option:** Wenn *operand1* alphanumerisches (A), Unicode (U) oder binäres Format (B) hat, können Sie die SUBSTRING-Option verwenden, um nur einen Teil des Urspungsfeldes zu übertragen. Hinter dem Feldnamen (*operand1*) geben Sie zuerst die Startposition (*operand3*) und dann die Länge (*operand4*) des zu übertragenden Feldabschnitts ein.	
INTO SUBSTRING (*operand2*, *operand5*, *operand6*)	**INTO-Klausel:** Sie können die SUBSTRING-Option auch in der INTO-Klausel verwenden, um die Urspungswerte in einen bestimmten Teil des Zielfeldes zu übertragen.	

Syntax-Element	Beschreibung
	Die Verwendung der SUBSTRING-Option in einem COMPRESS-Statement entspricht in beiden Fällen der in einem MOVE-Statement. Einzelheiten zur SUBSTRING-Option finden Sie beim MOVE-Statement.
WITH DELIMITERS	**Trennzeichen bei Werten im Zielfeld:** Möchten Sie, dass die Werte im Zielfeld jeweils durch ein Trennzeichen voneinander getrennt werden, dann verwenden Sie die DELIMITERS-Option. Wenn Sie WITH DELIMITERS ohne *operand7* angeben, werden die Werte durch das (mit der Session-Parameter ID definierte) Input-Delimiter-Zeichen voneinander getrennt.
WITH DELIMITERS *operand7*	**Spezielles Trennzeichen:** Wenn Sie WITH DELIMITERS *operand7* angeben, werden die Werte durch das mit *operand7* angegebene Zeichen voneinander getrennt. *operand7* muss ein einzelnes Zeichen sein. Wenn *operand7* eine Variable ist, muss sie Format/Länge (A1) oder (B1) haben. Wenn das Zielfeld das Format A oder B hat, muss das Format bzw. die Länge des Trennzeichens (A1), (B1) oder (U1) sein. Wenn das Zielfeld das Format U (Unicode) hat, muss das Format bzw. die Länge des Trennzeichens (A1), (B2) oder (U1) sein.

WITH ALL	**Anwendung der Trennzeichen:**
	
	Ohne ALL werden im Zielfeld Delimiter-Zeichen nur zwischen tatsächlich übertragenen Werten gesetzt. Zum Beispiel:
	
	`COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH DELIMITERS '*'`
	`Content of #TARGET is: A*C`
	
	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 Urspungsfelder minus 1. Dies kann z.B. sinnvoll sein, wenn der Inhalt des Zielfeldes mit einem SEPARATE-Statement anschließend wieder aufgeteilt werden soll. Zum Beispiel:
	
	`COMPRESS 'A' ' ' 'C' ' ' INTO #TARGET WITH ALL DELIMITERS '*'`
	`Content of #TARGET is: A**C*`

Verarbeitung COMPRESS

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 Quelloperanden verarbeitet worden sind. Es werden keine Zeichen abgeschnitten. Die Länge von *operand2* nach der COMPRESS-Operation entspricht dann der gemeinsamen Länge der Quelloperanden. Die aktuelle Länge einer dynamischen Variable kann durch die Systemvariable *LENGTH bestimmt werden.

Beispiele COMPRESS

- Beispiel 1 - COMPRESS-Statement
- Beispiel 2 - COMPRESS-Statement mit LEAVING NO SPACE
- Beispiel 3 - COMPRESS-Statement mit WITH DELIMITER
- Beispiel 4 - COMPRESS-Statement mit Editiermasken EM
- Beispiel 5 - COMPRESS-Statement mit Abstandselement

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   ANNUAL            #CCSALARY
                       CODE     SALARY
-------------------- -------- ----------     --------------------

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   ANNUAL            #CCSALARY
                        CODE     SALARY
-------------------- -------- ----------     --------------------

ABELLAN              PTA         1450000     PTA*1450000
ACHIESON             UKL           11300     UKL*11300
ADAM                 FRA          159980     FRA*159980
ADKINSON             USD           34500     USD*34500
```

Beispiel 4 - COMPRESS-Statement mit Editiermasken EM

```
** Example 'CMPEX4': COMPRESS (with edit mask EM)
************************************************************************
DEFINE DATA LOCAL
1 #A10     (A10)   INIT <'ABCDEF'>
1 #I4      (I4)    INIT <-123>
1 #T       (T)     INIT <E'2021-11-22 10:24:36'>
1 #L       (L)     INIT <TRUE>
1 #RESULT  (A70)
END-DEFINE
*
COMPRESS '#A:'  #A10  (EM=X_X_X)
         '#I4:' #I4   (EM=-9999)
         '#T:'  #T    (EM=YYYY-MM-DD_HH:II)
```

```
         '#L:'  #L    (EM=FALSE/TRUE)       INTO #RESULT
PRINT #RESULT
END
```

Ausgabe des Programms `CMPEX4`:

```
#A: A_B_C #I4: -0123 #T: 2021-11-22_10:24 #L: TRUE
```

Beispiel 5 - COMPRESS-Statement mit Abstandselement

```
** Example 'CMPEX5': COMPRESS (with spacer element)
************************************************************************
DEFINE DATA LOCAL
1 #A10     (A10)   INIT <'ABCDEF'>
1 #I4      (I4)    INIT <-123>
1 #T       (T)     INIT <E'2001-11-22 10:24:36'>
1 #L       (L)     INIT <TRUE>
*
1 #FIELD1  (A16)   INIT <'First name'>
1 #FIELD2  (A16)   INIT <'Name'>
1 #FIELD3  (A16)   INIT <'Birth'>
1 #VALUE1  (A32)   INIT <'John'>
1 #VALUE2  (A32)   INIT <'Smith'>
1 #VALUE3  (D)     INIT <D'2005-05-01'>
*
1 #ERROR   (I2)    INIT <76>
1 #TEXT    (A40)   INIT <'Error occurs in library'>
1 #LIBRARY (A8)    INIT <'XYZ'>
1 #DBID    (I4)    INIT <88>
1 #FNR     (I4)    INIT <317>
*
1 #RESULT  (A70)
END-DEFINE
*
* Use spacer elements and values directly followed by comma or period
*
COMPRESS 'Values found:' 1X #A10 (EM=X_X_X) ','
         1X
         #I4 (EM=-9999) ','
         1X
         #T  (EM=YYYY-MM-DD' 'HH:II) ','
         1X
         #L  (EM=FALSE/TRUE) '.'
         INTO #RESULT  LEAVING NO SPACE
PRINT #RESULT //
*
* Use spacer elements and delimiter asterisk
*
COMPRESS '#A10:' #A10 (EM=X_X_X)
         3X
         '#I4:'  #I4  (EM=-9999)
```

```
         3X
         '#T:'   #T   (EM=YYYY-MM-DD' 'HH:II)
         3X
         '#L:'   #L   (EM=FALSE/TRUE)
         INTO #RESULT  WITH DELIMITER '*'
PRINT #RESULT //
*
* Use spacer elements and asterisks for extra styling
*
COMPRESS #FIELD1 ':' 1X #VALUE1 3X '***' 3X
         #FIELD2 ':' 1X #VALUE2 3X '***' 3X
         #FIELD3 ':' 1X #VALUE3 (EM=DD.L(10)' 'YYYY)
         INTO #RESULT  LEAVING NO SPACE
PRINT #RESULT //
*
* Use spacer elements in complex text string
*
COMPRESS 'ERR' #ERROR (EM=9999) 2X
         #TEXT 1X '<' #LIBRARY '>' 1X
         'with DBID=' #DBID ',' 1X 'FNR=' #FNR '.'
         INTO #RESULT  LEAVING NO SPACE
PRINT #RESULT
*
END
```

Ausgabe des Programms CMPEX5:

```
Values found: A_B_C, -0123, 2001-11-22 10:24, TRUE.

#A10:*A_B_C*   *#I4:*-0123*   *#T:*2001-11-22 10:24*   *#L:*TRUE

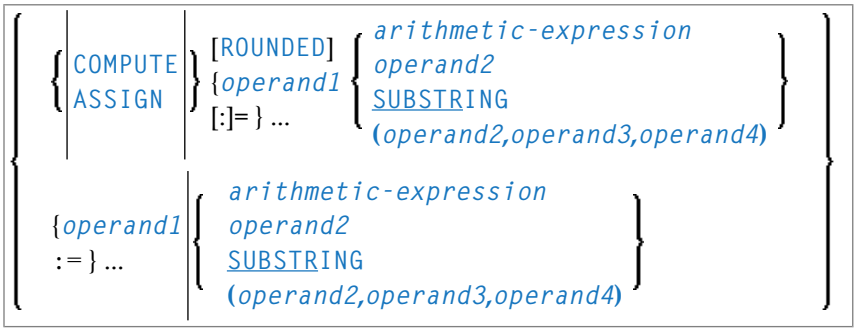
First name: John   ***   Name: Smith   ***   Birth: 01.May 2005

ERR0076  Error occurs in library <XYZ> with DBID=88, FNR=317.
```

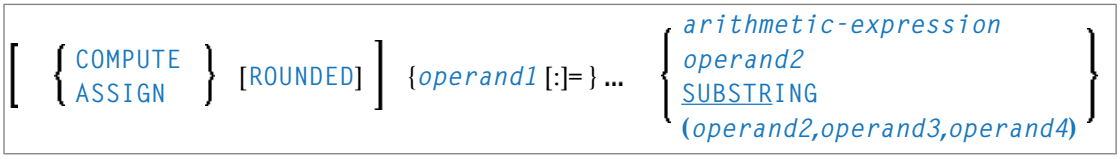
26 COMPUTE

- Funktion COMPUTE 202
- Syntax-Beschreibung COMPUTE 204
- Ergebnisgenauigkeit einer Division 207
- Beispiele COMPUTE 207

Structured Mode Syntax



Reporting Mode Syntax



Dieser Abschnitt 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 COMPUTE

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 Quelloperand (*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 Quelloperand 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 Progammierung* und dort insbesondere die folgenden Abschnitte:

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

Syntax-Beschreibung COMPUTE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur						Mögliche Formate													Referenzierung erlaubt	Dynam. Definition
operand1		S	A		M		A	U	N	P	I	F	B	D	T	L	C	G	O	ja	ja
operand2	C	S	A		N	E	A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand3	C	S							N	P	I		B*							ja	nein
operand4	C	S							N	P	I		B*							ja	nein

* Wenn *operand3* oder *operand4* eine binäre Variable ist, kann er nur mit einer Länge kleiner als/gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`COMPUTE | ASSIGN [:]=`	**Verwendung der Schlüsselwörter:** Sie können das Statement in Kurzform angeben und das Statement-Schlüsselwort `COMPUTE` (bzw. `ASSIGN`) weglassen. Wenn Sie im Structured Mode das Statement-Schlüsselwort weglassen, müssen Sie vor das Gleichheitszeichen (=) einen Doppelpunkt (:) schreiben. Verwenden Sie die `ROUNDED`-Option, müssen Sie den Statement-Namen angeben.
`ROUNDED`	**ROUNDED-Option:** Wenn Sie das Schlüsselwort `ROUNDED` angeben, wird der Wert auf- bzw. abgerundet, bevor er *`operand1`* zugewiesen wird. Die für das Runden gültigen Regeln finden Sie im Abschnitt *Regeln für arithmetische Operationen, Abschneiden und Runden von Feldwerten* im *Leitfaden zur Programmierung*.
`operand1`	**Ergebnisfeld:** *`operand1`* nimmt das Ergebnis der arithmetischen Operation bzw. Zuweisung auf. Zur Genauigkeit des Ergebnisses siehe Abschnitt *Genauigkeit von Ergebnissen bei arithmetischen Operationen* im *Leitfaden zur Programmierung*. Wenn *`operand1`* ein Datenbankfeld ist, ändert sich der Wert des Feldes auf der Datenbank dadurch nicht. Falls *`operand1`* eine dynamische Variable ist, wird er bis zur Länge von *`operand2`* oder bis zur Länge des Ergebnisses der arithmetische Operation einschließlich der nachfolgenden Leerzeichen aufgefüllt, und die Länge von *`operand1`* wird dann entsprechend angepasst. Die aktuelle Länge einer dynamischen Variablen kann durch die Systemvariable `*LENGTH` bestimmt werden. Allgemeine Informationen zu dynamischen Variablen entnehmen Sie dem Abschnitt *Dynamische und große Variablen benutzen*.
`arithmetic-expression`	**Arithmetischer Ausdruck:** Ein arithmetischer Ausdruck (*`arithmetic-expression`*) besteht aus einer oder mehreren Konstanten, Datenbankfeldern bzw. Benutzervariablen.

<table>
<tr><th>Syntax-Element</th><th>Beschreibung</th></tr>
<tr><td></td><td>In Natural verfügbare Mathematische Systemfunktionen (siehe Systemfunktionen-Dokumentation) können ebenfalls als arithmetische Operanden verwendet werden.

Die in einem arithmetischen Ausdruck verwendeten Operanden müssen eines der folgenden Formate haben: N, P, I, F, D oder T.

Zum Format der Operanden siehe auch Formatwahl im Hinblick auf die Verarbeitungszeit im Leitfaden zur Programmierung.

Die folgenden verbindenden Operatoren können verwendet werden:
<table>
<tr><th>Operator</th><th>Symbol</th></tr>
<tr><td>Klammern</td><td>( )</td></tr>
<tr><td>Potenzierung</td><td>**</td></tr>
<tr><td>Multiplikation</td><td>*</td></tr>
<tr><td>Division</td><td>/</td></tr>
<tr><td>Addition</td><td>+</td></tr>
<tr><td>Subtraktion</td><td>-</td></tr>
</table>
Jedem Operatorzeichen sollte jeweils mindestens ein Leerzeichen voran- und nachgestellt werden, damit es nicht zu Konflikten mit Variablennamen, die eines dieser Zeichen enthalten, kommen kann.

Bei der Verarbeitung arithmetischer Operationen gilt folgende Reihenfolge:

1. Klammerrechnung
2. Potenzrechnung
3. Multiplikation/Division (von links nach rechts, wie erkannt)
4. Addition/Subtraktion (von links nach rechts, wie erkannt)</td></tr>
<tr><td>operand2</td><td>Quellfeld:

operand2 ist das Quellfeld.

Wenn operand1 das Format C hat, kann operand2 auch als eine Attribut-Konstante angegeben werden (siehe Benutzerkonstanten im Leitfaden zur Programmierung).</td></tr>
<tr><td>SUBSTRING
(operand2,operand3,operand4)</td><td>SUBSTRING-Option:

Ohne SUBSTRING-Option wird der ganze Inhalt des operand2 übertragen.

Wenn das Format von operand1 und operand2 alphanumerisch, Unicode oder binär ist, ermöglicht es Ihnen die SUBSTRING-Option,</td></tr>
</table>

Syntax-Element	Beschreibung
	nur einen bestimmten Teil von *operand2* nach *operand1* zu übertragen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand2*) zunächst die erste Stelle (*operand3*) und dann die Länge (*operand4*) des Feldteils an, der übertragen werden soll. Beispiel: Um die dritte bis sechste Stelle des Feldes #B dem Feld #A zu übertragen, geben Sie an: `#A := SUBSTRING(#B,3,4)` Wenn Sie *operand3* weglassen, wird ab Anfang des Feldes übertragen. Wenn Sie *operand4* weglassen, wird ab der angegebenen Stelle (*operand3*) bis zum Ende des Feldes übertragen. **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). Siehe auch MOVE SUBSTRING.

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).

Beispiele COMPUTE

- 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 (N0.3)
1 #D (N0.5)
1 #E (N1.3)
1 #F (N5)
1 #G (A25)
1 #H (A3/1:3)
END-DEFINE
*
ASSIGN #A = 5                              WRITE NOTITLE '=' #A
ASSIGN #B = 'ABC'                          WRITE '=' #B
ASSIGN #C = .45                            WRITE '=' #C
ASSIGN #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 `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 'CPTEX1': 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 `CPTEX1`:

```
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
```

27 CREATE OBJECT

- Funktion CREATE OBJECT 212
- Syntax-Beschreibung CREATE OBJECT 212

```
CREATE OBJECT operand1 OF [CLASS] operand2
[GIVING operand4]
```

Dieser Abschnitt 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 CREATE OBJECT

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

Syntax-Beschreibung CREATE OBJECT

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S															O	nein	nein
operand2	C	S				A												ja	nein
operand4		S			N				I									ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Objekt-Handle:** *operand1* muss als Objekt-Handle (HANDLE OF OBJECT) definiert sein. Die Objekt-Handle wird gefüllt, wenn das Objekt erfolgreich erstellt wurde. Wenn *operand1* nicht erfolgreich zurückgegeben wird, enthält er den Wert NULL-HANDLE.
OF CLASS *operand2*	**Klassen-Name:** *operand2* 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.

Syntax-Element	Beschreibung
	``` CREATE OBJECT #O1 OF CLASS "Employee" or CREATE OBJECT #O1 OF CLASS "653BCFE0-84DA-11D0-BEB3-10005A66D231" ```
`GIVING` *operand4*	**GIVING-Klausel:** Wenn die `GIVING`-Klausel angegeben wird, enthält *operand4* entweder die Natural-Meldungsnummer, falls ein Fehler auftritt, oder Null bei fehlerfreier Ausführung. Wenn die `GIVING`-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung ausgelöst, falls ein Fehler auftritt.

28 DECIDE FOR

- Funktion DECIDE FOR 216
- Syntax-Beschreibung DECIDE FOR 216
- Beispiele DECIDE FOR 217

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

Dieser Abschnitt 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 DECIDE FOR

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 DECIDE FOR

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`FIRST CONDITION`	**Nur die erste Bedingung verarbeiten:** Nur die erste wahre Bedingung soll verarbeitet werden. Siehe auch ***Beispiel 1***.
`EVERY CONDITION`	**Alle Bedingungen verarbeiten:** Jede wahre Bedingung soll verarbeitet werden. Siehe auch ***Beispiel 2***.

Syntax-Element	Beschreibung
`WHEN logical-condition statement`	**Zu verarbeitende logische Bedingungen:** Mit dieser Klausel geben Sie eine oder mehrere logische Bedingungen (*logical-condition*) an, die verarbeitet werden sollen. Siehe Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung*).
`WHEN ANY statement`	**WHEN ANY-Klausel:** Mit `WHEN ANY` können Sie das (die) **Statement(s)** angeben, die ausgeführt werden sollen, wenn irgendeine der angegebenen Bedingungen erfüllt ist.
`WHEN ALL statement`	**WHEN ALL-Klausel:** Mit `WHEN ALL` 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 `EVERY` eingesetzt werden.
`WHEN NONE statement`	**WHEN NONE-Klausel:** Mit `WHEN NONE` können Sie das (die) **Statement(s)** angeben, die ausgeführt werden sollen, wenn keine der angegebenen Bedingungen erfüllt ist.
`END-DECIDE`	**Ende des DECIDE FOR-Statements:** Das für Natural reservierte Wort `END-DECIDE` muss zum Beenden des `DECIDE FOR`-Statements benutzt werden.

Beispiele DECIDE FOR

- 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 #PARM     (A1)
END-DEFINE
*
INPUT #FUNCTION #PARM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARM = 'X'
```

```
    WRITE 'Funktion A with parameter X selected.'
  WHEN #FUNCTION = 'B' AND #PARM = '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 #PARM
```

Nach Eingabe von `A` und `Y` und Drücken der EINGABE-Taste:

```
#FUNCTION A #PARM Y

Please enter a valid function.
```

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
```

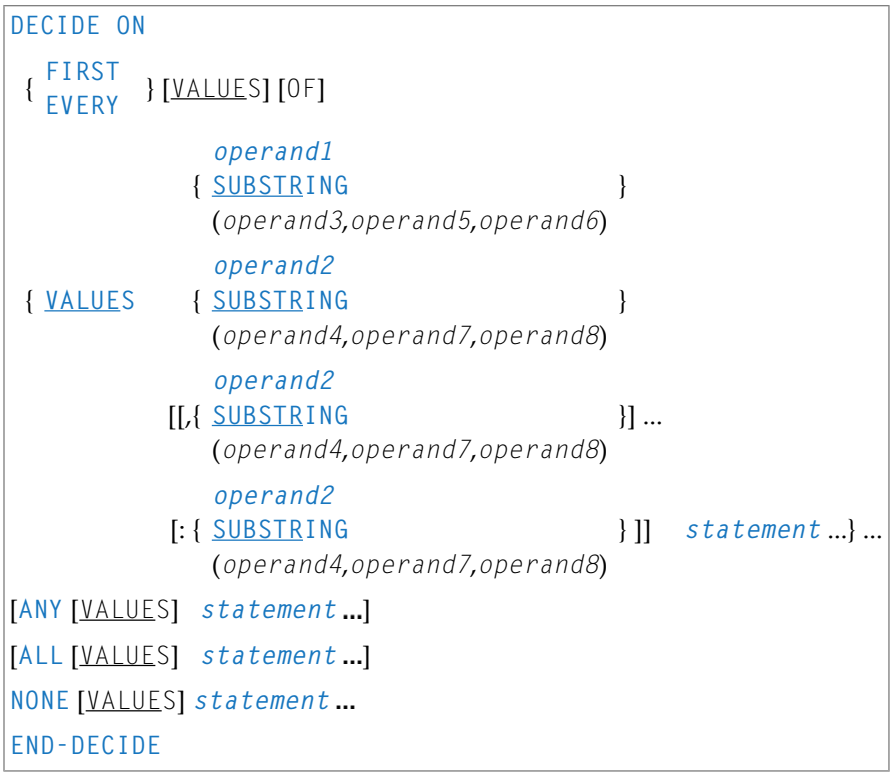
Drücken Sie dann die EINGABE-Taste:

```
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.
```

29 DECIDE ON

- Funktion DECIDE ON 222
- Syntax-Beschreibung DECIDE ON 223
- Beispiele DECIDE ON 225



Dieser Abschnitt 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 DECIDE ON

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 DECIDE ON

Operanden-Definitionstabelle:

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

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.
operand1	**Kontrollfeld:** Als *operand1* oder *operand2* geben Sie den Namen des Feldes an, dessen Werte geprüft werden sollen.
`VALUES` *operand2* `[[,`*operand2*`] ... [:`*operand2*`]`*statement* `...`	**Wert des Kontrollfeldes:** Mit dieser Klausel geben Sie den Wert (*operand2*) des Kontrollfeldes an, sowie die *statements*, 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

Syntax-Element	Beschreibung
	darf hierzu allerdings nicht verwendet werden, falls das Komma als Dezimalkomma (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 (*operand3*,*operand5*,*operand6*)	**SUBSTRING-Option:** 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. Nach dem Feldnamen (*operand3*) geben Sie zuerst die Startposition (*operand5*) und danach die Länge (*operand6*) des zu prüfenden Teils des Feldes an.
SUBSTRING (*operand4*,*operand7*,*operand8*)	Nach dem Feldnamen (*operand4*) geben Sie zuerst die Startposition (*operand7*) und danach die Länge (*operand8*) des zu prüfenden Teils des Feldes an.
ANY *statement*	**ANY-Klausel:** 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.
ALL *statement*	**ALL-Klausel:** 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. Die ALL-Klausel kann nur in Verbindung mit dem Schlüsselwort EVERY eingesetzt werden
NONE *statement*	**NONE-Klausel:** Mit NONE geben Sie das (die) **Statement(s)** an, die ausgeführt werden sollen, wenn keiner der angegebenen Werte gefunden wurde.
END-DECIDE	**Ende des DECIDE ON-Statements:** Das für Natural reservierte Wort END-DECIDE muss zum Beenden des DECIDE ON-Statements benutzt werden.

Beispiele DECIDE ON

- 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
```

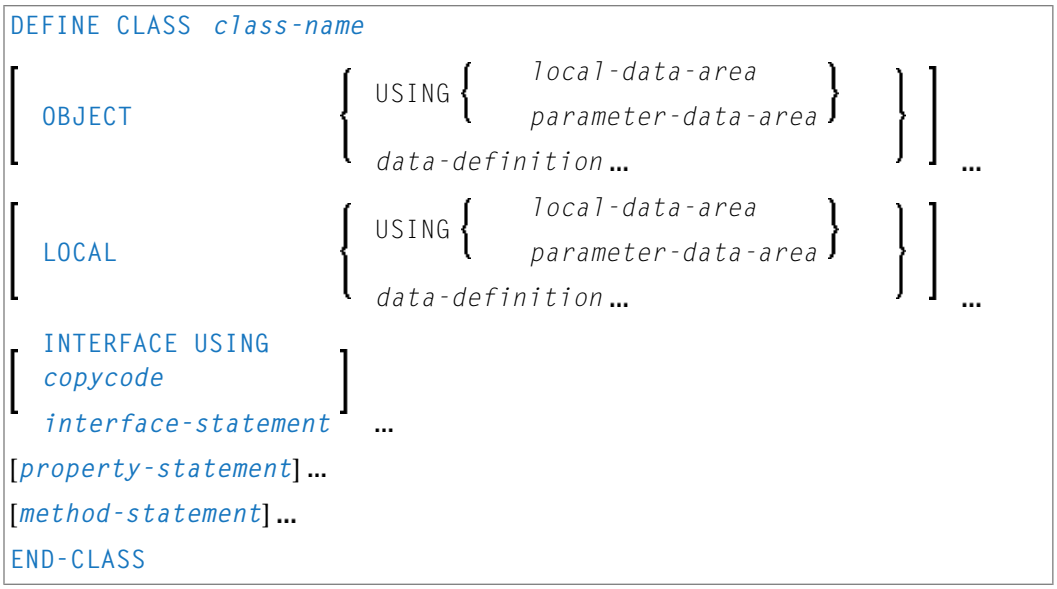
Nach Eingabe und Bestätigung des Wertes 4:

```
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
```

30 DEFINE CLASS

- Funktion DEFINE CLASS 228
- Syntax-Beschreibung DEFINE CLASS 228



Dieser Abschnitt 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 DEFINE CLASS

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 DEFINE CLASS

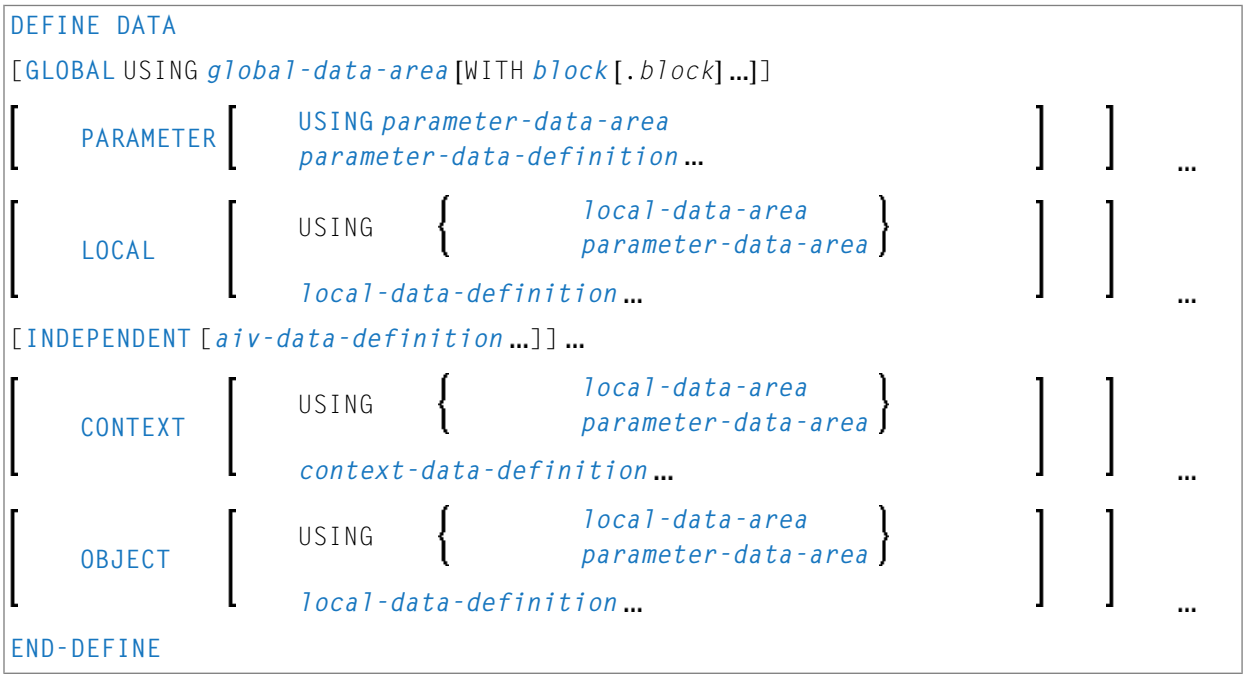
Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
class-name	**Klassen-Name:** 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: *company-name.application-name.class-name* Jeder Bestandteil zwischen den Punkten (...) muss den Natural-Namenskonventionen für Benutzervariablen entsprechen. Wenn die Klasse von Clients verwendet werden soll, die in unterschiedlichen Programmiersprachen geschriebenen sind, sollte der Klassen-Name so gewählt werden, dass er nicht gegen die in diesen Sprachen geltenden Namenskonventionen verstößt.
OBJECT	**OBJECT-Klausel:** Die OBJECT-Klausel dient dazu, die 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.
LOCAL	**LOCAL-Klausel:** 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. 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.
INTERFACE USING	**INTERFACE USING-Klausel:** Die INTERFACE USING-Klausel wird verwendet, um einen Copycode aufzunehmen, der INTERFACE-Statements enthält.
copycode	**Copycode:** Der von der INTERFACE USING-Klausel verwendete Copycode kann eines oder mehrere INTERFACE-Statements enthalten
interface-statement	**INTERFACE-Statement:** Das INTERFACE-Statement wird verwendet, um Methods und Properties für eine Klasse zu definieren.
property-statement	**PROPERTY-Statement:**

Syntax-Element	Beschreibung
	Das PROPERTY-Statement wird benutzt, um einer Property einen Objektdatenvariablen-Operanden als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.
method-statement	**METHOD-Statement:** Das METHOD-Statement wird benutzt, um einer Method ein Subprogramm als Implementierung zuzuweisen, und zwar außerhalb einer Schnittstellen-Definition.
END-CLASS	**Ende des DEFINE CLASS-Statements:** Das für Natural reservierte Wort END-CLASS muss zum Beenden des DEFINE CLASS-Statements benutzt werden.

IV DEFINE DATA

Allgemeine Syntax



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

Verwandte Themen im *Leitfaden zur Programmierung*: *Benutzung und Struktur des DEFINE DATA-Statements* | *Datenbereiche (Data Areas)* | *Speicherplatzausrichtung*

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

Allgemein:

Funktion und generelle Syntaxregeln

Datendefinitionen:

Definition von Global Data
Definition von Parameter Data
Definition von Local 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
Definition der Array-Dimension
Definition des Ausgangswertes
Ausgangswerte/Konstanten-Werte für ein Array
EM-, HD-, PD-Parameter für Feld/Variable

Beispiele:

Beispiele für die Benutzung des DEFINE DATA-Statements

31 Funktion und generelle Syntaxregeln

- Funktion DEFINE DATA 234
- Generelle Syntaxregeln DEFINE DATA 234
- Programmiermodi DEFINE DATA 234

Dieser Abschnitt behandelt folgende Themen:

Funktion DEFINE DATA

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 Angabe von Inline-Definitionen (siehe ***Lokale Daten-Definition***).

Generelle Syntaxregeln DEFINE DATA

- 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 (GLOBAL, PARAMETER, LOCAL, INDEPENDENT, CONTEXT oder OBJECT) angegeben werden.
- Sie können mehr als eine Klausel angeben. Falls jedoch die GLOBAL-Klausel und die PARAMETER-Klausel benutzt werden, muss die GLOBAL-Klausel die erste Klausel in dem Statement sein und die PARAMETER-Klausel muss auf die GLOBAL-Klausel folgen (ohne GLOBAL-Klausel kommt die PARAMETER-Klausel als erste Klausel, falls benutzt). Alle anderen Klauseln können in beliebiger Reihenfolge angegeben werden.
- Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

Programmiermodi DEFINE DATA

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

Allgemein gilt Folgendes:

- Structured Mode

- Reporting Mode

Structured Mode

Alle verwendeten Variablen (außer **anwendungsunabhängigen Variablen** = AIVs) müssen im `DEFINE DATA`-Statement definiert werden. Sie dürfen innerhalb eines Programms an keiner anderen Stelle definiert werden. AIVs dürfen nicht an anderer Stelle im Programm definiert werden, wenn ein `DEFINE DATA INDEPENDENT`-Statement benutzt wird.

Reporting Mode

Das `DEFINE DATA`-Statement ist 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.

32 Definition von Global Data

- Funktion DEFINE DATA GLOBAL 238
- Syntax-Beschreibung DEFINE DATA GLOBAL 238

Allgemeine Syntax von `DEFINE DATA GLOBAL`:

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

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

Funktion DEFINE DATA GLOBAL

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

Syntax-Beschreibung DEFINE DATA GLOBAL

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`USING` *`global-data-area`*	**GDA-Name:** Geben Sie den Namen der zu referenzierenden Global Data Area (GDA) an. Eine Global Data Area wird mit dem *Datenbereich-Editor (Data Area Editor)* erstellt. Sie enthält vordefinierte Datenelemente, die in das `DEFINE DATA GLOBAL`-Statement übernommen werden können. Im Gegensatz zur LDA können die in einer GDA definierten Datenelemente von mehreren Programmierobjekten referenziert werden. Weitere Informationen siehe Global Data Area im *Leitfaden zur Programmierung*.
`WITH` *`block`*	**Mit Datenblöcken:** Um Datenspeicherplatz einzusparen, können Sie eine Global Data Area mit Datenblöcken erstellen. Datenblöcke können sich bei der Programmausführung gegenseitig überlagern, wodurch Speicherplatz eingespart wird. Die maximale Anzahl der Blockebenen ist 8 (einschließlich des Master-Blocks). Weitere Informationen, siehe *Datenblöcke* im *Leitfaden zur Programmierung*.
`.block`	**Im Programm zu verwendende Datenblöcke:** *`.block`*-Notationen geben den Block oder die Blöcke an, der bzw. die im Programm benutzt wird bzw. werden.

Syntax-Element	Beschreibung
END-DEFINE	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

33 Definition von Parameter Data

- Funktion DEFINE DATA PARAMETER 242
- Einschränkungen DEFINE DATA PARAMETER 242
- Syntax-Beschreibung DEFINE DATA PARAMETER 242

Allgemeine Syntax von `DEFINE DATA PARAMETER`:

```
DEFINE DATA
PARAMETER [ USING parameter-data-area
          parameter-data-definition ... ]
END-DEFINE
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion DEFINE DATA PARAMETER

Das `DEFINE DATA PARAMETER`-Statement wird benutzt, um die Datenelemente zu definieren, die als Eingabeparameter in einem Natural-Subprogramm, einer externen Subroutine oder Helproutine 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 dann das Statement diese Data Area referenziert.

Einschränkungen DEFINE DATA PARAMETER

- 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.

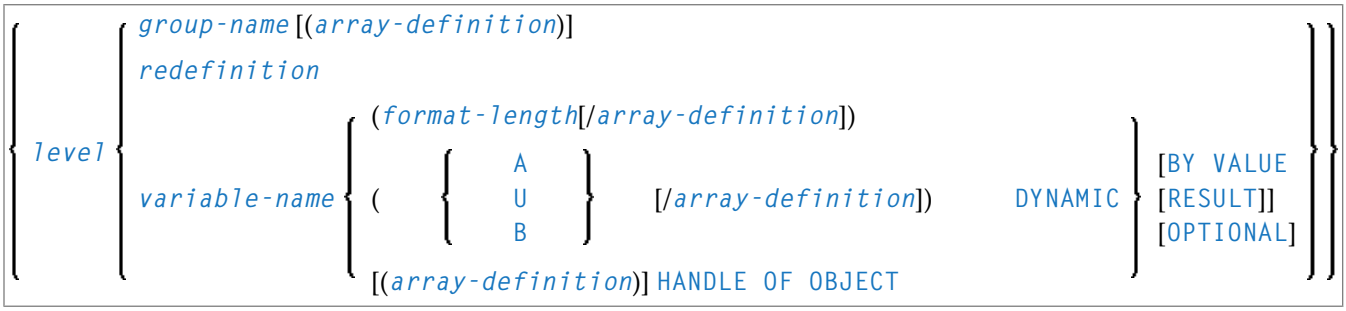
Syntax-Beschreibung DEFINE DATA PARAMETER

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`USING parameter-data-area`	**PDA-Name:** Der Name der *parameter-data-area*, die Datenelemente enthält, die als Parameter in einem Subprogramm, einer externen Subroutine oder einem Dialog benutzt werden.
parameter-data-definition	**Parameterdatendefinition:** Anstatt eine Parameter Data Area zu definieren, können Parameter auch direkt definiert werden; siehe ***Definition von Parameterdaten*** weiter unten.
`END-DEFINE`	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden des `DEFINE DATA`-Statements benutzt werden.

Definition von Parameterdaten

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



Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
level	**Level-Nummer:** 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. 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.

Syntax-Element	Beschreibung
	Eine Gruppe kann aus weiteren Gruppen bestehen. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.
group-name	**Gruppenname:** Der Name einer Gruppe. Der Name muss den Regeln zur Definition eines Natural-Variablennamens entsprechen. Siehe auch die folgenden Abschnitte: ■ *Namen von Benutzervariablen* in der Dokumentation *Natural benutzen*. ■ *Datenstrukturen kennzeichnen* im *Leitfaden zur Programmierung*.
array-definition	**Definition von Array-Dimensionen:** Mit *array-definition* 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***.
redefinition	**Redefinition:** Mit *redefinition* können Sie eine Gruppe oder ein einzelnes Feld oder eine einzelne Variable (d.h. Skalar oder Array) redefinieren. Siehe ***Redefinition***. **Anmerkung:** In einer Parameter Data Area ist eine Redefinition von Gruppen nur innerhalb eines REDEFINE-Blocks zulässig.
variable-name	**Name der Variablen:** Der der Variablen zuzuweisende Name. Es gelten die Regeln für Natural-Variablennamen. Informationen zu Namenskonventionen für Benutzervariablen, siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.
format-length	**Format/Länge:** Das Format und die Länge des Feldes. Informationen zu Format und Länge für Benutzervariablen siehe *Format und Länge von Benutzervariablen* im *Leitfaden zur Programmierung*.
HANDLE OF OBJECT	**Handle of Object:** Wird im Zusammenhang mit NaturalX benutzt. Ein Handle kennzeichnet ein Dialogelement im Code und wird in Handle-Variablen gespeichert. Weitere Informationen siehe *NaturalX* im *Leitfaden zur Programmierung*.
A, U or B	**Datentyp:** Alphanumerisch (A), Unicode (U) oder Binär (B) für dynamische Variablen.
DYNAMIC	**DYNAMIC-Option:** Ein Parameter kann als DYNAMIC definiert werden.

Syntax-Element	Beschreibung
	Weitere Informationen zur Verarbeitung von dynamischen Variablen, siehe *Dynamische Variablen* im *Leitfaden zur Programmierung*.
	Call-Modus: Je nachdem, ob der Call-By-Reference- oder Call-By-Value-Modus benutzt wird, gilt die betreffende Übertragungsart. Weitere Informationen siehe `CALLNAT`- Statement.
`(without BY VALUE)`	**Call-By-Reference-Modus:** Ohne `BY VALUE` (gilt standardmäßig) wird ein Parameter durch Referenzierung seiner Adresse („By Reference“) an ein Subprogramm, eine Subroutine oder eine Function 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/Function.
`BY VALUE`	**Call-By-Value-Modus:** Mit `BY VALUE` wird ein Parameter direkt als Wert (*`by value`*) an ein Subprogramm, eine Subroutine oder eine Function übergeben; d.h. statt der Adresse des Parameters wird der Wert selbst übergeben. Das bedeutet, das Feld in dem Subprogramm, der Subroutine oder der Function braucht nicht dasselbe Format und dieselbe Länge zu haben wie der Parameter beim `CALLNAT`-/`PERFORM`-Statement oder im Function Call. Format und Länge müssen lediglich datenübertragungskompatibel sein gemäß den *Regeln für arithmetische Operationen* und den *Kompatibilitätsregeln zur Datenübertragung* (siehe *Leitfaden zur Programmierung*). Mit `BY VALUE` können Sie zum Beispiel die Länge eines Feldes in einem Subprogramm, einer Subroutine oder Function vergrößern (falls eine Erweiterung des Subprogramms bzw. der Subroutine dies erforderlich machen sollte), ohne deswegen auch die Objekte, die das Subprogramm, die Subroutine bzw. die Function aufrufen, anpassen zu müssen.
	Beispiel für `BY VALUE`:

	`* Program`
	`DEFINE DATA LOCAL`
	`1 #FIELDA (P5)`
	`...`
	`END-DEFINE`
	`...`
	`CALLNAT 'SUBR01' #FIELDA`
	`...`

	`* Subroutine SUBR01`
	`DEFINE DATA PARAMETER`
	`1 #FIELDB (P9) BY VALUE`
	`END-DEFINE`
	`...`

`BY VALUE RESULT`	**Call-By-Value-Result-Modus:** Während `BY VALUE` für die Übergabe eines Parameters an ein Subprogramm, eine Subroutine oder eine Function gilt, bewirkt die Angabe von `BY VALUE RESULT` die Übergabe des Parameterwertes in beide Richtungen; d.h. der Parameterwert selbst wird vom aufrufenden Objekt an das Subprogramm, die Subroutine oder die Function

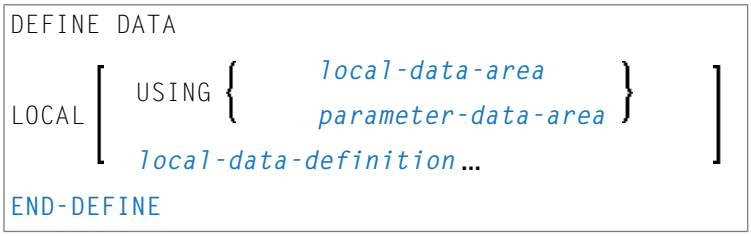
Syntax-Element	Beschreibung
	übergeben, und bei der Rückkehr zum aufrufenden Objekt wird der Parameterwert selbst von dem Subprogramm, der Subroutine bzw. der Function an das aufrufende Objekt zurückgegeben. Wenn Sie `BY VALUE RESULT` verwenden, müssen Format und Länge der betreffenden Felder in beide Richtungen datenübertragungskompatibel sein.
`OPTIONAL`	**Optionale Parameter:** Bei einem *ohne* `OPTIONAL` definierten Parameter (Standard) muss ein Wert vom aufrufenden Objekt übergeben werden. Bei einem *mit* `OPTIONAL` definierten Parameter muss ein Wert vom aufrufenden Objekt an diesen Parameter nicht unbedingt übergeben werden. Im aufrufenden Objekt wird die Notation *n*`X` benutzt, um Parameter anzugeben, die übersprungen werden, d.h. für die keine Werte übergeben werden. Bei der `SPECIFIED`-Option können Sie zur Laufzeit ermitteln, ob ein optionaler Parameter definiert worden ist oder nicht.

Siehe auch *Übereinstimmende Formatangaben bei Array-Dimensionen* im *Leitfaden zur Programmierung*.

34 Definition von Local Data

- Funktion DEFINE DATA LOCAL 248
- Einschränkung DEFINE DATA LOCAL 248
- Syntax-Beschreibung DEFINE DATA LOCAL 248

Allgemeine Syntax von `DEFINE DATA LOCAL`:



Dieser Abschnitt behandelt folgende Themen:

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

Funktion DEFINE DATA LOCAL

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 *`local-data-definition`*-Syntax (siehe ***Lokale Daten-Definition***)

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

Einschränkung DEFINE DATA LOCAL

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

Syntax-Beschreibung DEFINE DATA LOCAL

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

local-data-area	**LDA-Name:**
	
	Geben Sie den Namen der zu referenzierenden Local Data Area (LDA) an.
	
	Eine Local Data Area wird mit dem *Datenbereich-Editor (Data Area Editor)* erstellt. Sie enthält vordefinierte Datenelemente, die in das `DEFINE DATA LOCAL`-Statement übernommen werden können.
	
	Sie können mehr als eine Data Area referenzieren; in diesem Fall müssen Sie die reservierten Wörter `LOCAL` und `USING` wiederholen, zum Beispiel:
	
	`DEFINE DATA`
	`  LOCAL USING DATX_L`
	`  LOCAL USING DATX_P`
	
	`  ...`
	`END-DEFINE ;`
	
	Weitere Informationen siehe auch *Felder in einer separaten Data Area und Local Data Area* und *Local Data Area, Beispiel 2* im *Leitfaden zur Programmierung*.

parameter-data-area	**PDA-Name:** Geben Sie den Namen der zu referenzierenden Parameter Data Area (PDA) an. **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. Eine Parameter Data Area wird mit dem *Datenbereich-Editor (Data Area Editor)* erstellt. Sie enthält vordefinierte Datenelemente, die in das `DEFINE DATA LOCAL`-Statement übernommen werden können.
local-data-definition	**Lokale Daten-Definition:** Im Abschnitt ***Lokale Daten-Definition*** finden Sie Informationen, wie Sie Elemente oder Felder innerhalb des Statements selbst definieren können, das heißt, ohne dazu eine LDA oder eine PDA zu benutzen.
`END-DEFINE`	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden des `DEFINE DATA`-Statements benutzt werden.

Lokale Daten-Definition

Lokale Daten können direkt in einem Programm oder Subprogramm definiert werden. In diesem Fall gilt folgende Syntax:

```
{ level { group-name [(array-definition)]
        { variable-definition
        { view-definition
        { redefinition
```

Weitere Informationen siehe

- ***Beispiel 1 - DEFINE DATA LOCAL*** (Lokale 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:

Syntax-Element	Beschreibung
level	**Level-Nummer:** 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. 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. Eine Gruppe kann aus anderen 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.
group-name	**Gruppenname:** Der Name einer Gruppe. Der Name muss den Regeln zur Definition eines Natural-Variablennamens entsprechen. Siehe auch die folgenden Abschnitte: ■ *Namen von Benutzervariablen* in der Dokumentation *Natural benutzen*. ■ *Datenstrukturen kennzeichnen* im *Leitfaden zur Programmierung*.

Syntax-Element	Beschreibung
array-definition	**Definition von Array-Dimensionen:** Mit *array-definition* definieren Sie die untere und obere Grenze einer Dimension in einer Array-Definition. Siehe ***Definition von Array-Dimensionen***.
variable-definition	**Definition einer Variablen:** Die *variable-definition* dient zur Definition einer einzelnen Variablen (oder Feldes), die aus einem Wert (Skalar) oder mehreren Werten (Array) bestehen kann. Siehe ***Definition von Variablen***.
view-definition	**Definition einer Datenbanksicht:** Die *view-definition* wird benutzt, um eine View (Datenbanksicht) mit Bestandteilen aus einem Datendefinitionsmodul (DDM) zu definieren. Siehe ***View-Definition***.
redefinition	**Redefinition:** Eine *redefinition* kann zur Redefinition einer Gruppe, eines View-Felds, eines DDM-Felds oder eines einzelnen Felds oder einer einzelnen Variablen benutzt werden (d.h. Skalar oder Array). Siehe ***Redefinition***.

35 Definition von anwendungsunabhängigen Variablen

- Funktion DEFINE DATA INDEPENDENT 254
- Syntax-Beschreibung DEFINE DATA INDEPENDENT 254

Allgemeine Syntax von `DEFINE DATA INDEPENDENT`:

```
DEFINE DATA
 INDEPENDENT [aiv-data-definition ...]
END-DEFINE
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion DEFINE DATA INDEPENDENT

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 DEFINE DATA INDEPENDENT

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
aiv-data-definition	**AIV Data Definition:** 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	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort END-DEFINE muss zum Beenden des DEFINE DATA-Statements benutzt werden.

AIV Data Definition

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

Syntax-Element-Beschreibung:

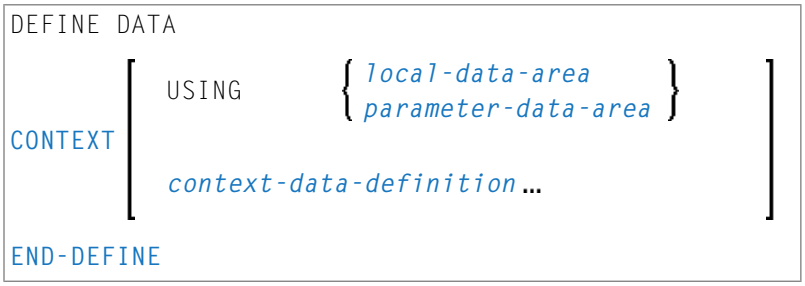
Syntax-Element	Beschreibung
level	**Level-Nummer:** Eine applikationsunabhängige Variable muss auf Level 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.
variable-definition	**Definition einer Variablen:** Eine Variablen-Definition wird zur Definition eines einzelnen Feldes 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.
redefinition	**Redefinition:** Mit einer *redefinition* 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.

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

36 Definition von Kontext-Variablen für den Natural RPC

- Funktion DEFINE DATA CONTEXT 258
- Einschränkungen DEFINE DATA CONTEXT 259
- Syntax-Beschreibung DEFINE DATA CONTEXT 259

Allgemeine Syntax von `DEFINE DATA CONTEXT`:



Dieser Abschnitt 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 DEFINE DATA CONTEXT

Das `DEFINE DATA CONTEXT`-Statement wird im Zusammenhang mit dem Natural RPC (Remote Procedure Call) verwendet. Es dient dort zur Definition von als Kontextvariablen bekannten Variablen, die für mehrere entfernte (remote) Subprogramme innerhalb einer Konversation zur Verfügung stehen sollen, ohne dass die Variablen explizit als Parameter mit den entsprechenden `CALLNAT`-Statements übergeben werden müssen.

Eine Kontextvariable wird über ihren Namen referenziert. Ihr Inhalt wird von allen in einer Konversation ausgeführten Natural-Objekten, diesen Namen referenzieren, geteilt (gemeinsam genutzt). Die Variable wird vom ersten ausgeführten Natural-Objekt zugeordnet, das die Definition der Variablen enthält, und wird freigegeben, wenn die Konversation beendet ist.

Eine Kontextvariable wird nicht mit Subprogrammen geteilt, die innerhalb der Konversation aufgerufen werden. Wenn ein solches Unterprogramm oder einer seiner Aufrufer eine Kontextvariable referenziert, wird für diese Variable ein getrennter Speicherbereich zugeordnet.

Kontextvariablen können auch in einem nicht-konversationellen `CALLNAT` benutzt werden. In diesem Fall existieren die Kontextvariablen nur während eines einzelnen Aufrufs dieses `CALLNAT`. Die Variable wird zugeordnet, wenn das entfernte (remote) Subprogramm gestartet wird, und die Zuordnung wird aufgehoben, wenn es endet. Der Inhalt wird von allen Programmierobjekten (jedoch nicht von Subprogrammen) gemeinsam verwendet, die von diesem nicht-konversationellen `CALLNAT` ausgeführt werden.

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

Weitere Informationen siehe *Einen Konversationskontext definieren* in der *Natural RPC (Remote Procedure Call)*-Dokumentation.

Einschränkungen DEFINE DATA CONTEXT

Eine Kontextvariable muss auf Level 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.

Syntax-Beschreibung DEFINE DATA CONTEXT

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

`USING` *`local-data-area`*	**LDA-Name:**
	
	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 `CONTEXT` und `USING` wiederholen, zum Beispiel:
	
	`DEFINE DATA`
	`  CONTEXT USING DATX_L`
	`  CONTEXT USING DATX_P`
	
	`  ...`
	`END-DEFINE ;`
	
	Weitere Informationen siehe *Felder in einer separaten Data Area definieren* im *Leitfaden zur Programmierung*.

`USING` *`parameter-data-area`*	**PDA-Name:** Eine Parameter Data Area enthält Datenelemente, die als Parameter in einem Subprogramm, einer externen Subroutine oder in einem Dialog benutzt werden.
`context-data-definition`	**Kontextdaten-Definition** Kontextdaten können auch direkt innerhalb eines Programms oder einer Routine definiert werden. Bei einer Kontextdaten-Definition gilt die **weiter unten** gezeigte Syntax.
`END-DEFINE`	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden des `DEFINE DATA`-Statements benutzt werden.

Kontextdaten-Definition

Kontextdaten können direkt innerhalb eines Programms oder einer Routine definiert werden. Bei einer direkten Datendefinition gilt die folgende Syntax:

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

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

Syntax-Element	Beschreibung
level	**Level-Nummer:** Eine anwendungsunabhängige Variable muss auf Stufe (Level) 01 definiert werden. Andere Levels werden nur bei einer Redefinition benutzt.
variable-definition	**Definition einer Variablen:** Eine Variablendefinition wird zur Definition eines einzelnen Feldes oder einer einzelnen Variablen benutzt, die einen Wert (Skalar) oder mehrere Werte (Array) haben kann. Siehe ***Definition von Variablen***. **Anmerkung:** Die CONSTANT-Klausel darf in diesem Zusammenhang nicht benutzt werden.
redefinition	**Redefinition:** Mit einer Redefinition redefinieren Sie eine Gruppe, eine View (Datensicht), ein DDM-Feld oder ein einzelnes Feld oder eine einzelne Variable (d.h. ein Skalar oder ein Array). Siehe ***Redefinition***.

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

37 Definition von NaturalX-Objekten

- Funktion DEFINE DATA OBJECT 262
- Syntax-Beschreibung DEFINE DATA OBJECT 262

Allgemeine Syntax von `DEFINE DATA OBJECT`:

```
DEFINE DATA
                  [ USING { local-data-area        } ]
OBJECT            [       { parameter-data-area    } ]
                  [
                  [ local-data-definition ...       ]
END-DEFINE
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion DEFINE DATA OBJECT

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 DEFINE DATA OBJECT

Syntax-Element-Beschreibung:

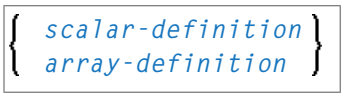
Syntax-Element	Beschreibung

`USING` *local-data-area*	**LDA-Name:**
	
	Eine LDA (Local Data Area) 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:
	
	`DEFINE DATA`
	`  OBJECT USING DATX_L`
	`  OBJECT USING DATX_P`
	
	`  ...`
	`END-DEFINE ;`
	
	Weitere Informationen siehe *Felddefinitionen im DEFINE DATA-Statement* im *Leitfaden zur Programmierung*.

Syntax-Element	Beschreibung
`USING` *`parameter-data-area`*	**PDA-Name:** 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.
`local-data-definition`	**Lokale Datendefinition:** Daten können auch direkt mit der im Abschnitt ***Lokale Datendefinition*** definiert werden.
`END-DEFINE`	**Ende des DEFINE DATA-Statements:** Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden des `DEFINE DATA`-Statements benutzt werden.

38 Definition von Variablen

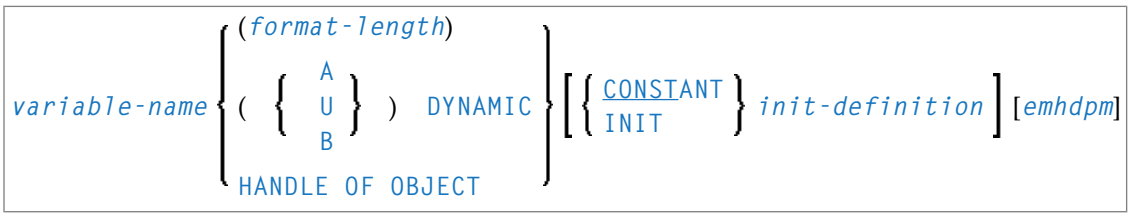
- Syntax-Beschreibung variable-definition 266



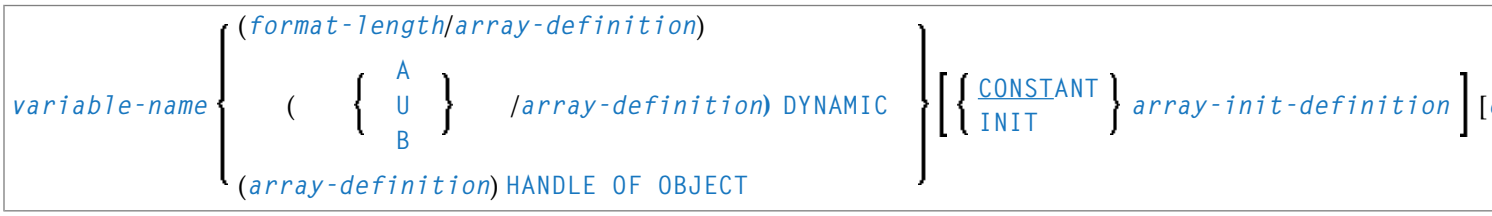
Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt ***Syntax-Symbole***.

Die *variable-definition* 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



Syntax-Beschreibung variable-definition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
variable-name	**Name der Variablen:** *variable-name* ist der der Variable zuzuweisende Name. Es gelten die Regeln für Natural-Variablennamen. Bei `DEFINE DATA INDEPENDENT` muss der Variablenname mit einem Plus-Zeichen (+) beginnen. Informationen zu Namenskonventionen für Benutzervariablen siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.
format-length	**Format/Längen-Definition** Format und Länge des Feldes.

Syntax-Element	Beschreibung
	Informationen zu Format/Längen-Definitionen von Benutzervariablen, siehe *Format und Länge von Benutzervariablen* im *Leitfaden zur Programmierung*.
HANDLE OF OBJECT	**Handle of Object:** Wird be NaturalX verwendet. Eine Handle kennzeichnet ein Dialogelement im Code und wird in Handle-Variablen gespeichert. Weitere Informationen siehe *NaturalX* im *Leitfaden zur Programmierung*.
A oder B oder U	**Datentyp:** Alphanumerisch (A), Unicode oder Binär (B) für dynamische Variablen.
array-definition	**Definition von Array-Dimensionen:** Mit *array-definition* definieren Sie die Unter- und Obergrenze einer Dimension in einer Array-Definition. Siehe ***Definition von Array-Dimensionen***.
DYNAMIC	**DYNAMIC-Option:** Ein Feld kann als dynamisch definiert werden. Weitere Informationen zur Verarbeitung von dynamischen Variablen siehe *Dynamische und große Variablen benutzen*.
CONSTANT	**CONSTANT-Option:** 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. Siehe auch *Felder definieren*, *Benutzerkonstanten Namens-Konstanten definieren* im *Leitfaden zur Programmierung*. **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 CONSTANT-Klausel darf nicht mit DEFINE DATA INDEPENDENT und DEFINE DATA CONTEXT benutzt werden. Die CONSTANT-Klausel kann nicht mit X-Arrays benutzt werden.
INIT	**INIT-Option:** Der Variablen bzw. dem Array soll ein Ausgangswert zugewiesen werden. Dieser Wert wird auch benutzt, wenn diese Variable bzw. dieses Array in einem RESET INITIAL-Statement referenziert wird. Wenn INIT nicht angegeben ist, wird ein Feld mit einem standardmäßigen Ausgangswert je nach seinem Format initialisiert (siehe Tabelle **Standard-Ausgangswerte**).

Syntax-Element	Beschreibung
	Siehe auch *Felder definieren*, *Ausgangswerte* im *Leitfaden zur Programmierung*. **Anmerkung:** Bei `DEFINE DATA INDEPENDENT` und `DEFINE DATA CONTEXT` wird die `INIT`-Klausel bei jedem ausgeführten Programmierobjekt ausgewertet, das diese Klausel enthält (nicht nur im Programmierobjekt, das die Variable zuweist). `INIT` funktioniert andes für globale Variablen. Die `INIT`-Klausel kann nicht für X-Arrays benutzt werden.
init-definition	**Initialwerte oder Konstantenwerte für eine Variable:** Mit der Option *init-definition* definieren Sie die Initialwerte oder Konstantenwerte für eine Variable. Siehe ***Definition eines Ausgangswerts***.
array-init-definition	**Initialwerte oder Konstantenwerte für ein Array:** Bei *array-init-definition* definieren Sie die Initialwerte oder Konstantenwerte für ein Array. Siehe ***Ausgangswerte/Konstanten-Werte für ein Array***.
emhdpm	**Parameter EM, HD, PM für Feld/Variable:** 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***.

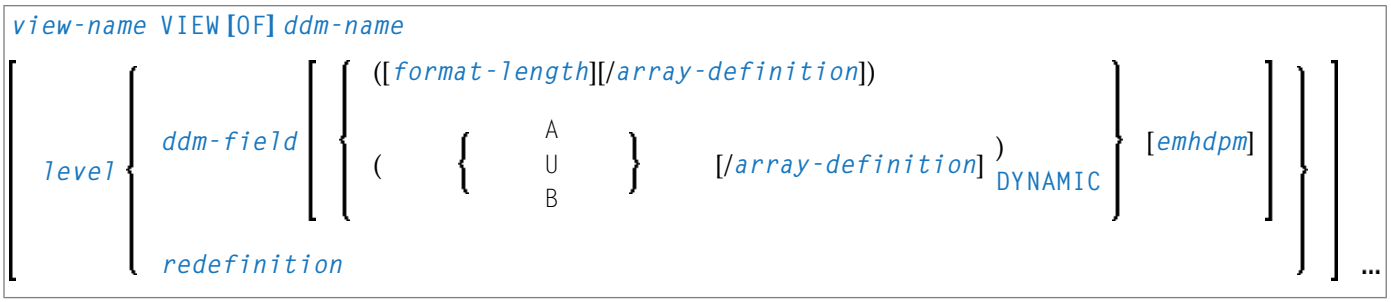
Standard-Ausgangswerte

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

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

39 View-Definition

- Syntax-Beschreibung view-definition 270



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

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 view-definition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
view-name	**View-Name:** Der Name, den die View (Datensicht) erhalten soll. Es gelten die Namenskonventionen für Natural-Variablen. Siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.
VIEW [OF] *ddm-name*	**DDM-Name:** Der Name des DDMs, aus dem die View gebildet wird.
level	**Level-Nummer:** 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.

Syntax-Element	Beschreibung
	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. Eine Gruppe kann ihrerseits Teil einer anderen Gruppe sein. Bei der Vergabe von Level-Nummern für eine Gruppe darf kein Level ausgelassen werden.
ddm-field	**DDM-Feldname:** Der im verwendeten DDM definierte Name eines Feldes. Bei der Definition einer View (Datensicht) für ein HISTOGRAM-Statement darf diese View lediglich den Deskriptor enthalten, den das HISTOGRAM-Statement benutzt.
redefinition	**Redefinition:** Eine *redefinition* kann zur Redefinition einer Gruppe, einer Views eines DDM-Felds oder eines einzelnen Feldes oder einer einzelnen Variable benutzt werden (d.h. Skalar oder Array). Siehe ***Redefinition***.
format-length	**Format/Längen-Definition** Format und Länge des definierten Feldes. Werden diese Angaben nicht gemacht, wird die Format-/Längendefinition aus dem DDM übernommen. Im Structured Mode muss die Definition von Format und Länge (wenn angegeben) dieselbe wie die vom DDM sein. Im Reporting Mode muss die Format-/Längendefinition kompatibel mit der im DDM sein.
A, U or B	**Datentyp:** Alphanumerisch (A), Unicode (U) oder binär (B) für dynamische Variablen. **Anmerkung:** 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.
array-definition	Abhängig vom benutzten Modus müssen Arrays (Periodengruppenfelder, multiple Felder) eventuell Informationen über ihre Ausprägungen aufnehmen. Siehe den Abschnitt ***Array-Definition in einer View*** weiter unten.
emhdpm	**Parameter EM, HD, PM für Feld/Variable:** 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***.
DYNAMIC	**DYNAMIC-Option:**

Syntax-Element	Beschreibung
	Definiert ein View-Feld als dynamisch. Weitere Informationen zur Verarbeitung von dynamischen Variablen siehe Abschnitt *Dynamische und große Variablen benutzen*.

Array-Definition in einer View (Datensicht)

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

- Array-Definition in einer View im Structured Mode
- Array-Definition in einer View (Datensicht) im Reporting Mode

Array-Definition in einer 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 einer View (Datensicht) 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).

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 einer View (Datensicht) 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 Fomat/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)

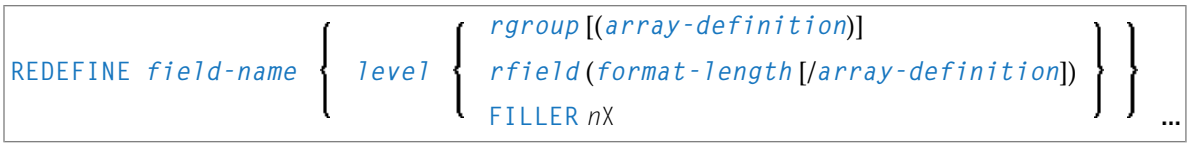
1 EMP2 VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE(A40)        /* ADDRESS LINE (1:1) IS ASSUMED

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

40 Redefinition

- Einschränkungen redefinition 276
- Syntax-Beschreibung redefinition 276



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

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

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

Einschränkungen redefinition

- Eine Redefinition eines View- oder DDM-Feldes für eine *parameter-data-definition* ist nicht möglich.
- Handles, X-Arrays und dynamische Variablen können nicht redefiniert werden und können nicht in einer Redefinitions-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 redefinition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
field-name	**Name des umzudefinierenden Feldes:** Der Name der Gruppe, des View-Feldes, des DDM-Feldes oder einzelnen Feldes, der/die/das redefiniert werden soll.
level	**Level-Nummer des umzudefinierenden Feldes:** 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.
rgroup	**Name der resultierenden Gruppe:**

Syntax-Element	Beschreibung
	Der Name der Gruppe, die sich aus der Redefinition ergibt. **Anmerkung:** Bei einer Redefinition innerhalb einer *view-definition* darf für *rgroup* kein Name vergeben werden, der schon als Feldname im zugrundeliegenden DDM existiert.
rfield	**Name des resultierenden Feldes:** Der Name des Feldes, das sich aus der Redefinition ergibt. **Anmerkung:** Bei einer Redefinition innerhalb einer *view-definition* darf für *rfield* kein Name vergeben werden, der schon als Feldname im zugrundeliegenden DDM existiert.
format-length	**Format/Länge des resultierenden Feldes:** Format und Länge von (*rfield*).
array-definition	**Definition von Array-Dimensionen:** Bei einer Array-Definition definieren Sie die Unter- und Obergrenze einer Dimension in einer Array-Definition. Siehe den Abschnitt *Definition von Array-Dimensionen*.
FILLER *n*X	**Definition von Füllbytes:** Mit dieser Notation können Sie in dem Feld, das redefiniert wird, *n* 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:

```
   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
   ...
```

Beispiel 2:

```
  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
  ...
```

Beispiel 3:

```
  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
  ...
```

41 Definition von Array-Dimensionen

- Syntax-Beschreibung array-dimension-definition 280

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

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

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.

Syntax-Beschreibung array-dimension-definition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
bound	**Unter- und Obergrenze:** Als Grenze (*bound*) kann eines der folgenden Elemente verwendet werden: ■ eine numerische Ganzzahl-Konstante ■ eine vorher definierte Namens-Konstante ■ (bei Datenbank-Arrays) eine vorher definierte Benutzervariable ■ ein Stern (*); dieser definiert einen erweiterbaren *bound*, auch bekannt als X-Array. Wenn nur ein Bound angegeben ist, stellt der Wert die Obergrenze dar, und die Untergrenze wird als 1 angenommen.

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
```

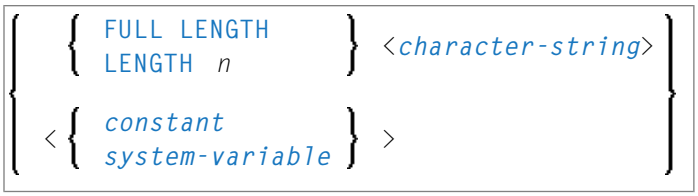
Um Laufzeitfehler zu vermeiden, sollte die maximale Anzahl der Ausprägungen eines solchen Arrays über einen anderen Parameter an das Subprogramm/die Subroutine/die Function übergeben werden. Als Alternative dazu können Sie die Systemvariable `*OCCURRENCE` benutzen.

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

- Einschränkung init-definition 284
- Syntax-Beschreibung init-definition 284



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

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 init-definition

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

Syntax-Beschreibung init-definition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
<constant>	**Konstantenwert-Option:** Der Konstantenwert, mit der die Variable initialisiert werden soll, bzw. der Konstantenwert, der dem Feld fest zugewiesen wird. Informationen zu Konstanten siehe Abschnitt *Benutzerkonstanten* im *Leitfaden zur Programmierung*.
<system-variable>	**Systemvariablenwert-Option:** Als Ausgangswert einer Variablen können Sie auch den Wert einer Natural-Systemvariablen benutzen, zum Beispiel:

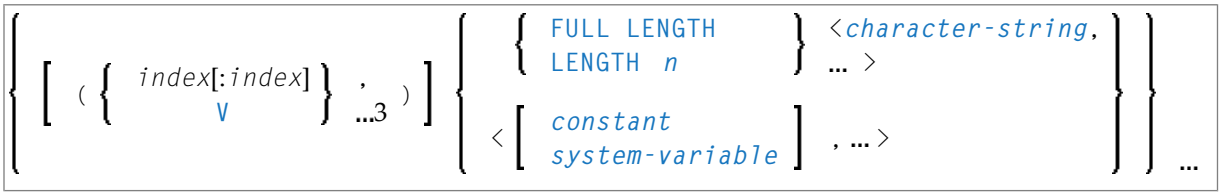
Syntax-Element	Beschreibung
	``` DEFINE DATA LOCAL 1 #MYDATE (D) INIT <*DATX> END-DEFINE ``` **Anmerkung:** Wenn die Variable in einem `RESET INITIAL`-Statement referenziert wird, wird die Systemvariable neu ausgewertet; d.h. die Variable wird nicht auf den Wert zurückgesetzt, den die Systemvariable zu Beginn der Programmausführung hatte, sondern auf den Wert, den sie zum Zeitpunkt der Ausführung des Statements `RESET INITIAL` hat.

`FULL LENGTH <character-string>`
 	
`LENGTH n <character-string>`
	Zeichenkette-Option für alphanumerische Variablen/Unicode-Variablen:
	
	Bei einer Variablen mit Natural-Datenformat A oder U kann eine Zeichenkette (*character-string*, z.B. `'ABC'`) als Ausgangswert benutzt werden, der das variable Feld ganz oder teilweise füllt.
	
	Eine Zeichenkette (*character-string*) ist eine Konstante mit Natural-Datenformat A oder U, siehe *Alphanumerische Konstanten* and *Unicode-Konstanten* im *Leitfaden zur Programmierung*.
	
	FULL LENGTH-Option:
	
	Bei der `FULL LENGTH`-Option wird eine Zeichenkette (*character-string*) wiederholt in das angegebene Feld verschoben, bis das Feld vollständig gefüllt ist.
	
	Im folgenden Beispiel wird das gesamte Feld mit Sternen (*) gefüllt.
	
	``
	DEFINE DATA LOCAL
	1 #FIELD (A25) INIT FULL LENGTH <'*'>
	END-DEFINE
	``
	LENGTH Option:
	
	Bei der `LENGTH` *n*-Option wird eine Zeichenkette (*character-string*) wiederholt in das angegebene Feld verschoben, bis die ersten *n* Stellen des Feldes gefüllt sind.
	
	Im folgenden Beispiel werden die ersten 4 Stellen des Feldes mit Ausrufungszeichen (!) gefüllt.
	
	``
	DEFINE DATA LOCAL
	1 #FIELD (A25) INIT LENGTH 4 <'!'>
	END-DEFINE
	``

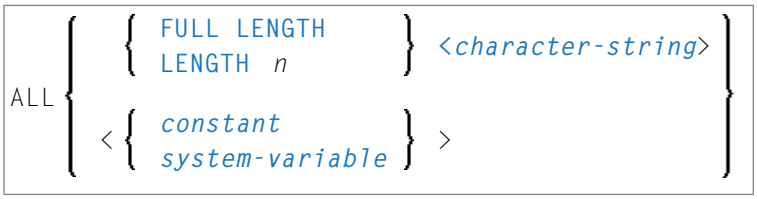
43 Ausgangswerte/Konstanten-Werte für ein Array

- Einschränkung array-init-definition 288
- Syntax-Beschreibung array-init-definition 289

Für ausgewählte Ausprägungen:



Für alle Ausprägungen:



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

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 array-init-definition

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

Syntax-Beschreibung array-init-definition

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ALL	**ALL-Option:** Alle Ausprägungen in allen Dimensionen des Arrays werden mit dem gleichen Wert initialisiert. Die ALL-Option kann mit keiner anderen Initialisierungsdefinition kombiniert werden.

index	**Index-Option:**
	
	Die im *index* angegebenen Ausprägungen des Arrays werden initialisiert.
	
	Wenn Sie einen *index* angeben, dürfen Sie mit *constant* nur einen einzigen Wert angeben (*constant* oder *system-variable*), der allen angegebenen Ausprägungen zugewiesen wird.
	
	Beispiele:
	
	`DEFINE DATA LOCAL ↩`
	
	`1 #FLD1 (A4/1:4)  INIT (1:3) <'A'>  /* A fills occurrences ↩ (1:3)`
	`1 #FLD2 (A4/1:4)  INIT (*)  <'B'>  /* B fills all occurrences`
	`1 #FLD3 (A4/1:2,1:4) INIT (2,3) <'C'>  /* C fills occurrence ↩ (2,3)`
	`END-DEFINE`

V	**V-Notation:** Die spezielle Index-Notation V wird verwendet, um eine konsekutive Folge von Array-Ausprägungen mit individuellen Werten zu füllen (*constant* oder *system-variable*). Sie können die V-Notation nur für eine Dimension eines Array angeben. Die Anzahl der zur Verfügung gestellten Werte darf die Anzahl der Ausprägungen der angegebenen Dimension übersteigen. Sie können die V-Notation bei einem eindimensionalen Array weglassen, weil dann der V-Index standardmäßig verwendet wird. Das folgende Beispiel zeigt, welche Werte welche Ausprägungen füllen, wenn V benutzt wird:

Syntax-Element	Beschreibung

	`DEFINE DATA LOCAL`
	`1 #FLD4 (A4/1:3)  INIT (V)  <'A','B'>  /* A fills (1) B ↩ fills (2)`
	`1 #FLD5 (A4/1:2,1:3) INIT (1,V) <'C','D'>  /* C, D fill ↩ (1,1:2)`
	`(2,V) <'F','G','H'> /* F, G, H fill ↩ (2,1:3)`
	`END-DEFINE`

constant	**Konstantenwert-Option:**
	
	Die Konstante (Wert), mit der das Array initialisiert werden soll.
	
	Ausprägungen, für die Sie keine Werte angeben, werden mit einem **Standardwert** initialisiert.
	
	In einer Liste aufeinander folgender Ausprägungen können Sie einzelne Ausprägungen überspringen, indem Sie nur Kommas (,) angeben. Sie müssen aber die Liste mit einem bestimmten Wert für die letzte Ausprägung abschließen.
	
	Weitere Informationen zur Definition von Konstanten finden Sie im Abschnitt *Benutzerkonstanten* im *Leitfaden zur Programmierung*.
	
	Anmerkung: Multiple Konstantenwerte/Systemvariablen müssen entweder durch das Input-Begrenzungszeichen (wie mit dem Session-Parameter `ID` angegeben) oder durch ein Komma (,) voneinander abgetrennt werden. Falls in der Werteliste Zahlen vorhanden sind und darin ein Komma als Dezimalstellenzeichen (mit dem Session-Parameter `DC`) definiert ist, müssen Sie entweder das Komma vom Wert durch ein zusätzliches Leerzeichen abtrennen oder das Eingabebegrenzungszeichen benutzen.
	
	Beispiel mit Input-Begrenzungszeichen-Einstellung `ID=;` und `DC=,`:
	
	`DEFINE DATA LOCAL`
	`1 #FLD1 (A4/1:3)  INIT <'A',,'C'>`
	`1 #NUM1 (N4,2/1:3) INIT <1 , 2 , 3>`
	`1 #NUM2 (N4,2/1:3) INIT <1;2;3>`
	`END-DEFINE`

system-variable	**Systemvariablenwert-Option:** Als Ausgangswert können Sie einem Array auch den Wert einer Natural-Systemvariablen zuweisen. Siehe auch die *Anmerkung* zur Konstantenwert-Option.
`FULL LENGTH` *<character-string>* `LENGTH` *n* *<character-string>*	**Zeichenketten-Option für alphanumerische Variablen/Unicode-Variablen:** Bei einer Variablen mit Natural-Datenformat A oder U kann eine Zeichenkette (*character-string*, z.B. `'ABC'`) als Ausgangswert benutzt werden, der das variable Feld ganz oder teilweise füllt.

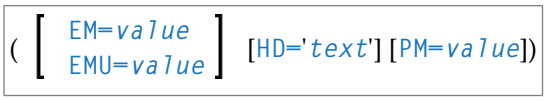
Syntax-Element	Beschreibung

	Eine Zeichenkette (*character-string*) ist eine Konstante mit Natural-Datenformat A oder U, siehe *Alphanumerische Konstanten* and *Unicode-Konstanten* im *Leitfaden zur Programmierung*.
	
	FULL LENGTH-Option:
	
	Bei der FULL LENGTH-Option wird eine Zeichenkette (*character-string*) wiederholt in die angegebene Array-Ausprägung verschoben, bis das Feld vollständig gefüllt ist.
	
	LENGTH Option:
	
	Bei der LENGTH *n*-Option wird eine bestimmte Zeichenkette (*character-string*) wiederholt in die angegebene Array-Ausprägung verschoben, bis die ersten *n* Stellen des Feldes gefüllt sind.
	
	Im folgenden Beispiel wird gezeigt, mit welchen Werten welche Ausprägungen gefüllt werden:
	
	`DEFINE DATA LOCAL`
	`1 #FLD1 (A6/1:3) INIT ALL FULL LENGTH <'X'>  /* XXXXXX in all occ.`
	`1 #FLD2 (A6/1:3) INIT ALL  LENGTH 5 <'NO'>  /* NONON in all occ.`
	`1 #FLD3 (A6/1:3) INIT (1:2) LENGTH 4 <'AB'>  /* ABAB in occ (1:2)`
	`1 #FLD4 (A6/1:3) INIT (V) FULL LENGTH <'X','Y'>/* XXXXXX in occ. (1),`
	`/* YYYYYY in occ. (2)`
	`END-DEFINE`
	
	Innerhalb einer *array-init-definition* dürfen Sie nur FULL LENGTH oder LENGTH *n* angeben; beide Notation dürfen nicht gemischt verwendet werden.

Weitere Beispiele für die Zuweisung von Ausgangswerten zu Arrays siehe ***Beispiel 2 - DEFINE DATA LOCAL (Array-Definition/Initialisierung)***.

44 Parameter EM, HD, PM für Feld/Variable

- Syntax-Beschreibung emhdpm 294



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

Mit der Option *emhdpm* 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 emhdpm

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

45 Beispiele für die Benutzung des DEFINE DATA-Statements

- Beispiel 1 - DEFINE DATA LOCAL (Lokale Daten-Definition) 296
- Beispiel 2 - DEFINE DATA LOCAL (Array-Definition/Initialisierung) 296
- Beispiel 3 - DEFINE DATA (View-Definition, Array-Redefinition) 300
- Beispiel 4 - DEFINE DATA (Global, Parameter und Local Data Areas) 301
- Beispiel 5 - DEFINE DATA (Initialisierung) 302
- Beispiel 6 - DEFINE DATA (Variables Array mit (1:V)) 302

Die folgenden Themen werden behandelt:

Beispiel 1 - DEFINE DATA LOCAL (Lokale 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 #AO1 (A5/1:4) INIT        <'A','B',,'D'>
1 #AO2 (A5/1:4) INIT  (V)   <'A','B'>
                      (4)   <'D'>
1 #AO3 (A5/1:4) INIT  (*)   <'A'>
1 #AO4 (A5/1:4) INIT  (2)   <'B'>
                      (3)   <'C'>
1 #AO5 (A5/1:4) INIT  (2:3) <'X'>
```

```
                       (4)   <'D'>
1 #A06 (A5/1:4) INIT   (*)   <'X'>
                       (3)   <'C'>
**-----------------------------------------
1 #A10 (A5/1:4) INIT         FULL LENGTH <'X'>
1 #A11 (A5/1:4) INIT         FULL LENGTH <,'B',,'D'>
1 #A12 (A5/1:4) INIT  (V)    FULL LENGTH <'A','B'>
1 #A13 (A5/1:4) INIT  (2)    FULL LENGTH <'B'>
1 #A14 (A5/1:4) INIT  (*)    FULL LENGTH <'X'>
**-----------------------------------------
1 #A20 (A5/1:4) INIT         LENGTH 2    <'A'>
1 #A21 (A5/1:4) INIT  (2)    LENGTH 2    <'B'>
                       (3)   LENGTH 3    <'C'>
1 #A22 (A5/1:4) INIT  (3:4)  LENGTH 2    <'X'>
1 #A23 (A5/1:4) INIT  (V)    LENGTH 2    <,'B',,'D'>
**-----------------------------------------
1 #A30 (A5/1:4) INIT  ALL              <'Z'>
1 #A31 (A5/1:4) INIT  ALL FULL LENGTH <'Z'>
1 #A32 (A5/1:4) INIT  ALL LENGTH 2    <'Z'>
**=========================================
1 #B01 (A5/1:2,1:4) INIT (2,V)   <'A','B',,'D'>
1 #B02 (A5/1:2,1:4) INIT (1,*)   <'X'>
                         (1,2)   <'B'>
                         (2,3)   <'F'>
1 #B03 (A5/1:2,1:4) INIT (1,1:2) <'X'>
                         (1,4)   <'Y'>
1 #B04 (A5/1:2,1:4) INIT (V,1)   <'A1','A2'>
1 #B05 (A5/1:2,1:4) INIT (V,*)   <'X','Y'>
**-----------------------------------------
1 #B10 (A5/1:2,1:4) INIT ALL   <'Z'>
1 #B11 (A5/1:2,1:4) INIT (1,*) FULL LENGTH <'Z'>
1 #B12 (A5/1:2,1:4) INIT (*,*) FULL LENGTH <'Z'>
1 #B13 (A5/1:2,1:4) INIT (1,*) LENGTH 2    <'Z'>
1 #B14 (A5/1:2,1:4) INIT (1,V) FULL LENGTH <'A',,'C'>
                         (2,V) FULL LENGTH <'E','F'>
1 #B15 (A5/1:2,1:4) INIT (1,*) FULL LENGTH <'X'>
                         (2,*) FULL LENGTH <'Y'>
                         (2,4) FULL LENGTH <'Z'>
*
END-DEFINE
**=========================================
WRITE 7X '(1)   (2)   (3)   (4)'
WRITE (AD=V) '=' #A01(*)
WRITE (AD=V) '=' #A02(*)
WRITE (AD=V) '=' #A03(*)
WRITE (AD=V) '=' #A04(*)
WRITE (AD=V) '=' #A05(*)
WRITE (AD=V) '=' #A06(*)
SKIP 1
WRITE (AD=V) '=' #A10(*)
WRITE (AD=V) '=' #A11(*)
WRITE (AD=V) '=' #A12(*)
```

```
WRITE (AD=V) '=' #A13(*)
WRITE (AD=V) '=' #A14(*)
SKIP 1
WRITE (AD=V) '=' #A20(*)
WRITE (AD=V) '=' #A21(*)
WRITE (AD=V) '=' #A22(*)
WRITE (AD=V) '=' #A23(*)
SKIP 1
WRITE (AD=V) '=' #A30(*)
WRITE (AD=V) '=' #A31(*)
WRITE (AD=V) '=' #A32(*)
SKIP 1
**=========================================
WRITE 6X '(1,1) (1,2) (1,3) (1,4)  (2,1) (2,2) (2,3) (2,4)'
WRITE (AD=V) '=' #B01(1,*) 2X #B01(2,*)
WRITE (AD=V) '=' #B02(1,*) 2X #B02(2,*)
WRITE (AD=V) '=' #B03(1,*) 2X #B03(2,*)
WRITE (AD=V) '=' #B04(1,*) 2X #B04(2,*)
WRITE (AD=V) '=' #B05(1,*) 2X #B05(2,*)
SKIP 1
WRITE (AD=V) '=' #B10(1,*) 2X #B10(2,*)
WRITE (AD=V) '=' #B11(1,*) 2X #B11(2,*)
WRITE (AD=V) '=' #B12(1,*) 2X #B12(2,*)
WRITE (AD=V) '=' #B13(1,*) 2X #B13(2,*)
WRITE (AD=V) '=' #B14(1,*) 2X #B14(2,*)
WRITE (AD=V) '=' #B15(1,*) 2X #B15(2,*)
**=========================================
END
```

Ausgabe des Programms DDAEX2:

```
Page        1

          (1)   (2)   (3)   (4)
#A01:     A     B           D
#A02:     A     B           D
#A03:     A     A     A     A
#A04:           B     C
#A05:           X     X     D
#A06:     X     X     C     X

#A10:     XXXXX
#A11:           BBBBB       DDDDD
#A12:     AAAAA BBBBB
#A13:           BBBBB
#A14:     XXXXX XXXXX XXXXX XXXXX

#A20:     AA
#A21:           BB    CCC
#A22:                 XX    XX
#A23:           BB          DD

#A30:     Z     Z     Z     Z
#A31:     ZZZZZ ZZZZZ ZZZZZ ZZZZZ
#A32:     ZZ    ZZ    ZZ    ZZ

          (1,1) (1,2) (1,3) (1,4) (2,1) (2,2) (2,3) (2,4)
#B01:                             A     B           D
#B02:     X     B     X     X                 F
#B03:     X     X           Y
#B04:     A1                      A2
#B05:     X     X     X     X     Y     Y     Y     Y

#B10:     Z     Z     Z     Z     Z     Z     Z     Z
#B11:     ZZZZZ ZZZZZ ZZZZZ ZZZZZ
#B12:     ZZZZZ ZZZZZ ZZZZZ ZZZZZ ZZZZZ ZZZZZ ZZZZZ ZZZZZ
#B13:     ZZ    ZZ    ZZ    ZZ
#B14:     AAAAA       CCCCC       EEEEE FFFFF
#B15:     XXXXX XXXXX XXXXX XXXXX YYYYY YYYYY YYYYY ZZZZZ
```

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                    SMITH                    SMITH
ENGLANDSVEJ 222          3152 SHETLAND ROAD       14100 ESWORTHY RD.
                         MILWAUKEE                MONTERREY
554349                   877-4563                 994-2260

SMITH                    SMITH
5 HAWTHORN               13002 NEW ARDEN COUR
OAK BROOK                SILVER SPRING
150-9351                 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`:

```
Page      1                                                  05-01-12  08:55:53

#FIELDA: HELLO      #FIELDB:    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
```

V

- 46 DEFINE FUNCTION 307
- 47 DEFINE PRINTER 315
- 48 DEFINE PROTOTYPE 329
- 49 DEFINE SUBROUTINE 337
- 50 DEFINE WINDOW 347
- 51 DEFINE WORK FILE 357

46 DEFINE FUNCTION

- Funktion DEFINE FUNCTION 308
- Syntax-Beschreibung DEFINE FUNCTION 308
- Beispiele DEFINE FUNCTION 312

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

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

Verwandtes Statement: `DEFINE PROTOTYPE`

Funktion DEFINE FUNCTION

Mit dem `DEFINE FUNCTION`-Statement können Sie eine benutzerdefinierte Funktion erstellen, die als ein Objekt des Typs Function gespeichert wird. Eine Function kann nur ein `DEFINE FUNCTION`-Statement enthalten.

Mit dem `DEFINE FUNCTION`-Statement wird der Name der Funktion, die Parameter, die lokalen und die anwendungsunabhängigen Variablen, das Ergebnis der Funktion und die Statements, die die Operationslogik bilden, definiert.

Weitere Informationen siehe folgende Abschnitte im *Leitfaden zur Programmierung*:

- Natural-Objekttyp Function
- *Function Call*

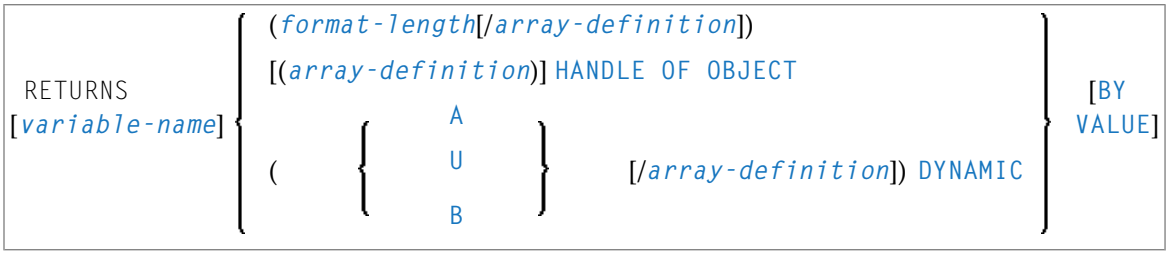
Syntax-Beschreibung DEFINE FUNCTION

Syntax-Element	Beschreibung
function-name	**Name der Funktion:** *function-name* ist der Name der aufzurufenden Funktion. Dabei gelten die im Kapitel *Namenskonventionen für Benutzervariablen* im Dokument *Natural benutzen* aufgeführten Regeln. *function-name* ist nicht notwendigerweise der gleiche Name wie der Name des gespeicherten Objekts, das die Definition der Function enthält. Sie dürfen denselben Funktionsnamen nicht zweimal in einer Library benutzen

Syntax-Element	Beschreibung
return-data-definition	Informationen zu dieser Klausel siehe ***Rückgabedatendefinition*** weiter unten.
function-data-definition	Informationen zu dieser Klausel siehe ***Funktionsdatendefinition*** weiter unten.
statement...	**Auszuführende(s) Statement(s):** Definiert die Operation(en), die ausgeführt werden soll(en), wenn die Function aufgerufen wird. Bildet die Function-Logik.
END-FUNCTION	**Ende des DEFINE FUNCTION-Statement:** Das für Natural reservierte Wort END-FUNCTION muss zum Beenden des DEFINE FUNCTION-Statements benutzt werden.

Rückgabedatendefinition

(*return-data-definition*)



Mit dieser Klausel wird das Format, die Länge und, falls zutreffend, die Array-Struktur des Ergebniswertes festgelegt, der von der Funktion zurückgegeben wird.

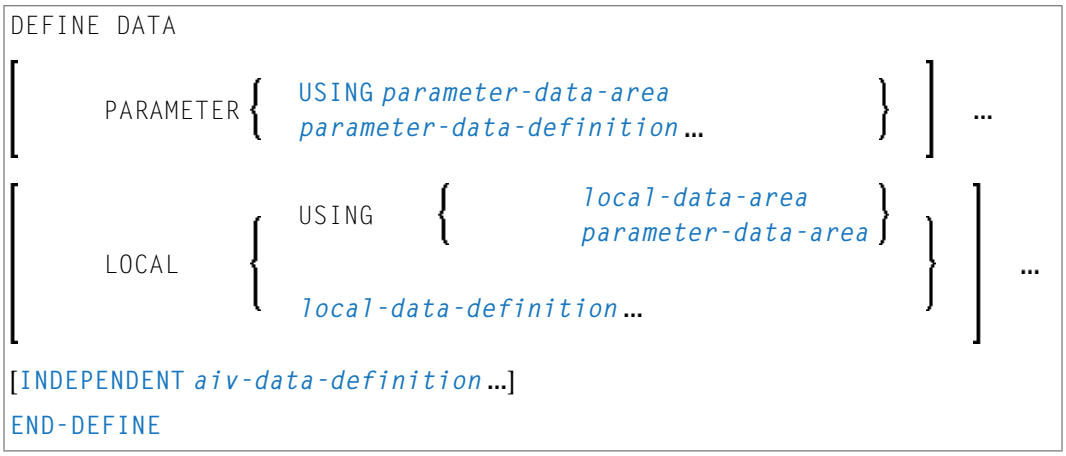
Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
variable-name	**Name des Rückgabewerts:** Hier kann optional ein Name angegeben werden, der benutzt werden kann, um auf das Rückgabefeld im Kode der Funktion zuzugreifen. Falls Sie keinen Namen angeben, wird der stattdessen der Name der Function verwendet.
format-length	**Format-/Länge-Definition:** Format und Länge des Ergebnisfeldes. Weitere Informationen siehe *Format und Länge von Benutzervariablen* im *Leitfaden zur Programmierung*.
array-definition	**Definition der Array-Dimensionen:** Falls das Ergebnisfeld ein Array-Feld ist, legen Sie hier die untere und obere Grenze einer Dimension fest.

Syntax-Element	Beschreibung
	Weitere Informationen siehe `DEFINE DATA`-Statement, ***Definition von Array-Dimensionen***.
`HANDLE OF OBJECT`	**Object-Handle:** Wird bei NaturalX verwendet. Weitere Informationen siehe *NaturalX* im *Programming Guide*.
`A`, `U` or `B`	**Datentyp:** Alphanumerisch (A), Unicode (U) oder binär (B) für ein dynamisches Ergebnis.
`DYNAMIC`	**Dynamische Variable:** Das Function-Ergebnis kann als `DYNAMIC` definiert werden. Weitere Informationen siehe *Dynamische Variablen* im *Leitfaden zur Programmierung*.
`BY VALUE`	**BY VALUE-Option:** Wenn Sie `BY VALUE` angeben, müssen das Format und die Länge des „sendenden“ Feldes (definiert in der *return-data-definition*-Klausel) und des „empfangenden“ Feldes (welches das Ergebnis der Function an der Stelle erhält, wo die Function aufgerufen wird) nur übertragungskompatibel sein. Das Format und die Länge des „empfangenden“ Felds werden wie folgt definiert: ■ entweder über eine explizite `(IR=)`-Klausel im Function Call ■ oder mit einem `DEFINE PROTOTYPE`-Statement ■ oder vom `RETURNS`-Feld des Function-Objekts übernommen, das schon existieren muss. Bezüglich der Datenübertragungskompatibilität gelten die Regeln in den Abschnitten *Regeln für arithmetische Operationen* und *Kompatibilitätsregeln zur Datenübertragung* im *Leitfaden zur Programmierung*. Wenn Sie `BY VALUE` nicht angeben, müssen das Format und die Länge des „empfangenden“ Feldes exakt mit den Eigenschaften des „sendenden“ Feldes übereinstimmen.

Funktionsdatendefinition

(*function-data-definition*)



Mit dieser Klausel werden die Parameter, die beim Aufrufen der Function mitgegeben werden sollen, und die von der Function benutzten Datenfelder festgelegt, zum Beispiel lokale Variable und anwendungsunabhängige Variable. Eine Global Data Area (GDA) kann nicht innerhalb der Function-Definition referenziert werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
PARAMETER USING *parameter-data-area*	**PDA-Name:** Der Name der Parameter Data Area (PDA), die Datenelemente enthält, die als Parameter in einem Function Call benutzt werden. Siehe auch ***Definition von Parameter Data*** in der DEFINE DATA-Statement-Beschreibung.
PARAMETER *parameter-data-definition*	**Parameterdatendefinition:** Anstatt eine Parameter Data Area zu definieren, können Sie Parameter auch direkt in einem Function Call definieren. Siehe auch ***Definition von Parameterdaten*** in der DEFINE DATA-Statement-Beschreibung.
LOCAL USING *local-data-area*	**LDA-Name:** Geben Sie den Namen der zu referenzierenden Local Data Area (LDA) an. Siehe auch ***Definition von Local Data*** in der DEFINE DATA-Statement-Beschreibung.
LOCAL USING *parameter-data-area*	**PDA-Name:** Geben Sie den Namen der zu referenzierenden Parameter Data Area (PDA) an.

Syntax-Element	Beschreibung
	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. Siehe auch ***Definition von Local Data*** in der `DEFINE DATA`-Statement-Beschreibung.
`LOCAL` *`local-data-definition`*	**Lokale Daten-Definition:** Informationen, wie Sie Elemente oder Felder innerhalb des Statements selbst definieren können, das heisst, ohne dazu eine LDA oder eine PDA zu benutzen, finden Sie im Abschnitt ***Lokale Daten-Definition*** in der `DEFINE DATA`-Statement-Beschreibung.
`INDEPENDENT` *`aiv-data-definition`*	**AIV-Daten-Definition:** Hier können Sie eine oder mehrere anwendungsunabhängige Variablen angeben. Siehe ***Definition von anwendungsunabhängigen Variablen*** in der `DEFINE DATA`-Statement-Beschreibung.
`END-DEFINE`	**Ende der Klausel:** Das für Natural reservierte Wort `END-DEFINE` muss zum Beenden der *`function-data-definition`*-Klausel benutzt werden.

Beispiele DEFINE FUNCTION

- Beispiel 1 - DEFINE FUNCTION
- Beispiel 2 - DEFINE FUNCTION mit Ergebniswert-Array

Beispiel 1 - DEFINE FUNCTION

```
** Example 'DFUEX1': DEFINE FUNCTION
************************************************************************
DEFINE FUNCTION F#FIRST-CHAR
  RETURNS #RESULT (A1)
  DEFINE DATA PARAMETER
    1 #PARM (A10)
  END-DEFINE
  /*
  #RESULT := #PARM        /* First character as return value.
END-FUNCTION
*
END
```

Die Funktion F#FIRST-CHAR wird in dem Beispielprogramm DPTEX2 in der Library SYSEXSYN verwendet. Siehe ***Beispiele*** in der DEFINE PROTOTYPE-Statement-Beschreibung.

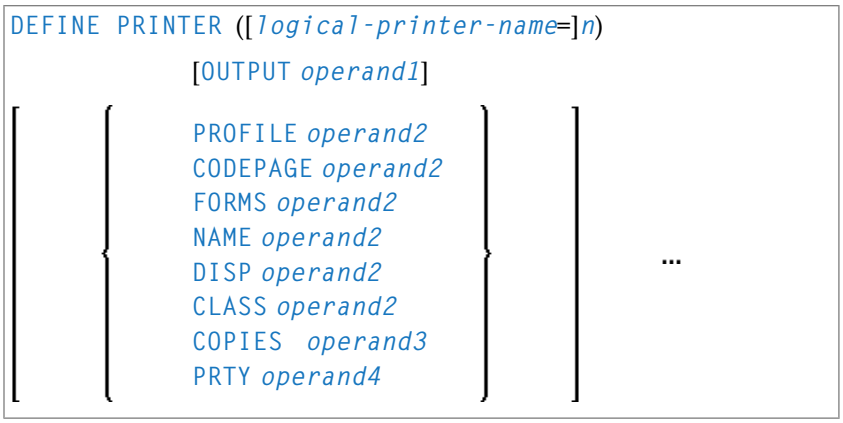
Beispiel 2 - DEFINE FUNCTION mit Ergebniswert-Array

```
** Example 'DFUEX2': DEFINE FUNCTION
************************************************************************
DEFINE FUNCTION F#FACTOR
  RETURNS (I2/1:3)
  DEFINE DATA PARAMETER
    1 #VALUE (I2)
  END-DEFINE
  /*
  F#FACTOR(1) := #VALUE * 1
  F#FACTOR(2) := #VALUE * 2
  F#FACTOR(3) := #VALUE * 3
  /*
END-FUNCTION
*
END
```

Die Funktion F#FACTOR wird in dem Beispielprogramm DPTEX1 in der Library SYSEXSYN verwendet. Siehe ***Beispiele*** in der DEFINE PROTOTYPE-Statement-Beschreibung.

47 DEFINE PRINTER

- Funktion DEFINE PRINTER 316
- Syntax-Beschreibung DEFINE PRINTER 317
- Druckername unter z/OS Batch, TSO und Server 320
- Druckername unter CICS 324
- Druckername unter Com-plete 324
- Druckername unter Com-plete/SMARTS 325
- Druckernamen unter Natural Advanced Facilities 325
- Druckername für zusätzliche Reports und Remote-Destinationen (Ausgabeziele) 325
- Beispiele DEFINE PRINTER 326



Dieser Abschnitt 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 DEFINE PRINTER

Das Statement `DEFINE PRINTER` dient dazu, einer Report-Nummer einen symbolischen Namen zuzuordnen und die Zuweisung eines Reports zu einer logischen Destination (Ausgabeziel, z.B. 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.

Weitere Informationen zum `DEFINE PRINTER`-Statement, siehe *Unicode- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*.

Syntax-Beschreibung DEFINE PRINTER

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A	U										ja	nein
operand2	C	S				A	U										ja	nein
operand3	C	S						N									ja	nein
operand4	C	S						N	P	I							ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(n)`	**Druckernummer:** Allen im Verlauf einer Session zu benutzenden Druckdateien muss im Voraus eine Zugriffsmethode zugewiesen werden, und zwar über den Subparameter `AM` (Typ der Zugriffsmethode) des Profilparameters `PRINT` oder automatisch mittels einer Definition in der JCL (nur bei Zugriffsmethode `AM=STD`). Die Druckernummer *n* kann ein Wert im Bereich von 0 - 31 sein. Dies ist die Nummer, die auch verwendet werden soll in einem `DISPLAY` / `WRITE` oder `CLOSE PRINTER`-Statement. Die Druckernummer 0 verweist auf den Hardcopy-Drucker. Einige Zugriffsmethoden unterstützen den Hardcopy-Drucker nicht, z.B. `AM=PC`.
logical-printer-name	**Logischer Druckername:** Als Option können Sie dem Drucker *n* einen logischen Druckernamen *logical-printer-name* zuweisen. Dieser Name kann für die *rep*-Notation in einem `DISPLAY`- oder `WRITE`-Statement benutzt werden. Die Namenskonventionen für *logical-printer-name* sind identisch mit den *Namenskonventionen für Benutzervariablen* (siehe Dokumentation *Natural benutzen*). Mehrere logische Namen können ein- und derselben Druckernummer zugewiesen werden. Im Gegensatz zum Wert des `OUTPUT`-Operanden (siehe unten), wird *logical-printer-name* zur Kompilierungszeit ausgewertet und ist deshalb unabhängig vom Programmsteuerungsablauf.
`OUTPUT` *operand1*	**Druckername:** Als *operand1* können Sie einen der folgenden Namen angeben: ■ den Druckernamen innerhalb des Online Spooling-Systems, ■ den der Druckernummer zuzuweisenden Druckdateinamen,

Syntax-Element	Beschreibung
	■ den Namen eines zusätzlichen Druckers oder ■ den Namen eines Remote-JES-Druckers. Siehe ***Druckername für zusätzliche Reports und Remote-Destinationen (Ausgabeziele)*** weiter unten. Der 8-Byte lange logische Druckername kann zu Anfang über den Subparameter `DEST` des Profilparameters `PRINT` definiert werden. Sein Standardwert ist vom Typ der Zugriffsmethode abhängig und kann durch *`operand1`* überschrieben werden. *`operand1`* kann 1 bis 253 Zeichen lang sein. Ist *`operand1`* eine Variable, muss die Länge mindestens 8 Bytes betragen. Sie können entweder einen Druckernamen oder einen logischen oder physischen Dataset-Namen angeben. Das mögliche Format ist abhängig von der Betriebssystemumgebung und von der über den Subparameter `AM` des Profilparameters `PRINT` für diese Druckernummer definierten Zugriffsmethode. Wenn der angegebene Name bereits für eine andere Druckernummer definiert ist, und dieser Drucker unbenutzt ist, d.h. den Status geschlossen hat, wird die Druckausgabe zu diesem Drucker weitergeleitet, wenn der Subparameter `ROUTE=ON` des Profilparameters `PRINT` für die spezifizierte Druckernummer angegeben wurde. Passt kein Druckername zu *`operand1`*, wird der unbenutzte Drucker mit der höchsten Nummer benutzt, und sein Name wird durch *`operand1`* überschrieben. Die Weiterleitung des Drucks erfolgt für den Benutzer unsichtbar und kann nicht mit dem `SYSFILE`-Kommando angezeigt werden. Informationen zu betriebssystemabhängigen oder TP-Monitor-abhängigen Drucker-Namenskonventionen befinden sich in folgenden Abschnitten: ■ **Druckername unter z/OS Batch, TSO und Server** ■ **Druckername unter CICS** ■ **Druckername unter Com-plete** ■ **Druckername unter Com-plete/SMARTS** ■ **Druckername unter Natural Advanced Facilities** ■ **Druckername für zusätzliche Reports und Remote-Destinationen (Ausgabeziele)**
	Bei den folgenden Klauseln können Sie Druckersteuer-Informationen angeben, die vom Spooling-System des TP-Monitors bzw. Betriebssystem interpretiert werden sollen. Sie können eine oder mehrere dieser Klauseln angeben, aber jede von diesen nur einmal.
`PROFILE` *`operand2`*	**Name der Druckersteuerzeichen-Tabelle:** Bei der `PROFILE`-Klausel geben Sie als *`operand2`* den Namen einer Druckersteuerzeichen-Tabelle an. Die maximal zulässige Länge für *`operand2`* ist 8 Bytes.

Syntax-Element	Beschreibung
	Sie definieren die Druckersteuerzeichen-Tabelle über den Profilparameter CCTAB (Definition der Drucker-Umschaltzeichensequenz). **Anmerkung:** Bei Natural Advanced Facilities kann die Druckersteuerzeichen-Tabelle online gepflegt werden (wie in der *Natural Advanced Facilities*-Dokumentation beschrieben).
CODEPAGE *operand2*	**Name der Codepage:** Mit CODEPAGE geben Sie als *operand2* den Namen (Format/Länge: A64) einer Codepage an, wie im NATCONFG-Modul angegeben. CODEPAGE wird ignoriert, wenn das Syntax-Element nicht für die entsprechende OUTPUT-Destination (Ausgabeziel) gilt.

Spooling-Systemparameter

Bei den im Folgenden aufgeführten Klauseln können Werte für Parameter des Spooling-Systems vom TP-Monitor angegeben werden. Den Standardwert dieser Klauseln können Sie mit den entsprechenden Subparametern des Profilparameters PRINT setzen (siehe *PRINT Schlüsselwortparameter für DEFINE PRINTER-Statement*).

Wird ein Drucker geschlossen, werden alle Optionen auf ihre Standardwerte zurückgesetzt. Wenn die Definitionen in einer Natural-Umgebung nicht eindeutig sind, wird empfohlen, sie in jedem Modul mittels des DEFINE PRINTER-Statements zu setzen.

Syntax-Element	Beschreibung
FORMS *operand2*	**Formular:** Maximale Länge des Operanden: 8 Bytes. Der Standardwert dieser Klausel kann mit dem Subparameter FORMS des Profilparameters PRINT gesetzt werden.
NAME *operand2*	**Listname:** Maximale Länge des Operanden: 8 Bytes. Der Standardwert dieser Klausel kann mit dem Subparameter NAME des Profilparameters PRINT gesetzt werden.
DISP *operand2*	**Disposition:** Maximale Länge des Operanden: 4 Bytes. Für die DISP-Klausel sind die möglichen Werte für *operand2*: DEL, HOLD, KEEP und LEAV.

Syntax-Element	Beschreibung
	Der Standardwert dieser Klausel kann mit dem Subparameter DISP des Profilparameters PRINT gesetzt werden. Wenn die DISP-Klausel weggelassen wird (oder falsch angegeben wird), gilt standardmäßig DEL.
CLASS *operand2*	**Spool-Klasse:** Maximale Länge des Operanden: 1 Byte. Der Standardwert dieser Klausel kann mit dem Subparameter CLASS des Profilparameters PRINT gesetzt werden.
COPIES *operand3*	**Anzahl zu druckender Exemplare:** *operand3* muss ein Ganzzahlwert sein. Der Standardwert dieser Klausel kann mit dem Subparameter COPIES des Profilparameters PRINT gesetzt werden. **Anmerkung:** Die Angabe in der COPIES-Klausel entspricht bei Natural Advanced Facilities der Angabe beim Attribut Duplicates. Gemeint ist in beiden Fällen die Anzahl der zu druckenden Exemplare.
PRTY *operand4*	**Listing-Priorität:** Mögliche Werte: 1 - 255. *operand4* muss ein Ganzzahlwert sein. Der Standardwert dieser Klausel kann mit dem Subparameter PRTY des Profilparameters PRINT gesetzt werden.

Druckername unter z/OS Batch, TSO und Server

Dieser Abschnitt behandelt folgende Themen:

- Logische Dataset-Namen
- Physische Dataset-Namen
- HFS-Datei
- JES-Spoolfile-Klasse
- NULLFILE
- Zuweisung und Freigabe von Datasets
- Druckdateien in Server-Umgebungen

Für eine mit der Zugriffsmethode AM=STD definierte Druckernummer können Sie *operand1* benutzen, um einen logischen oder physischen Dataset-Namen anzugeben, der dieser Druckernummer zugewiesen werden soll.

In diesem Fall kann *operand1* 1 bis 253 Stellen lang sein und folgenden Wert annehmen:

- ein logischer Dataset-Name (DD-Name, 1 bis 8 Stellen);
- ein physischer Dataset-Name eines katalogisierten Datasets (1 bis 44 Stellen) oder ein physischer Dataset-Member-Name (1 bis 44 Stellen für den Dataset-Namen plus 1 bis 8 Stellen in Klammern für den Member-Namen);
- ein Pfad- und Member-Name einer HFS-Datei (1 bis 253 Stellen) in einer MVS-UNIX-Services-Umgebung;
- eine JES-Spoolfile-Klasse
- `NULLFILE` (bezeichnet ein Dummy-Dataset).

Logische Dataset-Namen

Beispiel:

```
DEFINE PRINTER (21) OUTPUT 'SYSPRINT'
```

Das angegebene Dataset mit dem DD-Namen `SYSPRINT` muss zugewiesen worden sein, bevor das `DEFINE PRINTER`-Statement ausgeführt wird. Weitere Informationen entnehmen Sie dem Abschnitt *Zuweisung und Freigabe von Datasets* weiter unten.

Die Zuweisung kann über JCL, CLIST (TSO) oder dynamische Zuweisung (SVC 99) erfolgen. Für dynamische Zuweisung können Sie die Programmier-Schnittstelle (API) `USR2021N` in der Library `SYSEXT` verwenden.

Der in dem `DEFINE PRINTER`-Statement angegebene Dataset-Name überschreibt den in dem Subparameter `DEST` des Profilparameters `PRINT` angegebenen Namen.

Optional kann dem Dataset-Namen `DDN=` vorangestellt werden, um anzuzeigen, dass es sich um einen DD-Namen handelt, und um Namenskonflikte mit zusätzlichen Reports zu vermeiden. Zum Beispiel:

```
DEFINE PRINTER (22) OUTPUT 'DDN=SOURCE'
```

Physische Dataset-Namen

Beispiel:

```
DEFINE PRINTER (23) OUTPUT 'TEST.PRINT.FILE'
```

Das angegebene Dataset muss in katalogisierter Form vorhanden sein. Wenn das `DEFINE PRINTER`-Statement ausgeführt wird, wird das Dataset dynamisch über SVC 99 mit dem aktuellen DD-Namen und der JCL-Option `DISP=SHR` zugewiesen. Weitere Informationen entnehmen Sie dem Abschnitt *Zuweisung und Freigabe von Datasets* weiter unten.

Wenn der Dataset-Name 8 Stellen hat oder kürzer ist und keinen Punkt (.) enthält, könnte er fälschlich als DD-Name interpretiert werden. Um dies zu vermeiden, stellen Sie ihm `DSN=` voran. Zum Beispiel:

```
DEFINE PRINTER (22) OUTPUT 'DSN=PRINTXYZ'
```

Falls das Dataset ein PDS-Member ist, geben Sie den PDS-Member-Namen (1 bis 8 Stellen) in Klammern hinter dem Dataset-Namen (1 bis 44 Stellen) an. Zum Beispiel:

```
DEFINE PRINTER (4) OUTPUT 'TEST.PRINT.PDS(TEST1)'
```

Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

HFS-Datei

Beispiel:

```
DEFINE PRINTER (14) OUTPUT '/u/nat/rec/test.txt'
```

Der angegebene Pfadname muss existieren. Wenn das `DEFINE PRINTER`-Statement ausgeführt wird, wird die HFS-Datei dynamisch zugewiesen. Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

Bei dynamischer Zuweisung des Datasets werden folgende z/OS-Pfadoptionen verwendet:

```
PATHOPTS=(OCREAT,OTRUNC,ORDWR)
PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
FILEDATA=TEXT
```

Wird eine HFS-Datei geschlossen, wird sie automatisch von z/OS freigegeben (unabhängig vom Wert des Subparameters `FREE` im Profilparameter `PRINT`).

JES-Spoolfile-Klasse

Um ein JES-Spool-Dataset zu erzeugen, geben Sie `SYSOUT=`*x* an (wobei *x* die gewünschte Spoolfile-Klasse ist). Für die Standard-Spoolfile-Klasse geben Sie `SYSOUT=*` an.

Beispiele:

```
DEFINE PRINTER (10) OUTPUT 'SYSOUT=A'
DEFINE PRINTER (12) OUTPUT 'SYSOUT=*'
```

Um zusätzliche Parameter für die dynamische Zuweisung anzugeben, verwenden Sie statt des `DEFINE PRINTER`-Statements die Programmierschnittstelle (API) USR2021N in der Library `SYSEXT`.

NULLFILE

Um ein Dummy-Dataset zuzuweisen, geben Sie `NULLFILE` als *operand1* an:

```
DEFINE PRINTER (n) OUTPUT 'NULLFILE'
```

Dies entspricht der JCL-Definition:

```
// DD-name DD DUMMY
```

Zuweisung und Freigabe von Datasets

Wenn das `DEFINE PRINTER`-Statement ausgeführt wird und ein physischer Dataset-Name, eine HFS-Datei, eine Spoolfile-Klasse oder ein Dummy-Dataset angegeben wurde, wird das entsprechende Dataset dynamisch zugewiesen. Wenn eine logische Druckdatei bereits geöffnet ist, wird sie automatisch geschlossen, außer wenn der Subparameter `CLOSE=FIN` des Profilparameters `PRINT` angegeben wurde, wobei dann ein Fehler ausgegeben wird. Außerdem wird ein bestehendes Dataset mit dem gleichen aktuellen DD-Namen automatisch freigegeben, bevor das neue Dataset zugewiesen wird.

Um Fehler durch verfrühtes Öffnen von beim Programmstart noch nicht zugewiesenen Druckdateien zu vermeiden, sollten Druckdateien mit dem Subparameter `OPEN=ACC` (Öffnen bei erstem Zugriff) im Profilparameter `PRINT` definiert werden.

Im Falle einer HFS-Datei oder einer im `PRINT`-Profilparameter mit Subparameter `FREE=ON` definierten Druckdatei wird die Druckdatei automatisch freigegeben, sobald sie geschlossen worden ist.

Als Alternative für die dynamische Zuweisung und Freigabe von Datasets steht Ihnen die Programmierschnittstelle (API) `USR2021N` in der Library `SYSEXT` zur Verfügung. Diese API ermöglicht auch die Angabe zusätzlicher Parameter für die dynamische Zuweisung.

Druckdateien in Server-Umgebungen

In Server-Umgebungen kann es zu Fehlern kommen, wenn mehrere Natural-Sessions versuchen, ein Dataset mit dem gleichen DD-Namen zuzuweisen oder zu öffnen. Um dies zu vermeiden, geben Sie entweder im Profilparameter `PRINT` den Subparameter `DEST=*` an, oder Sie geben im `DEFINE PRINTER`-Statement `OUTPUT '*'` an; Natural generiert dann einen eindeutigen DD-Namen bei der Zuweisung der physischen Datasets, wenn das erste `DEFINE PRINTER`-Statement für die betreffende Druckdatei ausgeführt wird.

Alle Druckdateien, deren DD-Namen mit `CM` anfangen, werden von allen Sessions in einer Server-Umgebung gemeinsam benutzt. Eine solche Druckdatei wird von der ersten Session geöffnet, aber erst bei Beendigung des Servers physisch geschlossen. Weitere Informationen siehe Abschnitt *Natural as a Server* in der *Operations*-Dokumentation

Druckername unter CICS

Für eine mit der Zugriffsmethode `AM=CICS` definierte Druckernummer kann *operand1* in Abhängigkeit vom Subparameter `TYPE` im Profilparameter `PRINT` für den Drucker ein Übergangsdaten- oder Zwischenspeicher-Warteschlangen-Name (1 bis 8 Zeichen) sein. Für `TYPE=TD` (Übergangsdaten) werden nur die ersten 4 Zeichen von *operand1* berücksichtigt und die Destination (Ausgabemedium) für die Übergangsdaten muss vorher für CICS definiert worden sein.

Weitere Informationen entnehmen Sie auch dem Abschnitt *Natural Print and Work Files under CICS* im Natural CICS Interface-Teil der *TP Monitor Interfaces*-Dokumentation.

Druckername unter Com-plete

Wird `AM=COMP` gesetzt, kann eine gültige Druckernummer (TID) oder ein logischer Druckername zugewiesen werden. Beispiel:

```
DEFINE PRINTER (1) OUTPUT '11'
DEFINE PRINTER (2) OUTPUT 'P102'
```

Druckername unter Com-plete/SMARTS

Wird `AM=SMARTS` gesetzt, können Sie einen nicht definierten Druckernamen angeben. Zum Beispiel:

```
DEFINE PRINTER (14) OUTPUT '/nat/path/printer'
DEFINE PRINTER (14) OUTPUT '/nat/path/printer/file/'
DEFINE PRINTER (14) OUTPUT 'printer'
```

Es ist vom Parameter `MOUNT_FS` von SMARTS abhängig, ob die Datei auf einem SMARTS Portable File System oder auf dem Native File System residiert. Das erste Element des Pfades (`/nat/`) legt das Zieldateien-System fest.

Wenn die Zeichenkette mit einem Schrägstrich (/) abgeschlossen wird, wird das letzte Element als Namen der Druckdatei benutzt. Sonst wird der Name der Datei aus der Benutzer-ID und einer Reihenfolge-Nummer generiert. Wenn die Zeichenkette nicht mit einem Schrägstrich anfängt, wird der Pfad der Datei aus der Umgebungsvariable `$NAT_PRINT_ROOT` genommen.

Der angegebene Pfadname muss vorhanden sein. Wenn das Statement `DEFINE PRINTER` ausgeführt wird, wird die Datei dynamisch zugewiesen. Wenn das angegebene Member nicht vorhanden ist, wird ein neues Member dieses Namens erstellt.

Druckernamen unter Natural Advanced Facilities

Benutzer von Natural Advanced Facilities können den logischen Namen jedes vordefinierten logischen Druckerprofils angeben. Dieses logische Druckerprofil muss nicht zu dem gerade aktiven Benutzerprofil gehören; es darf jedes in der `NATSPOOL`-Datei definierte logische Druckerprofil sein. Dieses Profil gilt nur während der Ausführung des Programms, das das `DEFINE PRINTER`-Statement enthält.

Weitere Informationen siehe *Natural Advanced Facilities*-Dokumentation.

Druckername für zusätzliche Reports und Remote-Destinationen (Ausgabeziele)

Mit den folgenden Namen können Sie standardmäßig zusätzliche Reports und Remote-Destinationen (Ausgabeziele) zuweisen:

Report	Funktion
BROADCST	Ausgabe der Meldungszeile an einen TP-Monitor-Terminal. Gleiche Funktion wie MESSAGE (siehe unten), außer dass unter Com-plete die Meldung nicht an den gewünschten Terminal geschickt wird, bis keine Transaktionen auf diesem Terminal mehr aktiv sind.
CCONTROL	CCONTROL ist der Name einer bestimmten Druckersteuerzeichen-Tabelle, die mit dem Drucker *n*-1 in Verbindung steht; sie darf nicht geändert werden. Weitere Informationen entnehmen Sie dem Abschnitt *Printer-Advance Control Characters* in der *Operations*-Dokumentation.
CONNECT	Ausgabe in ein Con-nect-Fach. Anmerkung für die Natural-Installation: das NATPCNT-Modul von Natural muss mit dem Natural-Nukleus verbunden werden.
DUMMY	Ausgabe wird gelöscht.
HARDCOPY	Ausgabe an das aktuelle Hardcopy-Gerät.
INCORE	Ausgabe an die NSPF Incore-Datenbank.
INFOLINE	Ausgabe in der Natural-Infoline. Näheres zur Infoline siehe Terminalkommando %X in der *Terminalkommandos*-Dokumentation.
MESSAGE	Ausgabe der Meldungszeile an einen TP-Monitor-Terminal. Die ersten 8 Bytes einer Meldung müssen die Ziel-Terminal-ID enthalten. Für TSO ist die User-ID anstatt der Terminal-ID erforderlich. Ein Beispiel-Programm mit Namen MSGSW ist in der Library SYSEXTP vorhanden.
SOURCE	Ausgabe in den Arbeitsbereich des Natural-Editors.
WORKPOOL	Ausgabe in den Natural-ISPF-Workpool.

Daten in einer Remote-JES-Umgebung drucken

Sie können das Write-to-Spool Feature (siehe *Operations*-Dokumentation) benutzen, um Daten über einen Remote-JES-Knoten zu leiten und sie an einen Benutzer zu senden oder um sie auf einem Gerät zu drucken, das in der Remote-JES-Umgebung definiert ist.

Beispiele DEFINE PRINTER

- Beispiel 1 - Definition des Druckernamens für Com-plete
- Beispiel 2 - Definition des Druckernamens für Batch-Umgebung
- Beispiel 3 - Druckausgabe an Infoline

- Beispiel 4 - Benutzung einer Session mit vordefiniertem Drucker

Beispiel 1 - Definition des Druckernamens für Com-plete

```
/* PRINTER NAME DEFINITION FOR COM-PLETE
*
DEFINE PRINTER (1) OUTPUT 'TID100'
WRITE (1) 'PRINTED ON PRINTER TID100'
END
```

Beispiel 2 - Definition des Druckernamens für Batch-Umgebung

```
/* OUTPUT ON 'SYSPRINT' (FOR BATCH ENVIRONMENTS)
*
DEFINE PRINTER (REPORT1 = 1) OUTPUT 'SYSPRINT'
WRITE (REPORT1) 'REPORT 1 PRINTED ON PRINTER SYSPRINT'
*
/* OUTPUT TO DEFAULT PRINTER DESTINATION
/* DEFINED WITH PROFILE PARAMETER 'PRINT', SUBPARAMETER 'DEST'
*
DEFINE PRINTER (REPORT2 = 2)
WRITE (REPORT2) 'REPORT PRINTED TO DESTINATION'
```

Beispiel 3 - 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   ↩

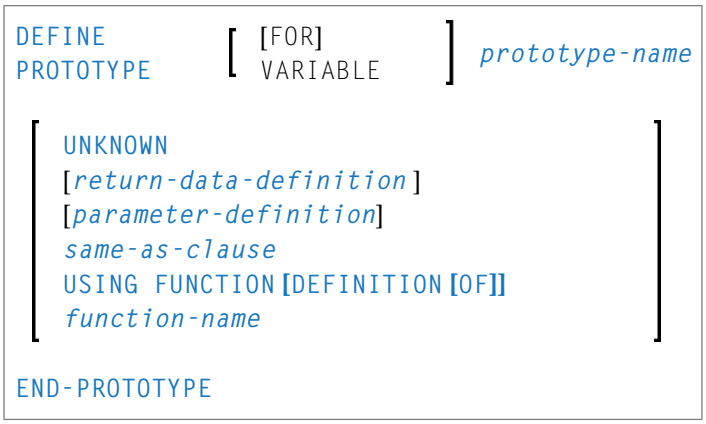
TEST OUTPUT
```

Beispiel 4 - Benutzung einer Session mit vordefiniertem Drucker

```
** Example 'DPREX1': DEFINE PRINTER
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
END-DEFINE
*
*  USE SESSION WITH DEFINED PRINTER 1
*
DEFINE PRINTER (INVOICE-LIST=1) OUTPUT 'OUTQ1'
LIMIT 5
READ EMPL-VIEW BY NAME
  WRITE (INVOICE-LIST) NAME
END-READ
*
END
```

48 DEFINE PROTOTYPE

- Funktion DEFINE PROTOTYPE 330
- Syntax-Beschreibung DEFINE PROTOTYPE 331
- Beispiele DEFINE PROTOTYPE 334



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

Verwandtes Statement: `DEFINE FUNCTION`

Funktion DEFINE PROTOTYPE

Das `DEFINE PROTOTYPE`-Statement dient dazu, die Eigenschaften für den Aufruf einer Function anzugeben:

- die Parameter, die an den Function Call übergeben werden sollen,
- der Ergebniswert, die vom Function Call zurückgegeben werden soll, und
- wie die Function aufgerufen wird: entweder mit dem Function-Namen, der im `DEFINE FUNCTION`-Statement definiert ist, oder mit einer alphanumerischen Variablen, die den Function-Namen enthält.

Diese Informationen dienen dazu, einen Function Call innerhalb eines Natural-Objekts zur Kompilierungszeit aufzulösen.

Ein `DEFINE PROTOTYPE`-Statement wird für einen Function Call nur dann benötigt, wenn Folgendes zutrifft:

- Der angegebene Function-Name ist eine alphanumerische Variable, die den Namen der zur Ausführungszeit aufzurufenden Function enthält.
- In dem Function Call ist keine `(IR=)`-Option angegeben, und ein katalogisiertes Objekt der aufgerufenen Funktion ist nicht verfügbar,
- Die im Function Call mitgegebenen Parameter sollen validiert werden, und das katalogisierte Objekt der aufgerufenen Funktion ist nicht verfügbar.

Wenn die Function von mehreren Objekten aufgerufen werden soll kann das `DEFINE PROTOTYPE`-Statement in ein Copycode-Objekt eingefügt werden.

Weitere Information finden Sie in folgenden Abschnitte im *Leitfaden zur Programmierung*:

- Natural-Objektyp Function
- *Function Call*

Syntax-Beschreibung DEFINE PROTOTYPE

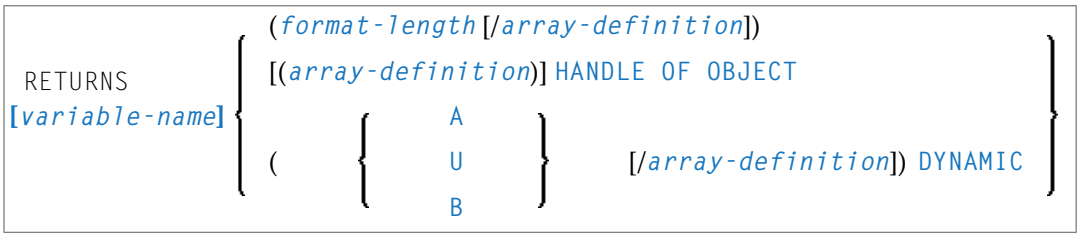
Syntax-Element	Beschreibung
`[VARIABLE]` *`prototype-name`*	**Prototyp-Name:** *`prototype-name`* ist Folgendes: ■ entweder der Name des Prototyps, dessen Parameter- und Ergebnisfeld-Definitionen verwendet werden sollen. Dieser Name stimmt in der Regel mit dem *`function-name`* im `DEFINE FUNCTION`-Statement der referenzierten Function überein; ■ oder der Name eines alphanumerischen Feldes, der als *`function-name`* in einem Function Call angegeben wird, wenn das Schlüsselwort `VARIABLE` angegeben ist. Dieses Feld muss den Namen der Function enthalten, die zur Ausführungszeit aufgerufen werden soll. Mit dem Feldnamen darf kein Array-Index-Ausdruck angegeben werden.
`UNKNOWN`	**UNKNOWN-Option:** Das Schlüsselwort `UNKNOWN` gibt an, dass das Function-Interface zurzeit nicht definiert ist. In diesem Fall wird das katalogisierte Objekt (falls verfügbar) nicht benutzt, um das Layout des Function-Ergebnisses und die Parameter-Beschreibung zu extrahieren. Wenn ein Function Call in ein Natural-Statement eingebettet ist, muss dass Ergebnis-Layout explizit mit einer `(IR=)`-Klausel angegeben werden. Außerdem werden in der Function enthaltene Parameter nicht geprüft.
`return-data-definition`	Siehe ***Rückgabedatendefinition*** weiter unten.
`parameter-definition`	Siehe ***Parameter-Definition*** weiter unten.
`same-as-clause`	Siehe ***SAME AS-Klausel*** weiter unten.

`USING FUNCTION`
`[DEFINITION [OF]]`
`function-name`
	Siehe ***USING FUNCTION-Klausel*** weiter unten.

`END-PROTOTYPE`	**Ende des DEFINE PROTOTYPE-Statement:** Das für Natural reservierte Wort `END-PROTOTYPE` muss zum Beenden des `DEFINE PROTOTYPE`-Statements benutzt werden.

Rückgabedatendefinition

(*return-data-definition*)



Mit dieser Klausel werden das Format und die Länge und, falls zutreffend, die Array-Struktur des Rückgabewerts festgelegt.

Wenn keine Festlegung der Rückgabedaten erfolgt, kann ein Function Call nur innerhalb eines Statements benutzt werden, wenn eine explizite (IR=)-Klausel verfügbar ist. Fehlt eine solche Klausel, kann die Function nur als ein Statement aufgerufen werden, aber nicht anstelle eines Operanden in einem Statement.

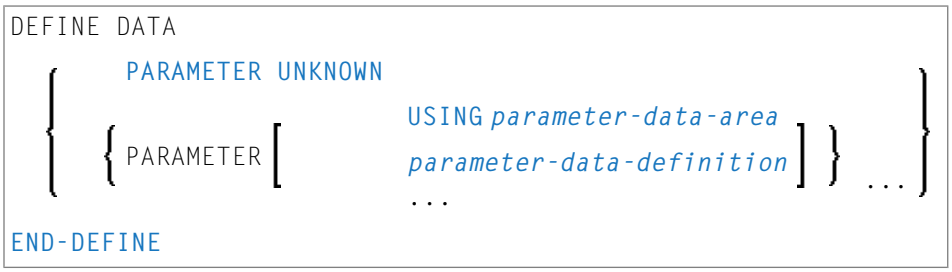
Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
variable-name	**Name des Rückgabewerts:** Die optionale Angabe bei *variable-name* hat hier keine Bedeutung. Sie ist nur vorhanden, um eine ähnliche Syntaxstruktur wie bei der ***Rückgabedatendefinition***-Klausel des DEFINE FUNCTION-Statements zu haben.
format-length	**Format-/Länge-Definition:** Format und Länge des Ergebnisfeldes. Weitere Informationen siehe *Format und Länge von Benutzervariablen* im *Leitfaden zur Programmierung*.
array-definition	**Definition der Array-Dimensionen:** Falls das Ergebnisfeld der Function ein Array-Feld ist, legen Sie hier die untere und obere Grenze einer Dimension fest. Weitere Informationen siehe DEFINE DATA-Statement, ***Definition von Array-Dimensionen***.
HANDLE OF OBJECT	**Object-Handle:** Wird bei NaturalX verwendet. Weitere Informationen siehe *NaturalX* im *Leitfaden zur Programmierung*.
A, U or B	**Datentyp:** Alphanumerisch (A), Unicode (U) oder binär (B) für ein numerisches Ergebnis.

Syntax-Element	Beschreibung
DYNAMIC	**Dynamische Variable:** Das Function-Ergebnis kann als DYNAMIC definiert werden. Weitere Informationen siehe *Dynamische Variablen* im *Leitfaden zur Programmierung*.

Parameter-Definition

(*parameter-definition*)



Mit dieser Klausel werden die Parameter festgelegt, die bei einem Function Call zur Verfügung gestellt werden. Das hier festgelegte Layout wird gegen die Parameter geprüft, welche in einem Function Call mitgegeben werden. Wird diese Klausel weggelassen, wird die Function als parameterfrei deklariert. In diesem Fall wird jeder Versuch, Parameter im Function Call mitzugeben, zurückgewiesen.

Die für die Namensgebung der Parameterfelder verwendeten Bezeichner haben keine Bedeutung. Sie sind hier nur vorhanden, um eine ähnliche Syntaxstruktur wie bei der DEFINE DATA PARAMETER-Syntax zu haben.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
PARAMETER UNKNOWN	**UNKNOWN-Option:** Wird diese Option benutzt, dann wird kein Parameter angegeben. Dadurch wird die Parameterprüfung im Function Call ausgeschaltet. Das hat zur Folge, dass eine beliebige Anzahl von Parametern in der Funktion akzeptiert wird.
USING *parameter-data-area*	**PDA-Name:** Name der Parameter Data Area, welche Datenelemente enthält, die als Parameter in einem Function Call verwendet werden. Siehe auch ***Definition von Parameter Data*** in der DEFINE DATA-Statement-Beschreibung.
parameter-data-definition	**Parameterdatendefinition:**

Syntax-Element	Beschreibung
	Anstatt eine Parameter Data Area zu definieren, können Parameter auch direkt in einem Function Call definiert werden. Siehe auch ***Direkte Definition von Parameterdaten*** in der DEFINE DATA-Statement-Beschreibung.
END-DEFINE	**Ende der Klausel:** Das für Natural reservierte Wort END-DEFINE muss zum Beenden der *parameter-definition*-Klausel benutzt werden.

SAME AS-Klausel

(*same-as-clause*)

```
SAME AS [PROTOTYPE] prototype-name
```

Durch Angabe dieser Klausel können die Parameter- und Ergebnisfeld-Definitionen eines anderes Prototyps verwendet werden, welcher zuvor im selben Natural-Objekt definiert worden ist.

Beispiele DEFINE PROTOTYPE

- Beispiel 1 - DEFINE PROTOTYPE mit einem definierten Function-Namen
- Beispiel 2 - DEFINE PROTOTYPE mit einem variablen Function-Namen

Beispiel 1 - DEFINE PROTOTYPE mit einem definierten Function-Namen

Dies ist eine Prototyp-Definition einer Funktion mit dem Namen F#FACTOR, wobei der *prototype-name* dem *function-name* entspricht, der im referenzierten DEFINE FUNCTION-Statement angegeben ist. Das von der Function zurückgelieferte Ergebnis hat das Format (I2/1:3), und es ist nur ein einzelner Parameter nötig, welcher das Format (I2) hat.

```
** Example 'DPTEX1': DEFINE PROTOTYPE and function call
************************************************************************
DEFINE DATA LOCAL
  1 #NUM (I2)
END-DEFINE
*
DEFINE PROTOTYPE F#FACTOR
  RETURNS (I2/1:3)
  DEFINE DATA PARAMETER
    1 #VALUE (I2)
  END-DEFINE
END-PROTOTYPE
*
```

```
#NUM := 3
*
WRITE 'Function call:' F#FACTOR(<#NUM>)(*)
*
END
```

Die Function F#FACTOR ist definiert in der Beispiel-Function DFUEX2 in der Library SYSEXSYN. Siehe *Beispiele* in der DEFINE FUNCTION-Statement-Beschreibung.

Ausgabe des Programms DPTEX1:

```
Function call:      3      6      9
```

Beispiel 2 - DEFINE PROTOTYPE mit einem variablen Function-Namen

Wegen des Schlüsselworts VARIABLE gibt dieser Prototyp einen Function Call, bei dem der referenzierte Prototyp-Name eine alphanumerische Variable ist, die den Namen der Function zum Zeitpunkt der Ausführung enthält.

```
** Example 'DPTEX2': DEFINE PROTOTYPE and function call
************************************************************************
DEFINE DATA LOCAL
  1 #NAME (A20)
  1 #TEXT (A10)                                                          ↩

END-DEFINE
*
DEFINE PROTOTYPE VARIABLE #NAME
  RETURNS #RETURN (A1)
  DEFINE DATA PARAMETER
    1 #IN (A10)
  END-DEFINE
END-PROTOTYPE
*
#NAME := 'F#FIRST-CHAR'
#TEXT := 'ABCDEFGHIJ'
*
WRITE 'First character:' #NAME(<#TEXT>)
*
END
```

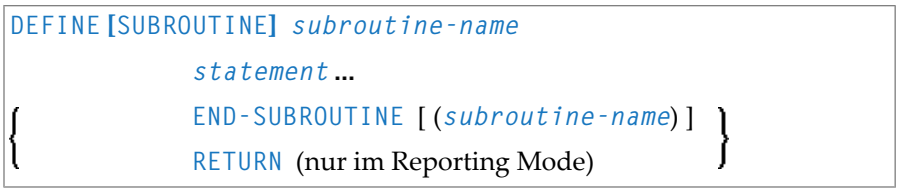
Die Funktion F#FIRST-CHAR ist definiert in der Beispiel-Function DFUEX1 in der Library SYSEXSYN. Siehe *Beispiele* in der DEFINE FUNCTION-Statement-Beschreibung.

Ausgabe des Programms `DPTEX2`**:**

```
First character: A ↩
```

49 DEFINE SUBROUTINE

- Funktion DEFINE SUBROUTINE 338
- Syntax-Beschreibung DEFINE SUBROUTINE 339
- Einschränkungen DEFINE SUBROUTINE 340
- Beispiele DEFINE SUBROUTINE 341



Dieser Abschnitt 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 Subprogrammen***

Funktion DEFINE SUBROUTINE

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.

Daten, die einer Subroutine zur Verfügung stehen

- 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 auf den aktuell eingerichteten globalen Datenbereich (Global Data Area) sowie den vom aufrufenden Programm verwendeten lokalen Datenbereich (Local Data Area) zugreifen.

Externe Subroutinen

Eine externe Subroutine hat Zugriff auf den aktuell eingerichteten globalen Datenbereich (Global Data Area). Außerdem können Parameter direkt mit dem PERFORM-Statement vom aufrufenden Objekt an die externe Subroutine übergeben werden. Dadurch kann die Größe des globalen Datenbereichs gering gehalten werden.

Eine externe Subroutine kann nicht auf den im aufrufenden Programm definierten lokalen Datenbereich (Local Data Area), allerdings kann eine externe Subroutine einen eigenen lokalen Datenbereich haben.

Syntax-Beschreibung DEFINE SUBROUTINE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
subroutine-name	**Name der Subroutine:** Für den Namen einer Subroutine (maximal 32 Zeichen lang) gelten die gleichen Namenskonventionen wie für Benutzervariablen. Siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*. 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	**Ende des DEFINE SUBROUTINE-Statements:** Im Structured Mode muss die Definition einer Subroutine mit dem Schlüsselwort END-SUBROUTINE beendet werden. Im Reporting Mode kann die Definition einer Subroutine mit dem Schlüsselwort RETURN oder mit END-SUBROUTINE beendet werden.
RETURN oder END-SUBROUTINE	
(subroutine-name)	**End-Subroutine-Label:**

Syntax-Element	Beschreibung
	Das End-Subroutine-Label verschafft Ihnen einen Überblick, welches END-SUBROUTINE-Schlüsselwort zu welcher Subroutine gehört. Der Name der Subroutine muss in Klammern gesetzt werden und darf erst nach dem END-SUBROUTINE-Schlüsselwort stehen. Das End-Subroutine-Label muss mit dem Namen der Subroutine übereinstimmen, sonst gibt es einen Compilerfehler.

Einschränkungen DEFINE SUBROUTINE

- 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).
- Der Name einer externen Subroutine darf in einer Bibliothek nur einmal verwendet werden.
- Das End-Subroutine-Label muss denselben Subroutinen-Namen erhalten, der mit DEFINE SUBROUTINE definiert wurde. Das End-Subroutine-Label kann nur mit END-SUBROUTINE definiert werden (siehe *Beispiel 3* 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
```

Beispiel 3 (ungültig):

```
...
DEFINE SUBROUTINE SUBR01
  ...
END-SUBROUTINE (SUBROUTINE-01)
...
DEFINE SUBROUTINE SUBR02
  ...
RETURN (SUBR02)
...
```

Beispiel 4:

```
...
DEFINE SUBROUTINE SUBR01
  ...
END-SUBROUTINE (SUBR01)                  /* subroutine label for SUBR01
DEFINE SUBROUTINE SUBR02
  ...
END-SUBROUTINE /* (SUBR02)               /* comment only
DEFINE SUBROUTINE SUBR03
  ...
END-SUBROUTINE /* Subroutine SUBR03      /* comment only
```

Beispiele DEFINE SUBROUTINE

- Beispiel 1 - Subroutine definieren
- Beispiel 2 - Beispiel-Struktur für externe Subroutine mittels GDA-Feldern

- Beispiel 3 - Subroutinen mit End-Subroutine-Label

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.
                          MILWAUKEE                 MONTERREY
554349                    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
```

Beispiel 3 - Subroutinen mit End-Subroutine-Label

```
** Example 'DSREX3': DEFINE SUBROUTINE with end-subroutine label
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 ADDRESS-LINE  (A20/2)
  2 PERSONNEL-ID
  2 MAR-STAT
  2 BIRTH
*
1 #ARRAY          (A75/1:6)
1 REDEFINE #ARRAY
  2 #ALINE        (A25/1:6,1:3)
1 #BIRTHDAY       (A25)
1 #FULL-NAME      (A25)
1 #MARITAL-STATUS (A25)
1 #X              (N2) INIT <1>
1 #Y              (N2) INIT <1>
END-DEFINE
*
FORMAT PS=20
LIMIT 5
*
FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  /*
  PERFORM GET-EMPLOYEES-DATA
  /*
  IF  #Y = 3
    RESET INITIAL #Y
    PERFORM PRINT-ARRAY
  ELSE
    ADD 1 TO #Y
  END-IF
  /*
  AT END OF DATA
    PERFORM PRINT-ARRAY
  END-ENDDATA
END-FIND
/*----------------------------------------------------------------------
/* ------------ End of main program -----------------------------------
/*----------------------------------------------------------------------
DEFINE SUBROUTINE GET-EMPLOYEES-DATA
  COMPRESS NAME ',' FIRST-NAME INTO #FULL-NAME LEAVING NO
  MOVE EDITED BIRTH (EM=DD.LLL.YYYY) TO #BIRTHDAY
  /*
  PERFORM MARITAL-STATUS
  /*
  MOVE #FULL-NAME      TO #ALINE (#X,#Y)
```

```
  MOVE PERSONNEL-ID    TO #ALINE (#X+1,#Y)
  MOVE #MARITAL-STATUS TO #ALINE (#X+2,#Y)
  MOVE #BIRTHDAY       TO #ALINE (#X+3,#Y)
  MOVE ADDRESS-LINE(1) TO #ALINE (#X+4,#Y)
  MOVE ADDRESS-LINE(2) TO #ALINE (#X+5,#Y)
END-SUBROUTINE (GET-EMPLOYEES-DATA)         /* end-subroutine label
*
DEFINE SUBROUTINE MARITAL-STATUS
  DECIDE ON FIRST VALUE OF MAR-STAT
    VALUE 'M'
      #MARITAL-STATUS := 'MARRIED'
    VALUE 'S'
      #MARITAL-STATUS := 'SINGLE'
    VALUE 'D'
      #MARITAL-STATUS := 'DIVORCED'
    VALUE 'W'
      #MARITAL-STATUS := 'WIDOWED'
    NONE
      RESET #MARITAL-STATUS
  END-DECIDE
END-SUBROUTINE /* (MARITAL-STATUS)          /* just a comment
*
DEFINE SUBROUTINE PRINT-ARRAY
  WRITE NOTITLE (AD=OI) #ARRAY(*)
  RESET #ARRAY(*)
  SKIP 1
END-SUBROUTINE (PRINT-ARRAY)                /* end-subroutine label
*
END
```

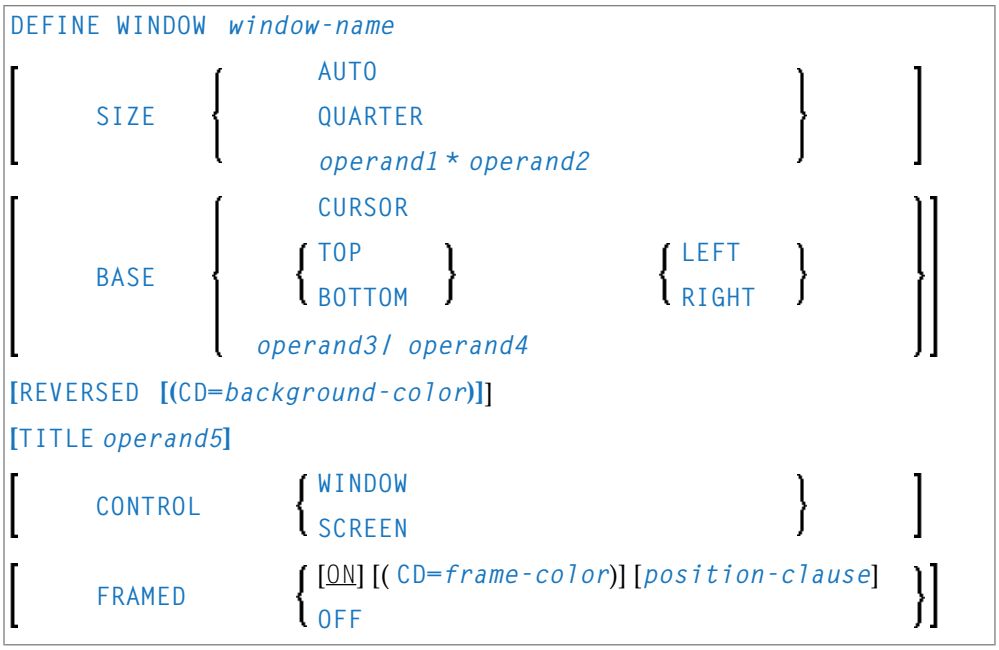
Ausgabe des Programms DSREX3:

```
SMITH,GERHARD            SMITH,SEYMOUR            SMITH,MATILDA
40000311                 20009300                 20014100
DIVORCED                 SINGLE                   MARRIED
21.Apr.1976              01.Jan.1970              01.Jan.1970
ENGLANDSVEJ 222          3152 SHETLAND ROAD       14100 ESWORTHY RD.
                         MILWAUKEE                MONTERREY

SMITH,ANN                SMITH,TONI
20015400                 20018800
WIDOWED                  SINGLE
07.Nov.1946              14.Dec.1941
5 HAWTHORN               13002 NEW ARDEN COUR
OAK BROOK                SILVER SPRING
```

50 DEFINE WINDOW

- Funktion DEFINE WINDOW 348
- Syntax-Beschreibung DEFINE WINDOW 349
- Schutz von Eingabefeldern in einem Fenster 353
- Aufrufen unterschiedlicher Fenster 353
- Beispiel DEFINE WINDOW 354



Dieser Abschnitt 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 DEFINE WINDOW

Das `DEFINE WINDOW`-Statement dient dazu, 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.

Siehe auch den Abschnitt *Bildschirmgestaltung/Fenster* im Kapitel *Gestaltung von Benutzeroberflächen von Anwendungen* im *Leitfaden zur Programmierung*.

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 DEFINE WINDOW

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I							ja	nein
operand2	C	S						N	P	I							ja	nein
operand3	C	S						N	P	I							ja	nein
operand4	C	S						N	P	I							ja	nein
operand5	C	S				A	U										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
window-name	Der *window-name* identifiziert das Fenster. Der Name darf bis zu 32 Stellen lang sein. Für Fensternamen gelten die gleichen Namenskonventionen wie für Benutzervariablen. Siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.
SIZE	Mit der SIZE-Klausel bestimmen Sie die Größe des Fensters. **Anmerkung:** Auf z/OS-Computern 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 z/OS-Computer der von einem Fenster überlagerte Bildschirmbereich breiter und der innerhalb eines Fensters sichtbare Seitenausschnitt schmäler als auf anderen Plattformen. Beispiel: Angenommen, die Größe eines Fensters ist mit SIZE 5 * 15 definiert (d.h. mit einer Breite von 15 Spalten): ■ Auf z/OS-Computern 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.

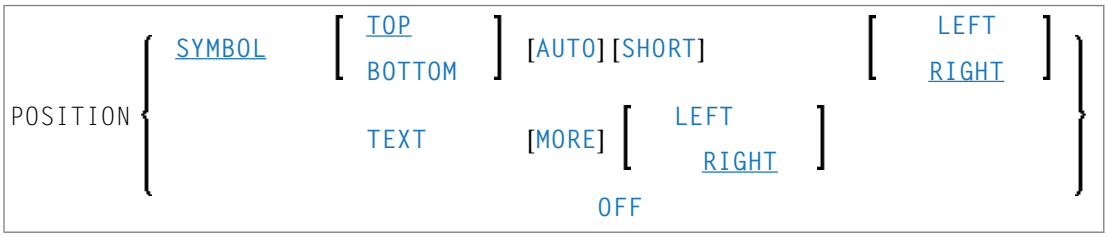
Syntax-Element	Beschreibung
`SIZE AUTO`	Die Größe des Fensters wird von Natural automatisch zur Laufzeit festgelegt. Die Größe wird wie folgt durch die in das Fenster hineingenerierten Daten bestimmt: ■ Die Zeilenanzahl des Fensters entspricht der Anzahl der generierten `INPUT`-Zeilen (plus gegebenenfalls 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: `SIZE` *operand1* * *operand2* Wenn Sie die `SIZE`-Klausel weglassen, gilt standardmäßig `SIZE AUTO`. **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.
`SIZE QUARTER`	Die Größe des Fensters entspricht einem Viertel der physischen Bildschirmgröße.
`SIZE` *operand1* * *operand2*	Die Größe des Fensters beträgt *n* Zeilen mal *n* Spalten. *operand1* bestimmt die Anzahl der Zeilen, *operand2* die Anzahl der Spalten. Die beiden Operanden dürfen keine Dezimalstellen enthalten. Ist das Fenster gerahmt (`FRAMED`), so schließt die angegebene Größe den Rahmen mit ein. Die kleinstmögliche Fenstergröße ist: ■ ohne Rahmen: 2 Zeilen mal 10 Spalten, ■ mit Rahmen: 4 Zeilen mal 13 Spalten. Die größtmögliche Fenstergröße ist die Größe des physischen Bildschirms.
`BASE`	Mit der `BASE`-Klausel bestimmen Sie die Position des Fensters auf dem physischen Bildschirm. Lassen Sie die `BASE`-Klausel weg, gilt standardmäßig `BASE CURSOR`.
`BASE CURSOR`	Platziert die obere linke Ecke des Fensters an die aktuelle Cursor-Position. Die Cursor-Position ist die physische Position des Cursors auf dem Bildschirm. 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.
`BASE TOP/BOTTOM LEFT/RIGHT`	Platziert das Fenster in die obere linke, untere linke, obere rechte bzw. untere rechte Ecke des physischen Bildschirms.
`BASE` *operand3*/*operand4*	Platziert die obere linke Ecke des Fensters in die angegebene Zeile/Spalte auf dem physischen Bildschirm. *operand3* bestimmt die Zeilennummer, *operand4* die Spaltennummer. Die beiden Operanden dürfen keine Dezimalstellen enthalten.

Syntax-Element	Beschreibung
	Ist es aufgrund der Größe des Fensters nicht möglich, das Fenster an die angegebene Position zu platzieren, erhalten Sie eine Fehlermeldung.
`REVERSED`	`REVERSED` bewirkt, dass das Fenster invers angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird `REVERSED` ignoriert).
`REVERSED CD=` *`background-color`*	Dies bewirkt, dass das Fenster invers und der Fensterhintergrund in der angegebenen Farbe (*`background-color`*) angezeigt wird (falls der verwendete Bildschirm dies ermöglicht; falls nicht, wird die betreffende Angabe ignoriert). Informationen über gültige Farbcodes. Session-Parameter `CD` in der *Parameter-Referenz*-Dokumentation.
`TITLE` *`operand5`*	Mit der `TITLE`-Klausel können Sie eine Überschrift für das Fenster angeben. Die angegebene Überschrift (*`operand5`*) 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 *`position-clause`* 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.

Syntax-Element	Beschreibung
FRAMED (CD= *frame-color*)	Dies bewirkt, dass der Fensterrahmen in der angegebenen Farbe (*frame-color*) angezeigt wird (falls ein Farbbildschirm verwendet wird; falls nicht, wird die Farbangabe ignoriert). Informationen über gültige Farbcodes siehe Session-Parameter CD in der *Parameter-Referenz*. **Anmerkung:** Bei Natural for Windows wird diese Angabe ignoriert.
position-clause	Die POSITION-Klausel wird nur auf z/OS-Computern ausgewertet: auf allen anderen Plattformen wird sie ignoriert. Einzelheiten, siehe ***POSITION-Klausel*** weiter unten.

POSITION-Klausel

Die POSITION-Klausel wird nur auf z/OS-Computern 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.

Syntax-Element	Beschreibung
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.

Syntax-Element	Beschreibung
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 Top und Bottom und zeigt nur das Wort More 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 DEFINE WINDOW

```
** 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`:

```
                                      +------------------------More:     + >+
> r                                   ! Page      1                         !
All    ....+....1....+....2....+....3.. !                                   !
  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 DEFINE WORK FILE 358
- Syntax-Beschreibung DEFINE WORK FILE 358
- Arbeitsdateiname unter z/OS Batch, TSO und Server 360
- Arbeitsdateiname unter CICS 363
- Arbeitsdateiname unter Com-plete/SMARTS 364

```
DEFINE WORK FILE                  { operand1 [TYPE operand2] }
work-file-number                  { TYPE operand2            }  [ATTRIBUTES {operand3}...]
```

Anmerkung: Die in eckigen Klammern [...] gezeigten Elemente sind optional, aber mindestens eines muss bei diesem Statement angegeben werden.

Dieser Abschnitt 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 DEFINE WORK FILE

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.

Alle während einer Session zu benutzenden Arbeitsdateien müssen mittels des Subparameters `AM` des Profilparameters `WORK` oder automatisch durch Definition in der JCL im Voraus einer Zugriffsmethode zugewiesen werden.

Anmerkung: Bezüglich Unicode- und Codepage-Support siehe *Arbeitsdateien und Druckdateien* in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Syntax-Beschreibung DEFINE WORK FILE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A	U									yes	no
operand2	C	S				A	U									yes	no
operand3	C	S				A	U									yes	no

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Nummer der Arbeitsdatei:** Gibt die Nummer der Arbeitsdatei an. Die Nummer der Arbeitsdatei ist entweder ■ eine numerische Konstante im Wertebereich 1:32 oder ■ eine mit der Klausel CONSTANT definierte numerische Variable im Format B/N/P/I, die einen Wert im Bereich 1:32 zuweist. Dezimalstellen bei den Formaten (N/P) sind nicht erlaubt. Dies ist die Nummer, die in einem READ WORK FILE-, WRITE WORK FILE- oder CLOSE WORK FILE-Statement verwendet wird.
operand1	**Name der Arbeitsdatei:** *operand1* ist der Name der Arbeitsdatei. Als *operand1* geben Sie den Namen des der Arbeitsdateinummer zuzuweisenden Datasets an. *operand1* kann 1 bis 253 Stellen lang sein. Sie können entweder einen logischen oder einen physischen Dateinamen angeben. Das mögliche Format ist abhängig von der Betriebssystemumgebung und der vom Subparameter AM des Profilparameters WORK definierten Zugriffsmethode. Einige Zugriffsmethoden unterstützen *operand1* nicht als Arbeitsdateinamen, z.B. AM=COMP und AM=PC. Wenn Sie *operand1* weglassen, wird der Wert von *operand1* bestimmt, indem für diese Arbeitsdateinummer der aktuelle Name genommen wird, der bei dem zuvor ausgeführten DEFINE WORK FILE-Statement angegeben wurde. Wenn noch kein DEFINE WORK FILE-Statement ausgeführt worden ist, wird der im Natural-Parametermodul angegebene Name verwendet. **Anmerkung:** Wenn *operand1* nicht angegeben wird, ist das Verhalten bei Natural for z/OS und Natural for Windows oder Linux verschieden. Informationen zu Betriebssystem- oder TP-Monitor-abhängigen Arbeitsdateinamen finden Sie in den folgenden Abschnitten: ■ **Arbeitsdateiname unter z/OS Batch, TSO und Server** ■ **Arbeitsdateiname unter CICS** ■ **Arbeitsdateiname unter Com-plete/SMARTS**

Syntax-Element	Beschreibung	
`TYPE operand2`	**TYPE-Klausel:** *operand2* gibt den Arbeitsdateityp an. Beim Wert von *operand2* wird nicht nach Groß- und Kleinbuchstaben unterschieden. Der Wert muss in Anführungszeichen (' . . . ') stehen oder in einer alphanumerischen Variable angegeben werden.	
	UNFORMATTED	Eine vollkommen unformatierte Datei. Es werden keine Formatier-Informationen geschrieben (weder für Felder noch für Sätze). `UNFORMATTED` behandelt eine Arbeitsdatei als einen Byte-Strom ohne Datensatz-Grenzen. Beachten Sie, dass der Type `UNFORMATTED` von Entire Connection zurückgewiesen wird. Format: `UNFORMATTED`
	FORMATTED	`FORMATTED` definiert eine reguläre datensatz-orientierte Arbeitsdatei, die derselben Verarbeitung unterliegt wie in vorherigen Natural-Versionen.
`[ATTRIBUTES {operand3}...]`	**ATTRIBUTES-Klausel:** Diese Klausel ist nur bei Natural for Windows oder Linux sinnvoll; bei Natural for z/OS wird sie ignoriert.	

Beispiele:

```
DEFINE WORK FILE 17 #FILE TYPE 'UNFORMATTED'
#TYPE := 'FORMATTED'
DEFINE WORK FILE 18 #FILE TYPE #TYPE
```

Arbeitsdateiname unter z/OS Batch, TSO und Server

Folgende Themen werden behandelt:

- Arbeitsdateiname - operand1
- Zuweisung und Freigabe von Datasets

- Arbeitsdateien in Server-Umgebungen

Arbeitsdateiname - operand1

Unter z/OS kann für eine Arbeitsdateinummer, die mit Zugriffsmethode `AM=STD` definiert ist, *`operand1`* folgendes sein:

- ein **logischer Dataset-Name** (DD-Name, 1 bis 8 Stellen);
- ein **physischer Dataset-Name** eines katalogisierten Datasets (1 bis 44 Stellen) oder ein physischer Dataset-Member-Name;
- ein Pfad- und Member-Name einer **HFS-Datei** (1 bis 253 Stellen) in einer MVS-UNIX-Services-Umgebung;
- eine **JES-Spoolfile-Klasse**;
- `NULLFILE`.

Logische Dataset-Namen	Beispiel:
	
	`DEFINE WORK FILE 21 'SYSOUT1'`
	
	Das angegebene Dataset `SYSOUT1` muss zugewiesen worden sein, bevor das `DEFINE WORK FILE`-Statement ausgeführt wird.
	
	Die Zuweisung kann über JCL, CLIST (TCO) oder dynamische Zuweisung (SVC 99) erfolgen. Für dynamische Zuweisung können Sie die Programmierschnittstelle (API) `USR2021N` in der Library `SYSEXT` verwenden.
	
	Der im `DEFINE WORK FILE`-Statement angegebene Dataset-Name überschreibt den im Subparameter `DEST` des Profilparameters `WORK` angegebenen Namen.
	
	Optional kann dem Dataset-Namen `DDN=` vorangestellt werden, um anzuzeigen, dass es sich um einen DD-Namen handelt. Zum Beispiel:
	
	`DEFINE WORK FILE 22 'DDN=MYWORK'`

Physische Dataset-Namen	Beispiel: `DEFINE WORK FILE 23 'TEST.WORK.FILE'` Das angegebene Dataset muss in katalogisierter Form vorhanden sein. Wenn das `DEFINE WORK FILE`-Statement ausgeführt wird, wird das Dataset dynamisch über SVC 99 mit dem aktuellen DD-Namen und der Option `DISP=SHR` zugewiesen. Wenn der Dataset-Name 8 Stellen oder kürzer ist und keinen Punkt (.) enthält, könnte er fälschlich als DD-Name interpretiert werden. Um dies zu vermeiden, stellen Sie ihm `DSN=` voran. Zum Beispiel:

	`DEFINE WORK FILE 22 'DSN=WORKXYZ'`
	
	Falls das Dataset ein PDS-Member ist, geben Sie den PDS-Member-Namen (1 bis 8 Stellen) in Klammern hinter dem Dataset-Namen (1 bis 44 Stellen) an. Zum Beispiel:
	
	`DEFINE WORK FILE 4 'TEST.WORK.PDS(TEST1)'`
	
	Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.

HFS-Dateien	Beispiel:
	
	`DEFINE WORK FILE 14 '/u/nat/rec/test.txt'`
	
	Der angegebene Pfadname muss existieren. Wenn das `DEFINE WORK FILE`-Statement ausgeführt wird, wird die HFS-Datei dynamisch zugewiesen. Falls das angegebene Member nicht existiert, wird ein neues Member unter diesem Namen angelegt.
	
	Bei der dynamischen Zuweisung des Datasets werden folgende z/OS-Pfadoptionen verwendet:
	
	`PATHOPTS=(OCREAT,OTRUNC,ORDWR)`
	`PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)`
	`FILEDATA=TEXT`
	
	Wird eine HFS-Datei geschlossen, wird sie automatisch von z/OS freigegeben (unabhängig vom Wert des Subparameters `FREE` des Profilparameters `WORK`).
	
	Um eine HFS-Datei zu lesen, müssen Sie statt des `DEFINE WORK FILE`-Statements die Programmierschnittstelle (API) `USR2021N` (dynamische Dataset-Zuweisung) verwenden, und zwar wegen der `OTRUNC`-Option. Diese Option setzt die HFS-Datei beim ersten Lesezugriff zurück und bewirkt eine leere Datei.

JES-Spoolfile-Klasse	Um ein JES-Spool-Dataset zu erzeugen, geben Sie `SYSOUT=`*`x`* an (wobei *`x`* die gewünschte Spoolfile-Klasse ist). Für die Standard-Spoolfile-Klasse geben Sie `SYSOUT=*` an.
	
	Beispiele:
	
	`DEFINE WORK FILE 10 'SYSOUT=A'`
	`DEFINE WORK FILE 12 'SYSOUT=*'`
	
	Um zusätzliche Parameter für die dynamische Zuweisung anzugeben, verwenden Sie statt des `DEFINE WORK FILE`-Statements die Programmierschnittstelle (API) `USR2021N` (dynamische Dataset-Zuweisung) in der Library `SYSEXT`.

NULLFILE	Um ein Dummy-Dataset zuzuweisen.

Zuweisung und Freigabe von Datasets

Wenn das DEFINE WORK FILE-Statement ausgeführt wird und ein physischer Dataset-Name, eine HFS-Datei, eine Spoolfile-Klasse oder ein Dummy-Dataset angegeben wurde, wird das entsprechende Dataset automatisch zugewiesen. Wenn die logische Datei bereits geöffnet ist, wird sie automatisch geschlossen, außer wenn der Subparameter CLOSE=FIN des Profilparameters WORK angegeben wurde, wobei dann eine Fehlermeldung ausgegeben wird. Außerdem wird ein bestehendes Dataset mit dem gleichen aktuellen DD-Namen automatisch freigegeben, bevor das neue Dataset zugewiesen wird. Um Fehler durch verfrühtes Öffnen von beim Programmstart noch nicht zugewiesenen Arbeitsdateien zu vermeiden, sollten Arbeitsdateien mit dem Subparameter OPEN=ACC (Öffnen bei erstem Zugriff) im Profilparameter WORK definiert werden.

Im Falle einer HFS-Datei oder einer im Profilparameter WORK mit Subparameter FREE=ON definierten Arbeitsdatei wird die Arbeitsdatei automatisch freigegeben, sobald sie geschlossen worden ist. Als Alternative steht Ihnen für die dynamische Zuweisung und Freigabe von Datasets die Programmierschnittstelle (API) USR2021N in Library SYSEXT zur Verfügung. Dieses API ermöglicht auch die Angabe zusätzlicher Parameter für die dynamische Zuweisung.

Arbeitsdateien in Server-Umgebungen

In Server-Umgebungen kann es zu Fehlern kommen, wenn mehrere Natural-Sessions versuchen, ein Dataset mit dem gleichen DD-Namen zuzuweisen oder zu öffnen. Um dies zu vermeiden, geben Sie entweder im Profilparameter WORK den Subparameter DEST=* an, oder Sie programmieren DEFINE WORK FILE '*' vor dem eigentlichen DEFINE WORK FILE-Statement; Natural generiert dann einen eindeutigen DD-Namen bei der Zuweisung der physischen Datasets, wenn das erste DEFINE WORK FILE-Statement für die betreffende Arbeitsdatei ausgeführt wird.

Alle Arbeitsdateien, deren DD-Namen mit CM anfangen, werden von allen Sessions in einer Server-Umgebung gemeinsam benutzt. Eine gemeinsame benutzte Arbeitsdatei, die für Ausgabe von der ersten Session geöffnet wird, wird bei Beendigung des Servers physisch geschlossen. Eine gemeinsame benutzte Arbeitsdatei, die für Eingabe geöffnet wird, wird physisch geschlossen, wenn die letzte Session sie schließt, d.h. wenn sie eine Dateiende-Bedingung erhält. Beim gleichzeitigen Lesen einer gemeinsam benutzten Arbeitsdatei wird einem READ WORK FILE-Statement nur ein Datei-Datensatz geliefert.

Arbeitsdateiname unter CICS

Für eine mit der Zugriffsmethode AM=CICS definierte Arbeitsdateinummer kann *operand1* in Abhängigkeit vom Subparameter TYPE des Profilparameters WORK für die Arbeitsdatei ein Übergangsdaten- oder Zwischenspeicher-Warteschlangen-Name (1 bis 8 Zeichen) sein. Für TYPE=TD (Übergangsdaten) werden nur die ersten 4 Zeichen von *operand1* berücksichtigt und das Ausgabemedium für die Übergangsdaten muss vorher für CICS definiert worden sein.

Weitere Informationen über Arbeitsdateien siehe auch den Abschnitt *Natural Print and Work Files under CICS* in der *Natural CICS Interface*-Dokumentation.

Arbeitsdateiname unter Com-plete/SMARTS

Unter Com-plete stehen PFS-Dateien mit der Zugriffsmethode `AM=SMARTS` zur Verfügung. Es kann ein Arbeitsdateiname zugewiesen werden, auch wenn er für Natural nicht definiert worden ist.

```
DEFINE WORK (14) '/nat/path/workfile'
DEFINE WORK (14) 'workfile'
```

Es ist vom Parameter `MOUNT_FS` von SMARTS abhängig, ob die Datei auf einem SMARTS Portable File System oder auf dem Native File System residiert. Das erste Element des Pfads (`/nat/`) legt das Zieldatei-System fest.

Wenn die Zeichenkette nicht mit einem Schrägstrich (/) beginnt, wird der Pfad der Datei von der Umgebungsvariablen `$NAT_WORK_ROOT` genommen.

Der angegebene Pfadname muss vorhanden sein. Wenn das Statement `DEFINE WORK FILE` ausgeführt wird, wird die Datei dynamisch zugewiesen. Wenn das angegebene Member nicht vorhanden ist, wird ein neues Member dieses Namens angelegt.

VI

- 52 DELETE 367
- 53 DELETE (SQL) 371
- 54 DISPLAY 377
- 55 DIVIDE 401
- 56 DO/DOEND 409
- 57 DOWNLOAD PC FILE 413
- 58 EJECT 421
- 59 END 427
- 60 END TRANSACTION 431
- 61 ESCAPE 437
- 62 EXAMINE 443
- 63 EXPAND 467

52 DELETE

- Funktion DELETE 368
- Einschränkung DELETE 368
- Syntax-Beschreibung DELETE 368
- Datenbank-spezifische Anmerkungen DELETE 369
- Beispiele DELETE 369

```
DELETE [RECORD] [IN] [STATEMENT] [(r)]
```

Dieser Abschnitt 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 DELETE

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 DELETE

Das DELETE-Statement darf nicht mit einem FIND-, READ- oder GET-Statement in derselben Quellcode-Zeile stehen.

Syntax-Beschreibung DELETE

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
(*r*)	**Statement-Referenz:** Die Notation (*r*) 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 DELETE

VSAM-Datenbanken	Das DELETE-Statement ist nicht auf VSAM-ESDS (Entry-Sequenced Data Sets) anwendbar.
SQL-Datenbanken	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. Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht gelöscht werden.

Beispiele DELETE

- 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 DELETE (SQL)

- Funktion DELETE (SQL) 372
- Syntax 1 - Searched DELETE 372
- Syntax 2 - Positioned DELETE 374

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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *DELETE - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion DELETE (SQL)

Das SQL-`DELETE`-Statement dient dazu, entweder Zeilen aus einer Tabelle zu löschen, ohne einen Cursor zu verwenden (Searched `DELETE`), oder Zeilen aus einer Tabelle zu löschen, auf die der Cursor positioniert ist (Positioned `DELETE`).

Zwei unterschiedliche Strukturen sind möglich:

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 Null, eine, mehrere oder alle Zeilen einer Tabelle löschen. Welche Zeilen gelöscht werden, bestimmen Sie mit einer Suchbedingung (*`search-condition`*), die auf die Tabelle angewandt wird. Optional ist es möglich, dem Tabellennamen einen *`correlation-name`* zuzuweisen.

Anmerkung: Die Anzahl der Zeilen, die mit einem Searched `DELETE` tatsächlich gelöscht worden sind, kann mit der Systemvariablen `*ROWCOUNT` ermittelt werden; see *Systemvariablen*-Dokumentation.

Common Set-Syntax:

```
DELETE FROM table-name [correlation-name] [WHERE search-condition]
```

Extended Set-Syntax:

```
DELETE FROM table-name [period-clause] [correlation-name]
[include-columns [SET assignment-clause]]
[WHERE search-condition]
[FETCH FIRST row-limit]
[      WITH { RR }      ]      [SKIP LOCKED DATA] [QUERYNO integer]
            { RS }
            { CS }
```

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`FROM table-name`	**FROM-Klausel:** In dieser Klausel wird die Tabelle angegeben, aus der die Zeilen gelöscht werden sollen.
period-clause	**Period-Klausel:** Gibt an, dass für das Ziel der Aktualisierungsoperation eine Period-Klausel gilt. Weitere Informationen siehe ***Period-Klausel*** im Abschnitt *Grundlegende Syntaxbestandteile*.
correlation-name	**Korrelationsname:** Optional. Dem Tabellennamen kann ein Korrelationsname zugeordnet werden.
include-columns	**Include Columns-Klausel:** Optional. Gibt einen Satz Spalten an, die zusammen mit den *table-name*-Spalten in die Ergebnistabelle des `DELETE`-Statement aufgenommen werden, wenn dieses in der `FROM`-Klausel eines `SELECT`-Statement verschachtelt ist. Weitere Informationen siehe ***include-columns***.
`SET` *assingment-clause*	**SET Assignment-Klausel:** Leitet die Zuordnung von Werten zu den inkludierten Spalten der Include Columns-Klausel ein. Siehe Assignment-Klausel beim SQL `UPDATE`-Statement.
`WHERE` *search-condition*	**WHERE-Klausel:** Diese Klausel dient zur Angabe der Selektionskriterien für die zu löschenden Zeilen. Wenn keine WHERE-Klausel angegeben wird, wird die gesamte Tabelle gelöscht.
`FETCH FIRST`	**FETCH FIRST-Klausel:** Begrenzt die Auswirkungen des `DELETE`-Statement auf eine Teilmenge der in Frage kommenden Zeilen. Entspricht der beim `SELECT`-Statement beschriebenen `FETCH FIRST`-Klausel zur Begrenzung der Zeilen.
`WITH`	**WITH Isolation Level-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese Klausel ermöglicht die explizite Angabe der beim Suchen der zu löschenden Zeile benutzten Isolationsstufe. `CS` — Cursor Stability

Syntax-Element	Beschreibung	
	RR	Repeatable Read
	RS	Read Stability
SKIP LOCKED DATA	**SKIP LOCKED DATA-Klausel:** Diese Klausel gibt an, dass Zeilen übersprungen werden, wenn auf der Zeile inkompatible Sperren durch andere Transaktionen vorhanden sind.	
QUERYNO *integer*	**QUERYNO-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese Klausel gibt die in EXPLAIN-Ausgaben und Ablaufverfolgungssätzen für dieses Statement zu benutzende Anzahl explizit an. Die Anzahl wird als QUERYNO-Spalte in der PLAN_TABLE für die Zeilen benutzt, die Informationen zu diesem Statement enthalten.	

Syntax 2 - Positioned DELETE

Ein Positioned DELETE bezieht sich immer auf einen Cursor innerhalb einer Datenbankschleife. Es muss daher dieselbe Tabelle referenzieren wie das entsprechende SELECT-Statement, sonst wird eine Fehlermeldung zurückgegeben. Ein Positioned DELETE kann nicht mit einer Selektion ohne Curosor verwendet werden.

In seiner Funktion entspricht das Positioned DELETE-Statement dem Natural-DML-Statement DELETE.

Common Set-Syntax:

```
DELETE FROM table-name WHERE CURRENT OF CURSOR [(r)]
```

Extended Set-Syntax:

```
DELETE FROM table-name WHERE CURRENT OF
CURSOR [(r)]   [ FOR ROW { [:]host-variable } OF ROWSET ]
                       { integer }
```

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`FROM table-name WHERE CURRENT OF CURSOR`	**FROM-Klausel:** Diese Klausel gibt die Tabelle an, aus der die Zeilen gelöscht werden sollen.
`(r)`	**Statement-Referenz:** Die Notation `(r)` dient zur Referenzierung des Statements, welches zur Selektion der zu löschenden Zeile verwendet wurde. Wenn keine Statement-Referenz angegeben wird, dann bezieht sich das `DELETE`-Statement auf die jeweils innerste aktive Verarbeitungsschleife, in der ein Datenbankdatensatz ausgewählt wurde.
`FOR ROW ... OF ROWSET`	**FOR ROW … OF ROWSET-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Die optionale `FOR ROW ... OF ROWSET`-Klausel für Positioned SQL DELETE-Statements gibt an, welche Zeile des aktuellen Rowset gelöscht werden muss. Sie sollte nur angegeben werden, wenn sich das `DELETE`-Statement auf ein `SELECT`-Statement mit Positionierung von Rowsets und Spalten-Arrays in dessen `INTO`-Klausel bezieht. Wenn diese Klausel weggelassen wird, werden alle Zeilen des aktuellen Rowset gelöscht.

54 DISPLAY

- Funktion DISPLAY 378
- Syntax-Beschreibung DISPLAY 378
- Standardwerte DISPLAY 391
- Beispiele DISPLAY 392

```
DISPLAY [( rep)] [options ] {[ / ... ] [output-format ] output-element} ...
```

Dieser Abschnitt 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 DISPLAY

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 DISPLAY

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
(*rep*)	**Report-Spezifikation:** Mit der Notation (*rep*) können Sie einen bestimmten anderen Report angeben, 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 (*rep*) nicht angegeben

Syntax-Element	Beschreibung
	wird, bezieht sich das DISPLAY-Statement auf den ersten ausgegebenen Report (Report 0). Wenn diese Druckdatei für Natural als PC definiert ist, wird der Report auf den PC per Download heruntergeladen, siehe ***Beispiel 8***. Weitere Informationen darüber, wie das Format eines Ausgabe-Reports gesteuert wird, siehe *Steuern der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
options	**Anzeige-Optionen:** Einzelheiten siehe ***Anzeige-Optionen*** weiter unten.
output-format	**Ausgabeformat-Definitionen:** Einzelheiten siehe ***Ausgabeformat-Definitionen*** weiter unten.
/	**Zeilenvorschub - Schrägstrich-Notation:** Ein Schrägstrich (/) innerhalb eines Textelementes bewirkt einen Zeilenvorschub innerhalb des Textes. 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. Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*: ■ *Zeilenvorschub - die Schrägstrich-Notation (/)* ■ *Beispiel für Zeilenvorschub in DISPLAY-Statement* ■ *Spaltenüberschriften unterdrücken - die Schrägstrich-Notation ('/')*
output-element	**Ausgabe-Element:** Einzelheiten siehe ***Ausgabe-Element*** weiter unten.

Anzeige-Optionen

```
[NOTITLE] [NOHDR] [ [AND] [GIVE] [SYSTEM] FUNCTIONS ] [(statement-parameters)]
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

NOTITLE	**Unterdrückung der Standard-Kopfzeile:**
	
	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 (TP-Betrieb) bzw. zu Beginn des Jobs (Batch-Betrieb) gesetzt. Die Standardkopfzeile kann mit einer eigenen Kopfzeile überschrieben werden, die mit einem WRITE TITLE-Statement angegeben wird. Das Schlüsselwort NOTITLE innerhalb des DISPLAY-Statements bewirkt, dass die Generierung einer Standardkopfzeile unterdrückt wird.
	
	Beispiele:
	
	■ Generierte Standard-Titelzeile wird ausgegeben:
	
	`DISPLAY NAME`
	
	■ Eigene Titelzeile wird ausgegeben:
	
	`DISPLAY NAME WRITE TITLE 'user-title'`
	
	■ Keine Titelzeile wird ausgegeben:
	
	`DISPLAY NOTITLE NAME`
	
	Anmerkung: Wird die NOTITLE-Option verwendet, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten auf denselben Report schreiben.

NOHDR	**Spaltenüberschriften:** Für jede mittels eines DISPLAY-Statements ausgegebene Spalte von Feldwerten wird eine Spaltenüberschrift ausgegeben; hierbei gilt folgendes: ■ Sie können mit dem DISPLAY-Statement explizit eine Spaltenüberschrift angeben, und zwar (in Apostrophen) vor dem jeweiligen Feldnamen. Zum Beispiel: `DISPLAY 'EMPLOYEE' NAME 'SALARY' SALARY` ■ 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.

Syntax-Element	Beschreibung

	■ Ist im DDM keine Spaltenüberschrift definiert, wird stattdessen der (im DDM definierte) Feldname als Überschrift verwendet.
	■ 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.
	`DISPLAY NAME SALARY #NEW-SALARY`
	■ 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.
	Unterdrücken von Spaltenüberschriften:
	Um die Spaltenüberschrift für ein einzelnes Feld zu unterdrücken, geben Sie vor dem betreffenden Feldnamen die folgenden Zeichen (Apostroph-Schrägstrich-Apostroph) an:
	`'/'`
	Beispiel:
	`DISPLAY '/' NAME 'SALARY' SALARY`
	Sollen gar keine Spaltenüberschriften ausgegeben werden, geben Sie das Schlüsselwort NOHDR (für „No Header“) an:
	`DISPLAY NOHDR NAME SALARY`
	Anmerkung:
	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	**Benutzung von Systemfunktionen:** Die GIVE SYSTEM FUNCTIONS-Klausel dient zur Auswertung der folgenden Natural-Systemfunktionen: AVER, COUNT, MAX, MIN, NAVER, NCOUNT, NMIN, SUM, TOTAL. Die Systemfunktionen werden ausgewertet, wenn das DISPLAY-Statement, das die GIVE SYSTEM FUNCTIONS-Klausel enthält, ausgeführt wird.

Syntax-Element	Beschreibung
	Die Systemfunktionen können anschließend von einem Statement, das aufgrund einer End-of-Page-Bedingung ausgeführt wird, referenziert werden. **Anmerkung:** 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. 3. Anstelle des Schlüsselworts `GIVE` können Sie auch das Schlüsselwort `GIVING` verwenden. Siehe auch ***Beispiel 2 - DISPLAY-Statement mit GIVE SYSTEM FUNCTIONS-Klausel***.
statement-parameters	**Parameter-Definition auf Statement-Ebene:** 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. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Quellcode-Zeilen erstrecken. **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)***. Siehe auch: ■ ***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	Attribut-Definition	SE
AL	Alphanumerische Länge der Ausgabe	SE
BX	Feldumrahmung (Box-Definition)	SE
CD	Farbdefinition	SE
CV	Kontrollvariable	SE
DF	Datumsformat	SE
DL	Ausgabelänge	SE
DY	Dynamische Attribute	SE
EM	Edit Mask	SE
EMU	Unicode-Editiermaske	E
ES	Leerzeilenunterdrückung	S
FC	Füllzeichen für DISPLAY-Statement	SE
FL	Gleitkomma-Mantissenlänge	SE
GC	Füllzeichen für Gruppenüberschriften	SE
HC	Überschriften-Zentrierung	SE
HW	Überschriftenbreite	SE
IC	Einfügungszeichen	SE
ICU	Unicode-Einfügungszeichen	SE
IS	Unterdrückung identischer Werte	SE
LC	Vorangestellte Zeichens	SE
LCU	Vorangestellte Unicode-Zeichen	SE
LS	Zeilenlänge	S
MC	Anzahl multipler Feldwerte (Kann nur im Reporting Mode verwendet werden.)	S
MP	Maximale Seitenzahl eines Reports	S
NL	Numerische Länge der Ausgabe	SE
PC	Anzahl der Periodengruppen-Ausprägungen (Kann nur im Reporting Mode verwendet werden.)	S
PM	Druck-Modus	SE
PS	Länge einer Reportseite	S
SF	Spaltenabstand	SE
SG	Vorzeichen-Stelle	SE
TC	Nachgezogene Zeichen	SE

Session-Parameter, die beim DISPLAY-Statement angegeben werden können		Spezifikation (S = auf Statement-Ebene, E = auf Elementebene)
TCU	Nachgezogene Zeichen (Unicode)	SE
UC	Unterstreichungszeichen	SE
ZP	Anzeige von Nullwerten	SE

Die einzelnen Parameter sind in der *Parameter-Referenz*-Dokumentation beschrieben.

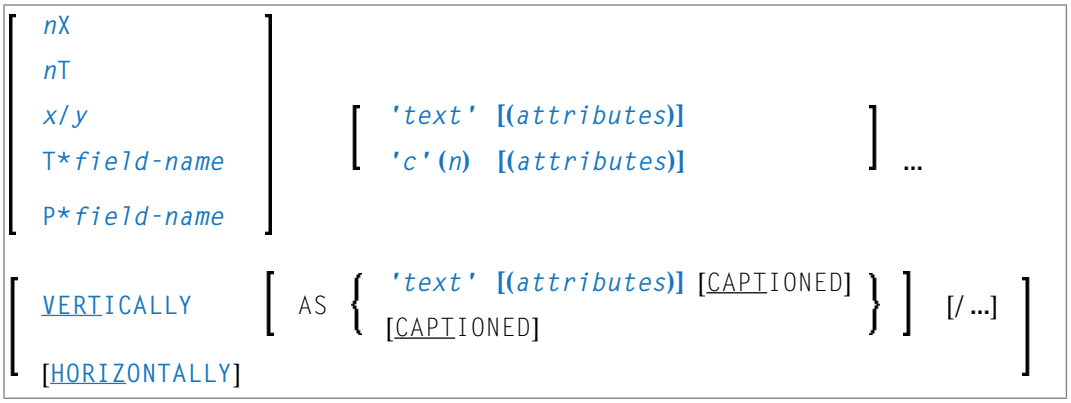
Siehe auch die folgenden Themen im *Leitfaden zur Programmierung*:

- *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 (PM=I)    'Text'          '='   VARI             /*    Text 4321
DISPLAY NOHDR           'Text' (PM=I)   '='   VARI (PM=I)      /*    txeT 4321
DISPLAY NOHDR           'Text' (PM=I)   '='   VARI             /*    txeT 1234
END
```

Ausgabeformat-Definitionen



Feldpositionierung

Syntax-Element	Beschreibung
*n*X	**Spaltenabstand:** Mit dieser Notation fügen Sie zwischen den auszugebenden Spalten *n* Leerstellen ein. *n* darf nicht 0 sein. Beispiel: `DISPLAY NAME 5X SALARY` Siehe auch: ■ ***Beispiel 1 - DISPLAY-Statement mit Notation nX- und nT*** (weiter unten) ■ *Spaltenabstand - der SF-Parameter und die Notation nX* (im *Leitfaden zur Programmierung*)
*n*T	**Setzen von Tabulatoren:** Mit dieser Notation setzen Sie Tabulatoren, d.h. die Ausgabe eines Wertes beginnt ab Spalte *n*. Zurückpositionieren ist nicht erlaubt. Im folgenden Beispiel wird NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben: `DISPLAY 25T NAME 50T SALARY` Siehe auch: ■ ***Beispiel 1 - DISPLAY-Statement mit Notation nX- und nT*** (weiter unten) ■ *Tabulator-Notation nT* (im *Leitfaden zur Programmierung*)
x/*y*	**x/y-Positionierung:** 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. Zurückpositionieren ist nicht erlaubt.
T**field-name*	**Feldbezogene Positionierung:** Mit dieser Notation wird die Position eines Feldes nach der Position eines in einem vorhergehenden DISPLAY-Statement ausgegebenen Feldes (*field-name*) ausgerichtet. Zurückpositionieren ist nicht erlaubt.
P**field-name*	**Feld- und zeilenbezogene Positionierung:**

Syntax-Element	Beschreibung
	Mit dieser Notation werden Position und Ausgabezeile eines Feldes nach denen eines in einem vorhergehenden `DISPLAY`-Statement ausgegebenen Feldes (*`field-name`*) ausgerichtet. Dies wird meist bei vertikalen Ausgaben eingesetzt. Zurückpositionieren ist nicht erlaubt. Siehe auch: ■ ***Beispiel 3 - DISPLAY-Statement mit der Notation P**** (weiter unten) ■ *Tab-Notation P*`field`* (im *Leitfaden zur Programmierung*)

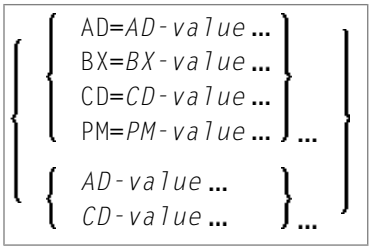
Spaltenüberschriften

Syntax-Element	Beschreibung
'*`text`*' '/'	**Textzuweisung:** 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: `DISPLAY 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT` Werden vor einem Feldnamen mehrere Textelemente '*`text`*' angegeben, so wird das *letzte* als Spaltenüberschrift verwendet und die anderen werden in der Ausgabespalte vor dem Feldwert ausgegeben. Siehe auch: ■ *Eigene Spaltenüberschriften definieren* (im *Leitfaden zur Programmierung*) ■ *Text Notation, Text-Notation, Text für ein Statement definieren* (im *Leitfaden zur Programmierung*) ■ ***Beispiel 4 - DISPLAY-Statement mit 'text', 'c'(n) und Attribut-Notation*** (weiter unten)
'*`c`*'*(n)*	**Wiederholungszeichen:** Das in Apostrophen (') stehende Zeichen *`c`* (character) wird *n*-mal unmittelbar vor dem Feldwert ausgegeben. Beispiel: `DISPLAY '*' (5) '=' NAME` führt zur Ausgabe von:

Syntax-Element	Beschreibung
	`***** SMITH` Siehe auch: ■ *Text-Notation, n mal vor einem Feldwert anzuzeigendes Zeichen definieren* (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*
- *BX - Box 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:

Syntax-Element	Beschreibung
VERTICALLY	**DISPLAY VERT ohne AS-Klausel:** Vertikale Spaltenausrichtung. Es wird keine Spaltenüberschrift erzeugt, wenn die AS-Klausel weggelassen wird. Beispiel: `DISPLAY VERT NAME SALARY` Siehe auch Beispiel *DISPLAY VERT ohne AS-Klausel* im *Leitfaden zur Programmierung*.
AS '*text*'	**DISPLAY VERT AS '*text*'-Klausel:** Vertikale Spaltenausrichtung. Wenn AS '*text*' angegeben wird, wird der in Apostrophen stehende Text als Spaltenüberschrift ausgegeben. Siehe auch Beispiel *DISPLAY VERT AS 'text'* im *Leitfaden zur Programmierung*. Wenn Sie einen Schrägstrich (/) in der Zeichenkette '*text*' angeben, werden mehrere Zeilen mit Spaltenüberschriften erzeugt. Beispiel: `DISPLAY VERT AS 'LAST/NAME' NAME`
AS '*text*' CAPTIONED	**DISPLAY VERT AS '*text*' CAPTIONED-Klausel:** 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. Beispiel: `DISPLAY VERT AS 'PERSONS/SELECTED' CAPTIONED NAME FIRST-NAME ↩` Siehe auch Beispiel *DISPLAY VERT AS 'text' CAPTIONED* im *Leitfaden zur Programmierung*.
AS CAPTIONED	**DISPLAY VERT AS CAPTIONED-Klausel:** Vertikale Spaltenausrichtung. Wenn AS CAPTIONED angegeben wird, wird der standardmäßige Überschriften-Text für das Feld ausgegeben (entweder Überschriften-Text oder den Feldnamen). Beispiel:

Syntax-Element	Beschreibung
	`DISPLAY VERT AS CAPTIONED NAME FIRST-NAME`
`HORIZONTALLY`	**DISPLAY HORIZONTAL-Klausel:** Horizontale Spaltenausrichtung. Dies ist der standardmäßige Anzeigemodus.

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.

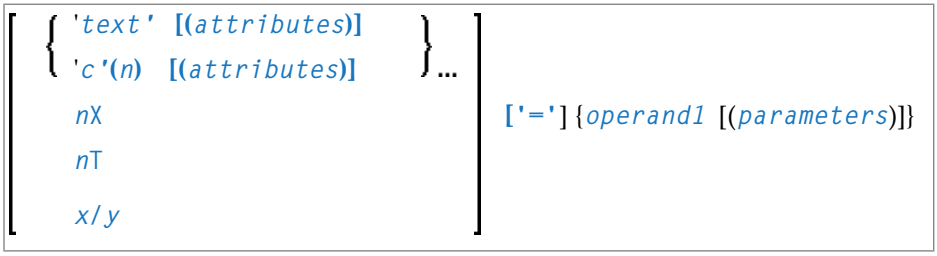
Es sind bis zu 15 Spaltenüberschriftenzeilen erlaubt.

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



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
		S	A	G	N	A	N	P	I	F	B	D	T	L		G	O		
operand1		S	A	G	N	A	N	P	I	F	B	D	T	L		G	O	ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
nX	**Spalten-Abstand:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
nT	**Setzen von Tabulatoren:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
x/y	**x/y-Positionierung:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
'text'	**Textzuweisung:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
'c'(n)	**Wiederholungszeichen:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
'text' '=' *'c' (n) '='*	Wird *'text' '='* vor einem Feld angegeben, so wird *text* unmittelbar vor dem Feldwert ausgegeben. Dies gilt analog dazu für *'c' (n) '='*. Beispiel:

Syntax-Element	Beschreibung
	`DISPLAY '*****' '=' NAME`
attributes	**Ausgabe-Attribute:** Wie unter *Ausgabeformat-Definitionen* (**siehe oben**).
operand1	Das auszugebende Feld.
parameters	**Parameter-Definiton auf Elementebene (Feldebene):** Unmittelbar nach *operand1* 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. Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Quellcode-Zeilen erstrecken. Siehe auch: ■ ***Liste der Parameter*** ■ ***Beispiel für die Benutzung von Parametern auf Statement- und Elementebene (Feldebene)***

Standardwerte DISPLAY

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 DISPLAY

- 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         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

50005500  BLOND                ALEXANDRE               172000 FRA
50005300  MAIZIERE             ELISABETH               166900 FRA
50004900  CAOUDAL              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            ANNUAL   CURRENCY
                             POSITION            SALARY     CODE
-------------------- ------------------------- ---------- --------

ABELLAN              MAQUINISTA                  1450000 PTA
                                                 1392000 PTA

ACHIESON             DATA BASE ADMINISTRATOR       11300 UKL
                                                   10500 UKL

ADAM                 CHEF DE SERVICE              159980 FRA
                                                       0
```

```
ADKINSON             PROGRAMMER                     34500 USD
                                                    31700 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              SALARY   CURRENCY
          CITY
        POSITION
------------------------- ---------- --------

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
*
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             NAME         +++++++++++++++TELEPHONE+++++++++++++++
       ID
                                             AREA              TELEPHONE
                                             CODE
================   ================   ==================   ==================

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
```

55 DIVIDE

- Funktion DIVIDE 402
- Syntax 1 - DIVIDE-Statement ohne GIVING-Klausel 402
- Syntax 2 - DIVIDE-Statement mit GIVING-Klausel 403
- Syntax 3 - DIVIDE-Statement mit REMAINDER-Klausel 404
- Beispiel für DIVIDE-Statement 406

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: ADD | COMPRESS | COMPUTE | EXAMINE | MOVE | MOVE ALL | MULTIPLY | RESET | SEPARATE | SUBTRACT

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion DIVIDE

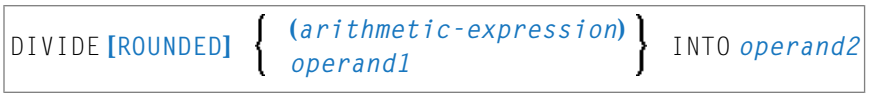
Mit dem Statement DIVIDE können Sie einen Operanden durch einen anderen dividieren.

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.

Dieses Statement hat drei verschiedene Syntax-Strukturen.

Syntax 1 - DIVIDE-Statement ohne GIVING-Klausel



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A		N		N	P	I	F							yes	no
operand2		S	A		M		N	P	I	F							yes	no

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung:
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 INTO *operand2*	**Operanden:** *operand1* ist der Divisor, *operand2* ist der Dividend. Das Ergebnis wird in *operand2* (Ergebnisfeld) gespeichert. Somit ist das Statement gleichbedeutend mit: `operand2 := operand2 / operand1` Wenn ein *arithmetischer Ausdruck* benutzt wird, darf *operand2* kein Array-Bereich sein. Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. *operand2*) ausgewertet. Weitere Informationen siehe *Genauigkeit von Ergebnissen bei arithmetischen Operationen* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Syntax 2 - DIVIDE-Statement mit GIVING-Klausel



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

Operanden-Definitionstabelle:

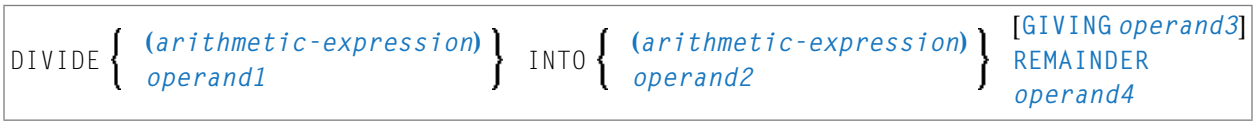
Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A		N			N	P	I	F								yes	no
operand2	C	S	A		N			N	P	I	F								yes	no
operand3		S	A			A	U	N	P	I	F	B*							yes	yes

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung:
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 INTO *operand2* GIVING *operand3*	**Operanden:** *operand1* ist der Divisor, *operand2* ist der Dividend. Das Ergebnis wird in *operand3* (Ergebnisfeld) gespeichert. Somit ist das Statement gleichbedeutend mit: `operand3 := operand2 / operand1` Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. *operand3*) ausgewertet. Weitere Informationen siehe *Genauigkeit von Ergebnissen bei arithmetischen Operationen* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Syntax 3 - DIVIDE-Statement mit REMAINDER-Klausel



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

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A		N			N	P	I								yes	no
operand2	C	S	A		N			N	P	I								yes	no
operand3		S	A			A	U	N	P	I	F	B*		T				yes	yes
operand4		S	A			A	U	N	P	I	F	B*		T				yes	yes

* Format B von *operand3* und *operand4* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung:
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 *operand2*	**Operanden:** *operand1* ist der Divisor, *operand2* ist der Dividend. Wird die GIVING-Klausel nicht benutzt, wird das Ergebnis in *operand2* gespeichert. Das Ergebnisfeld kann ein Datenbankfeld oder eine Benutzervariable sein. Falls *operand2* eine Konstante oder eine nicht änderbare Natural-Systemvariable ist, dann ist die GIVING-Klausel erforderlich.
GIVING *operand3*	**GIVING-Klausel:** Wird diese Klausel benutzt, dann wird *operand2* nicht verändert, und das Ergebnis wird in *operand3* gespeichert. Die Anzahl der Dezimalstellen für das Ergebnis der Division wird vom Ergebnisfeld (d.h. *operand2*, falls keine GIVING-Klausel benutzt wird, oder *operand3*, falls die GIVING-Klausel benutzt wird) ausgewertet. Weitere Informationen siehe *Genauigkeit von Ergebnissen bei arithmetischen Operationen* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

REMAINDER *operand4*	**REMAINDER-Klausel:**
	
	Der nach der Division verbleibende Rest wird in das in *operand4* angegebene Feld gestellt.
	
	■ Wenn die GIVING-Klausel benutzt wird, ist das Statement gleichbedeutend mit:
	
	`operand3 := operand2 / operand1`
	`operand4 := operand2 - (operand3 * operand1)`
	
	Keiner der vier Operanden darf ein Array-Bereich sein.
	
	■ Wenn die GIVING-Klausel nicht benutzt wird, ist das Statement gleichbedeutend mit:

Syntax-Element	Beschreibung:
	temporary := operand2 *operand2 := operand2 / operand1* *operand4 := temporary - (operand2 * operand1)* dabei ist *temporary* ein temporäres Feld mit Format/Länge wie bei *operand2*. Für jeden dieser Schritte gelten die unter *Genauigkeit von Ergebnissen bei arithmetischen Operationen* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung* beschriebenen Regeln.

Beispiel für DIVIDE-Statement

```
** 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
```

56 DO/DOEND

- Funktion DO/DOEND 410
- Einschränkungen DO/DOEND 410
- Beispiel DO/DOEND 411

```
DO statement ... DOEND
```

Dieser Abschnitt behandelt folgende Themen:

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

Gehört zur Funktionsgruppe: ***Reporting Mode-Statements***

Funktion DO/DOEND

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

Anmerkung: Wenn Sie nur ein einziges Statement angeben, das in Abhängigkeit von einer logischen Bedingung ausgeführt werden soll, können Sie die Statements DO und DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Einschränkungen DO/DOEND

- 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 DO/DOEND

```
** 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:

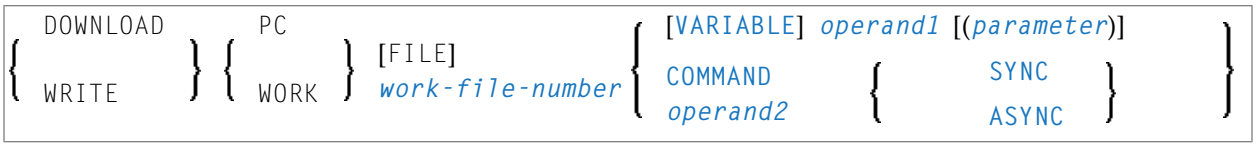
```
PERSONNEL         NAME           ANNUAL           MAKE         MAINT-COST
   ID                            SALARY
--------- -------------------- ---------- -------------------- ----------

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
```

57 DOWNLOAD PC FILE

- Funktion DOWNLOAD PC FILE 414
- Syntax-Beschreibung DOWNLOAD PC FILE 414
- Externe Darstellung der Felder 416
- Beispiele DOWNLOAD PC FILE 417



Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements: CLOSE PC FILE | UPLOAD PC FILE | WRITE WORK FILE

Funktion DOWNLOAD PC FILE

Dieses Statement dient dazu, Daten zum PC zu übertragen.

Siehe auch die *Natural Connection-Dokumentation* und die *Entire Connection-Dokumentation*

Syntax-Beschreibung DOWNLOAD PC FILE

Operanden-Definitionstabelle:

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

Die Formate C und G sind bei Natural Connection nicht gültig.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Zu benutzende Arbeitsdateinummer:** Diese Nummer muss der Nummer unter den Arbeitsdateinummern für den PC entsprechen, wie sie in Natural definiert sind.
VARIABLE	**Format:** Die Datensätze in der PC-Datei werden in variablem Format geschrieben. Beachten Sie, dass variable Datensätze nicht in PC Spreadsheet-Formate konvertiert werden können.
operand1	**Feld-Spezifikation:**

Syntax-Element	Beschreibung
	Mit *operand1* geben Sie die Felder an, die zum PC heruntergeladen werden sollen.
COMMAND	**COMMAND-Klausel:** Mit dieser Klausel können Sie PC-Kommandos (d.h. Kommandos, die in die Kommandozeile von Entire Connection auf dem PC eingegeben werden können) zum PC senden. Entire Connection überprüft, ob das gesendete Kommando gültig ist oder nicht. Ein Kommando, das vom PC nicht erkannt werden kann, wird zurückgewiesen. In diesem Fall gibt Natural eine Fehlermeldung mit dem Inhalt aus, dass das heruntergeladene Kommando vom PC zurückgewiesen wurde. Sie können die COMMAND-Klausel benutzen, um zum Beispiel Entire Connection-Tasks von einem z/OS-Computer aus auszuführen. Wenn Sie die Task DIR haben, die PC Directory-Informationen anzeigt, können Sie diese direkt aus Ihrem Natural-Programm heraus auf einem z/OS-Computer mit dem folgenden Statement anstoßen: `DOWNLOAD PC FILE 7 COMMAND 'DIR'` In ***Beispiel 2*** weiter unten wird die COMMAND-Klausel benutzt, um den Namen der PC-Datei zu definieren, die die heruntergeladenen Daten aufnehmen soll. Auf diese Art können Sie die Eingabeaufforderung nach dem Namen der Datei umgehen.
operand2	**COMMAND-Spezifikation:** Mit *operand2* geben Sie das DOS-Kommando oder die Entire Connection-Task an, die auf dem PC ausgeführt werden soll. *operand2* muss eine alphanumerische Konstante oder Variable sein.
SYNC	**SYNC-Option:** Mit SYNC geben Sie an, dass der PC nach Ausführen und Beenden von COMMAND die Kontrolle an Natural zurückgibt. SYNC kann benutzt werden, um zum Beispiel sicherzustellen, dass das Kommando SET PCFILE ausgeführt worden ist, bevor eine Datei-Übertragung startet.
ASYNC	**A SYNC-Option:** Mit ASYNC legen Sie fest, dass der PC die Kontrolle sofort an Natural zurückgibt, unabhängig davon, ob die Ausführung von COMMAND beendet wurde oder nicht.
parameter	**Editiermaske-Parameter:** Als *parameter* können Sie den Session-Parameter EM oder EMU angeben:
	EM= **Editiermaske** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der *Parameter-Referenz*-Dokumentation. Der

Syntax-Element	Beschreibung	
		EM-Parameter kann nicht bei Gruppenoperanden angewendet werden.
	EMU=	**Unicode-Editiermaske** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EMU in der *Parameter-Referenz*-Dokumentation. Der EMU-Parameter kann nicht bei Gruppenoperanden angewendet werden.

Externe Darstellung der Felder

In der externen Datei werden Felder, die mit einem DOWNLOAD PC FILE-Statement geschrieben werden, entsprechend ihrer internen Definition dargestellt, es sei denn, es wird eine Editiermaske angewendet.

- Felder ohne Eingabemasken
- Felder mit Eingabemasken

Felder ohne Eingabemasken

Bei Feldern ohne Editiermasken werden die Feldwerte nicht bearbeitet, sondern in ihrem internen Format geschrieben.

- **Format A und B**
  Die Anzahl der Bytes in der externen Datei entspricht der im Natural-Programm definierten internen Länge. Es erfolgt keine Bearbeitung und der Wert enthält kein Dezimalzeichen.
- **Format N**
  Die Anzahl der Bytes in der externen Datei entspricht der Summe der internen Stellen vor und nach dem Dezimalzeichen. Das Dezimalzeichen ist in der Ausgabe nicht enthalten.
- **Format P**
  Die Anzahl der Bytes in der externen Datei entspricht der Summe der Stellen vor und nach dem Dezimalzeichen, plus 1 für das Vorzeichen, geteilt durch 2 und aufgerundet auf das nächste volle Byte.

Anmerkung: Bei Feldern, die ohne Editiermaske geschrieben werden, wird keine Formatkonvertierung durchgeführt.

Beispiele für Felddarstellung (ohne Eingabemasken)

Felddefinition	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

Felder mit Eingabemasken

Bei Anwendung einer Editiermaske wird das Feld zunächst gemäß der angegebenen Editiermaske formatiert, und *der resultierende formatierte Wert wird in die Arbeitsdatei geschrieben*.

- In diesem Fall wird die interne Darstellung nicht verwendet.
- Die Länge des in die Datei geschriebenen Wertes entspricht der Länge der resultierenden formatierten Zeichenkette.

Beispiele DOWNLOAD PC FILE

- Beispiel 1 - Benutzung des Statements DOWNLOAD PC FILE
- Beispiel 2 - Benutzung der COMMAND-Klausel
- Beispiel 3 - DOWNLOAD PC FILE mit Editiermaske

Beispiel 1 - Benutzung des Statements DOWNLOAD PC FILE

Das folgende Programm veranschaulicht die Benutzung des Statements `DOWNLOAD PC FILE`. Die Daten werden zunächst selektiert und dann mittels Arbeitsdatei `PC FILE 7` zum PC heruntergeladen.

```
** Example 'PCDOEX1': DOWNLOAD PC FILE
**
** 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
  DOWNLOAD PC FILE 7 CITY NAME PERSONNEL-ID    /* Data download
```

```
END-FIND
END
```

Ausgabe des Programms PCDOEX1:

Wenn Sie das Programm starten, erscheint ein Fenster, in dem Sie den Namen der PC-Datei angeben, in die die Daten heruntergeladen werden sollen. Die Daten werden dann auf den PC heruntergeladen.

Beispiel 2 - Benutzung der COMMAND-Klausel

Das folgende Programm veranschaulicht die Benutzung der COMMAND-Klausel im Statement DOWNLOAD PC FILE. Der Name der empfangenden PC-Datei wird als erstes definiert. Dann werden die Daten selektiert und auf diese Datei heruntergeladen.

```
** Example 'PCDOEX2': DOWNLOAD PC FILE
**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
01 PERS VIEW OF EMPLOYEES
  02 PERSONNEL-ID
  02 NAME
  02 CITY
01 CMD (A80)                                  /* Variable for transfer
END-DEFINE                                    /* of the PC command
*
MOVE 'SET PCFILE 7 DOWN DATA PERS.NCD' TO CMD  /* PC command to define
*
DOWNLOAD PC FILE 6 COMMAND CMD                /* Command download
*
FIND PERS WITH CITY = 'NEW YORK'              /* Data selection
  DOWNLOAD PC FILE 7 CITY NAME PERSONNEL-ID   /* Data download
END-FIND
END
```

Anmerkung: Die PC-Dateinummern in zwei aufeinanderfolgenden DOWNLOAD PC FILE-Statements müssen sich unterscheiden.

Ausgabe des Programms PCDOEX2:

Wenn Sie das Programm starten, werden die Daten zu der PC-Datei heruntergeladen, die im Programm angegeben wurde. Ein Fenster erscheint nicht.

Beispiel 3 - DOWNLOAD PC FILE mit Editiermaske

Das folgende Programm veranschaulicht die Benutzung des Statement `DOWNLOAD PC FILE`. Die Daten werden zunächst ausgewählt und dann unter Verwendung der Arbeitsdatei 6 auf den PC heruntergeladen.

```
** Example 'PCDOEX3': DOWNLOAD PC FILE with Edit Mask                    ↩

**
** NOTE: Example requires that Natural Connection is installed.
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
  2 BIRTH
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DOWNLOAD PC FILE 6
         PERSONNEL-ID NAME BIRTH (EM=YYYY/MM/DD)
END-FIND
```

Ausgabe des Programms `PCDOEX3`:

Wenn Sie das Programm ausführen, werden die Daten in die im Programm angegebene PC-Datei heruntergeladen. Es wird kein Fenster angezeigt.

58 EJECT

- Funktion EJECT 422
- Syntax-Beschreibung EJECT 422
- Verarbeitung 424
- Beispiel EJECT 424

Dieser Abschnitt 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 EJECT

Das EJECT-Statement kann dazu verwendet werden, einen Seitenvorschub auszulösen.

Vgl. auch Natural Profil- und Session-Parameter EJ in der *Parameter-Referenz*.

Syntax-Beschreibung EJECT

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



Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung	
EJECT ON/OFF (*rep*)	**Mit Report-Spezifikation - Online- und Batch-Verarbeitung:**	
	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:**	

Syntax-Element	Beschreibung	
	`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 *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.	

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:

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 *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
`IF LESS THAN operand1 LINES LEFT`	Ein Seitenvorschub wird nur ausgeführt, wenn die aktuelle Zeile für die Seite größer als die Seitenlänge minus `operand1` ist. `operand1` 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` oder `WRITE TRAILER` verknüpften Statements aus. Ebensowenig 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 EJECT

```
** 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 '%%'                          47T '%%' /
              20T '%%' 3X 'REPORT OF EMPLOYEES' 47T '%%' /
              20T '%%' 3X '  SORTED BY CITY    ' 47T '%%' /
              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
************************************************************************
```

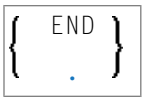
Nach dem Drücken von EINGABE:

```

************************************************************************
ALBUQUERQUE          ROLLING              MANAGER                             2
************************************************************************
************************************************************************
ALBUQUERQUE          FREEMAN              MANAGER                             5
************************************************************************
************************************************************************
ALBUQUERQUE          LINCOLN              ANALYST                             8
************************************************************************
************************************************************************
ALFRETON             GOLDBERG             JUNIOR                             11
************************************************************************
```

59 END

- Funktion END 428
- Syntax-Beschreibung END 428
- Beispiele END 429



Dieser Abschnitt behandelt folgende Themen:

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

Funktion END

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 Quellcode-Zeilennummer) 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

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
END	**Schlüsselwort:** Das für Natural reservierte Schlüsselwort END dient normalerweise zum Markieren des physischen Endes eines Natural-Programms.
.	**Punkt:** Anstatt des für Natural reservierten Schlüsselworts END kann ein Punkt (.) benutzt werden. Falls Sie statt END einen Punkt (.) verwenden und sich in derselben Zeile noch andere Statements befinden, müssen Sie dem Punkt mindestens ein Leerzeichen voranstellen.

Beispiele END

Einige typische Beispiele finden Sie im Abschnitt *Beispiele für die Benutzung des DEFINE DATA-Statements*.

60 END TRANSACTION

- Funktion END TRANSACTION 432
- Einschränkung END TRANSACTION 433
- Syntax-Beschreibung END TRANSACTION 433
- Betroffene Datenbanken 433
- Datenbank-spezifische Anmerkungen 434
- Beispiele END TRANSACTION 434

```
END [OF] TRANSACTION [operand1 ...]
```

Dieser Abschnitt 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 END TRANSACTION

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 END TRANSACTION

Das Statement END TRANSACTION kann nicht mit Entire System Server benutzt werden.

Syntax-Beschreibung END TRANSACTION

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S			N	A	U	N	P	I	F	B	D	T					ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Speicherung von Transaktionsdaten:** 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 GET TRANSACTION DATA-Statement wieder gelesen werden. Die Transaktionsdaten werden auf die mit dem Profilparameter ETDB angegebene Datenbank geschrieben. Wenn der ETDB-Parameter nicht gesetzt ist, werden die Transaktionsdaten auf die mit dem Profilparameter UDB angegebene Datenbank geschrieben. **Ausnahme**: Auf z/OS-Computern werden die Transaktionsdaten auf die Datenbank geschrieben, auf der sich die Natural-Security-Systemdatei (FSEC) befindet (ist FSEC nicht angegeben, dann ist sie identisch mit der Natural-Systemdatei FNAT; ist Natural Security nicht installiert, dann werden die Transaktionsdaten auf die Datenbank geschrieben, auf der sich FNAT befindet).

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 bestimmen Datenbank gespeichert. Für welche Datenbanken das Statement außerdem ausgeführt wird, hängt vom `ET`-Parameter (siehe oben) ab.

Datenbank-spezifische Anmerkungen

VSAM Databases	Informationen zur Transaktionslogik, die beim Zugriff auf VSAM gilt, finden Sie in der *Natural for VSAM* -Dokumentation.
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.

Beispiele END TRANSACTION

- 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
```

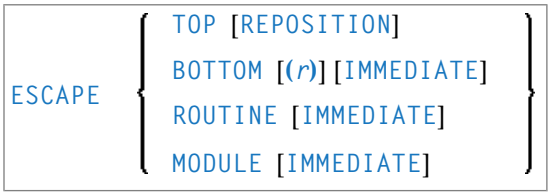
Nach Änderung und Bestätigung der Personalnummer:

```
NAME LAWLER
FIRST-NAME SUNNY
CITY MILWAUKEE
```

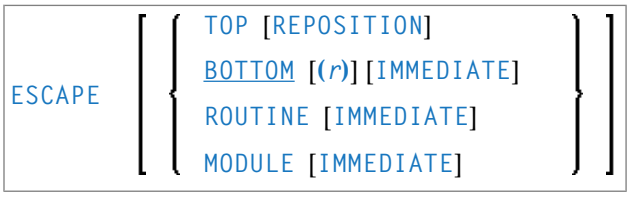
61 ESCAPE

- Funktion ESCAPE 438
- Syntax-Beschreibung ESCAPE 439
- Beispiel für ESCAPE-Statement 441

Structured Mode-Syntax



Reporting Mode-Syntax



Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements:

- FIND | FOR | HISTOGRAM | PARSE JSON | PARSE XML | READ | READ RESULT SET (SQL) | READ WORK FILE | READLOB | REPEAT | SORT
- CALL | CALL FILE | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | FETCH | PERFORM

Gehört zur Funktionsgruppe:

- *Schleifenverarbeitung*
- *Aufruf von Programmen und Subprogrammen*

Funktion ESCAPE

Das Statement ESCAPE dient dazu, den linearen Ablauf der Ausführung einer Verarbeitungsschleife oder eines Subprogramms 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 Subprogramm (Subroutine, Subprogramm oder mit FETCH RETURN aufgerufenes Programm), werden die innerhalb der Verarbeitungsschleife aufgerufenen Subprogramme 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

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
ESCAPE TOP	**TOP-Option:** TOP bedeutet, dass die Verarbeitung am Anfang der Verarbeitungsschleife fortgesetzt werden soll, d.h. die Schleife wird erneut von Anfang an durchlaufen.
REPOSITION	**TOP REPOSITION-Option:** Wenn ein ESCAPE TOP REPOSITION-Statement ausgeführt wird, fährt Natural sofort mit der Verarbeitung am Anfang der aktiven READ-Schleife fort und benutzt dabei den aktuellen Wert der Suchvariable als neuen Startwert. Gleichzeitig setzt ESCAPE TOP REPOSITION die Systemvariable *COUNTER auf Null (0). ESCAPE TOP REPOSITION kann innerhalb einer READ-Statementschleife angegeben werden, die auf Adabas- oder VSAM-Datenbanken zugreift. Das betreffende READ-Statement muss die Option WITH REPOSITION enthalten.
ESCAPE BOTTOM	**BOTTOM-Option:** BOTTOM bedeutet, dass die Verarbeitung mit dem ersten Statement nach der Verarbeitungsschleife fortgesetzt werden soll. Die Schleife wird beendet, und schleifenabschließende Verarbeitungen (abschließendes BREAK und END DATA) werden für alle zu beendenden Schleifen ausgeführt. Im Reporting Mode ist ESCAPE BOTTOM die Voreinstellung.
(*r*)	**Statement-Referenzierung:**

Syntax-Element	Beschreibung
	Notation (*r*): Wenn auf BOTTOM 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 ESCAPE BOTTOM-Statement innerhalb der referenzierten Verarbeitungschleife steht.
IMMEDIATE	**IMMEDIATE-Option:** Wenn Sie das Schlüsselwort IMMEDIATE angeben, werden keine abschließenden schleifenbeendenden Verarbeitungen durchgeführt.
ESCAPE ROUTINE	**ROUTINE-Option:** Diese Option bewirkt, dass das aktive Natural-Subprogramm, das entweder über PERFORM, CALLNAT, FETCH RETURN oder als Hauptprogramm aufgerufen wurde, die Kontrolle abgibt. Im Falle einer Subroutine wird die Verarbeitung mit dem ersten Statement fortgesetzt, das auf das Statement folgt, mit dem die Subroutine aufgerufen wurde. Im Falle eines Hauptprogramms gelangt Natural in den Kommando-Modus. Alle aktiven Schleifen innerhalb des Subprogramms 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.
ESCAPE MODULE	**MODULE-Option:** 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. Wenn ESCAPE MODULE in einer Hierarchie interner Subroutinen benutzt wird, ermöglicht es diese Option, alle auf dieser Ebene laufenden Subprogramme sofort zu verlassen. Wenn keine interne Subroutine aktiv ist, führt ESCAPE MODULE zum gleichen Ergebnis wie ESCAPE ROUTINE. ESCAPE MODULE ist nur bei internen Subroutinen von Bedeutung. Bei externen Subroutinen, Subprogrammen und aufgerufenen Programmen hat diese Option denselben Effekt wie ESCAPE ROUTINE. Wie bei ESCAPE ROUTINE wird eine schleifenabschließende Verarbeitung ausgeführt. Geben Sie aber das Schlüsselwort IMMEDIATE an, wird keine schleifenbeendende Verarbeitung ausgeführt.

Beispiel für ESCAPE-Statement

```
** 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=O) '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
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
```

62 EXAMINE

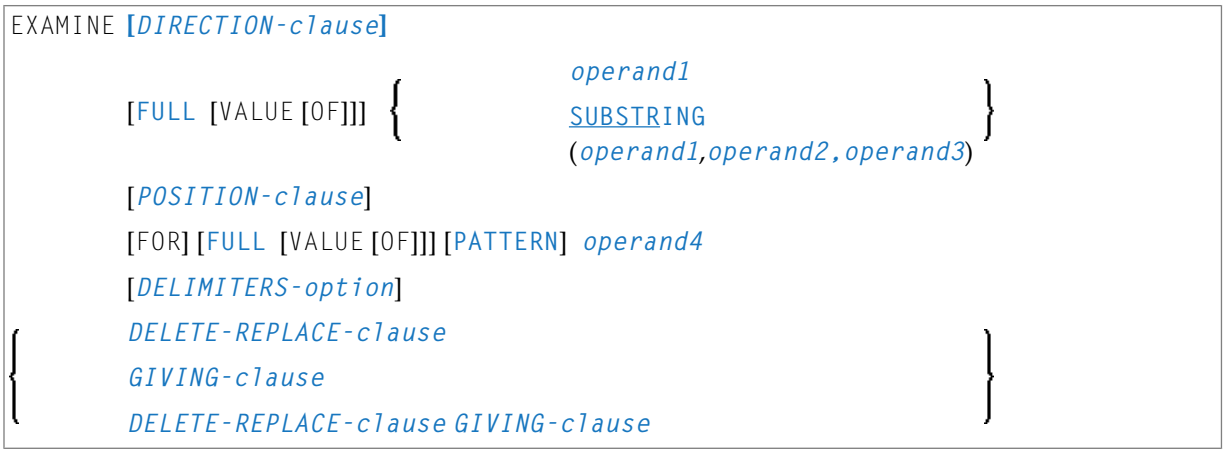
- Syntax 1 - EXAMINE 444
- Syntax 2 - EXAMINE TRANSLATE 453
- Syntax 3 - EXAMINE für Unicode-Grapheme 455
- Beispiele EXAMINE 457

Dieser Abschnitt 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



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 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 gegen eine andere Zeichenkette auszutauschen;
- eine Zeichenkette zu löschen.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C*	S	A			A	U				B							ja	nein
operand2	C	S						N	P	I	B*							ja	nein
operand3	C	S						N	P	I	B*							ja	nein
operand4	C	S				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.

* *operand4* kann auch ein Array sei, siehe ***Suchen und Ersetzen mit mehreren Werten***.

* Format B von *operand2* und *operand3* kann nur mit einer Länge von kleiner gleich 4 verwendet werden.

Syntax-Element-Beschreibung:

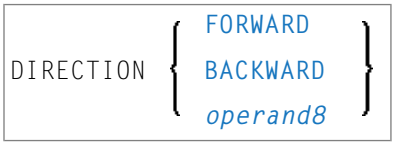
Syntax-Element	Beschreibung
DIRECTION-clause	**DIRECTION-Klausel:** Mit dieser Klausel legen Sie die Suchrichtung fest. Weitere Informationen siehe ***DIRECTION-Klausel*** weiter unten.
operand1	**Zu untersuchendes Feld:** *operand1* ist das Feld, dessen Inhalt untersucht werden soll. Ist *operand1* 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.
POSITION-clause	**POSITION-Klausel:** Mit dieser Klausel können Sie für die Untersuchung eine Start- und Endeposition innerhalb von *operand1* (oder dem Substring von *operand1*) angeben. Weitere Informationen siehe ***POSITION-Klausel*** weiter unten.
operand4	**Für die EXAMINE-Operation zu benutzender Wert:** *operand4* ist der Wert, nach dem in dem untersuchten Feld oder Feldern gesucht wird. Sie können nach einem einzelnen Wert oder nach mehreren Werten suchen. Weitere Informationen zu *operand4* und *operand6* siehe *operand6*, der in der unten beschriebenen ***DELETE REPLACE-Klausel*** benutzt wird.
FULL	**FULL-Option:**

Syntax-Element	Beschreibung
	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.
SUBSTRING	**SUBSTRING-Option:** Normalerweise wird der ganze Inhalt des Feldes untersucht, und zwar vom Anfang des Feldes bis zum Ende bzw. bis zum letzten Zeichen, das kein Leerzeichen ist. Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes zu untersuchen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (Startposition, *operand2*) und dann die Länge (*operand3*) des Feldteils an, der untersucht werden soll. Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A zu untersuchen, geben Sie folgendes an: `EXAMINE SUBSTRING(#A,5,8).` **Anmerkung:** 1. Wenn Sie *operand2* weglassen, wird ab Anfang des Feldes untersucht. 2. Wenn Sie *operand3* weglassen, wird ab der angegebenen Stelle (*operand2*) 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 (Systemvariable *LENGTH) ändert sich nicht als Ergebnis der EXAMINE-Operation, ungeachtet der Tatsache, ob eine DELETE- oder REPLACE-Operation ausgeführt wurde oder nicht.
PATTERN	**PATTERN-Option:** 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. *operand4* kann dann die folgenden Platzhalter für nicht zu untersuchende Stellen enthalten: ■ 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. Beispiel: Mit PATTERN 'NAT*AL' könnten Sie ein Feld nach jedem Wert absuchen, in dem NAT und AL vorkommt, 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).

Syntax-Element	Beschreibung
DELIMITERS-option	**DELIMITERS-Option:** Diese Option wird zum Suchen eines Wertes benutzt, der Delimiter aufweist. Einzelheiten siehe ***DELIMITERS-Option*** weiter unten.
DELETE-REPLACE-clause	**DELETE REPLACE-Klausel:** Die DELETE-Option dieser Klausel wird zum Löschen jedes Suchwertes (*operand4*) benutzt, der in *operand1* gefunden wird. Die REPLACE-Option wird zum Ersetzen jedes in *operand1* gefundenen Suchwertes (*operand4*) durch den in *operand6* angegebenen Wert benutzt. Siehe ***DELETE REPLACE-Klausel*** weiter unten.
GIVING-clause	**GIVING-Klausel:** Siehe ***GIVING-Klausel*** weiter unten.

DIRECTION-Klausel

Diese Klausel bestimmt die Suchrichtung.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand8	C	S				A1										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FORWARD	**Suchrichtung von links nach rechts:** Wenn Sie FORWARD angeben, wird der Feldinhalt von links nach rechts untersucht.
BACKWARD	**Suchrichtung von rechts nach links:** Wenn Sie BACKWARD angeben, wird der Feldinhalt von rechts nach links untersucht.
operand8	**Alternative Angabe:** Wenn Sie *operand8* angeben, wird die Suchrichtung durch den Inhalt von *operand8* bestimmt. *operand8* muss mit Format/Länge A1 definiert werden. Wenn *operand8* ein "F" enthält, ist die Suchrichtung FORWARD; wenn *operand8* ein "B" enthält, dann ist die

Syntax-Element	Beschreibung
	Suchrichtung BACKWARD. Alle anderen Werte sind ungültig und werden zurückgewiesen: wenn *operand8* eine Konstante ist, wird der Wert zur Kompilierzeit zurückgewiesen; wenn *operand8* eine Variable ist, wird der Wert zur Laufzeit zurückgewiesen.

Anmerkung: Wenn die DIRECTION-Klausel nicht angegeben ist, wird die Standardsuchrichtung FORWARD benutzt.

POSITION-Klausel

Mit dieser Klausel können Sie für die Untersuchung eine Start- und Endeposition innerhalb von *operand1* (oder dem Substring von *operand1*) angeben.

```
[[STARTING] FROM [POSITION] operand9] [ { ENDING AT } [POSITION] operand10 ]
                                      { THRU      }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand9	C	S				N	P	I								ja	nein
operand10	C	S				N	P	I								ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FROM*operand9*	**Startposition:** *operand9* definiert die Startposition für die Untersuchung.
ENDING AT / THRU *operand10*	**Endeposition:** *operand10* 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 duch die Suchrichtung bestimmt:

Surchrichtung	Standardstartposition	Standardendeposition
FORWARD	1 (erstes Zeichen)	Länge von *operand1* (letztes Zeichen)
BACKWARD	Länge von *operand1* (letztes Zeichen)	1 (erstes Zeichen)

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
operand5	C	S				A	U				B						ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ABSOLUTE	**Absolute Suche:** 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.
WITH DELIMITERS	**Suchwert mit beliebigen Begrenzungszeichen:** Bei Angabe der Option 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. Die Definition des Begrenzungszeichens kann mit dem Profilparameter SCTAB geändert werden.
WITH DELIMITERS *operand5*	**Suchwert mit bestimmten Begrenzungszeichen:** Mit WITH DELIMITERS *operand5* wird ein Wert gesucht, der von dem oder von einem in *operand5* angegebenen Zeichen eingegrenzt ist. Wenn der Suchwert am Anfang bzw. Ende des untersuchten Feldes gefunden wurde, muss nur die rechte bzw. linke Seite durch eines der in *operand5* angegebenen Zeichen begrenzt werden.

DELETE/REPLACE-Klausel

```
[AND] { DELETE [FIRST]
      { REPLACE [FIRST] [WITH] [FULL [VALUE [OF]]] operand6
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand6	C	S	A*			A	U				B					ja	nein

* *operand6* kann auch ein Array sein, siehe ***Suchen und Ersetzen mit mehreren Werten***.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
DELETE	**Lösch-Option:** Dient zum Löschen der ersten (oder aller) Ausprägung(en) des Suchwertes (*operand4*) im Inhalt von *operand1*.
REPLACE	**Austausch-Option:** Wird benutzt zum Ersetzen der ersten (oder aller) Ausprägung(en) des Suchwertes (*operand4*) in *operand1* durch den in *operand6* angegebenen Austauschwert.
FIRST	**Löschen/Austauschen des ersten identischen Werts:** Wenn Sie das Schlüsselwort FIRST angeben, wird nur der erste identische Wert gelöscht/ersetzt.

Anmerkungen:

1. Wenn die REPLACE-Operation zur Generierung von mehr Zeichen führt, als in den *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*.

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 Übertragungs-

kompatibilität 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 bei 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* überschreibt, 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:

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
```

Siehe auch ***Beispiel 3 - EXAMINE und REPLACE mit mehreren Werten***.

GIVING-Klausel

```
[     {     GIVING [IN] operand7          }     ]
      {     [GIVING] NUMBER [IN] operand7 }
[[GIVING] POSITION [IN] operand7]
[[GIVING] LENGTH [IN] operand7]
[[GIVING] INDEX [IN] operand7 ...3]
```

Operanden-Definitionstabelle:

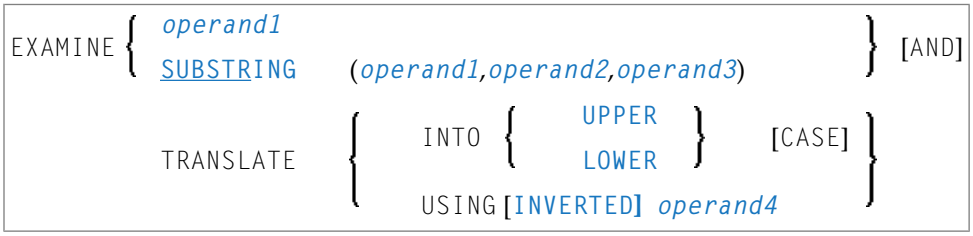
Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand7		S					N	P	I							ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
GIVING	**GIVING-Klausel:** Wenn nur das Schlüsselwort GIVING angegeben wird, entspricht dies GIVING NUMBER (Voreinstellung).
NUMBER	**GIVING NUMBER-Klausel:** Wird benutzt, um die Zahl zu erhalten, wie oft der zu suchende Wert (*operand4*) in dem Feld (*operand1*) gefunden wird, dessen Inhalt überprüft werden soll.
POSITION	**GIVING POSITION-Klausel:** Mit GIVING POSITION erhalten Sie die Byte-Position, die der erste gefundene Wert (*operand4*) innerhalb von *operand1* (bzw. des Substrings von *operand1*) innehat.
LENGTH	**GIVING LENGTH-Klausel:** Mit GIVING LENGTH erhalten Sie die Länge des Inhalts von *operand1* (bzw. des Substrings von *operand1*), nachdem alle DELETE- bzw. REPLACE-Operationen abgeschlossen sind. Nachfolgende Leerzeichen werden ignoriert.
operand7	**Anzahl der Ausprägungen:** 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 *operand7* ...3	**GIVING INDEX-Klausel:** This option is only applicable if the underlying field to be examined is an array field. GIVING INDEX wird benutzt, um die Zahl der Array-Ausprägung (Index) von *operand1* zu erhalten, in welcher der erste Suchwert gefunden wurde.

Syntax-Element	Beschreibung
	operand7 muss so oft angegeben werden, wie Dimensionen in *operand1* vorhanden sind (maximal dreimal). *operand7* gibt 0 zurück, falls der Suchwert in keiner der Ausprägungen gefunden wird. **Anmerkung:** Falls der Indexbereich von *operand1* die Ausprägung 0 (zum Beispiel 0:5) enthält, dann ist ein Wert 0 in *operand7* nicht eindeutig. In diesem Fall sollten Sie eine zusätzliche GIVING NUMBER-Klausel angeben, um sich zu vergewissern, ob der Suchwert tatsächlich gefunden wurde oder nicht.

Syntax 2 - EXAMINE TRANSLATE



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
operand1		S	A			A				B						ja	nein
operand2	C	S					N	P	I	B*						ja	nein
operand3	C	S					N	P	I	B*						ja	nein
operand4		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:

Syntax-Element	Beschreibung
`EXAMINE operand1`	**Umsetzung des kompletten Feldinhalts:** *operand1* ist das Feld, dessen Inhalt umgesetzt werden soll.
`EXAMINE SUBSTRING operand1 operand2 operand3`	**Umsetzung von Teilen des Feldinhalts:** Normalerweise wird der Inhalt des gesamten Feldes umgesetzt. Die Option `SUBSTRING` ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes umzusetzen. In der `SUBSTRING`-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) des Feldteils, der umgesetzt werden soll, an. Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes `#A` umzusetzen, geben Sie folgendes an: `EXAMINE SUBSTRING(#A,5,8) AND TRANSLATE ...` **Anmerkung:** Wenn Sie *operand2* weglassen, wird ab Anfang des Feldes umgesetzt. Wenn Sie *operand3* weglassen, wird ab der ersten Stelle bis zum Ende des Feldes umgesetzt.
`TRANSLATE INTO UPPER CASE`	**Umsetzung in Großbuchstaben:** Der Inhalt von *operand1* wird in Großbuchstaben umgesetzt.
`TRANSLATE INTO LOWER CASE`	**Umsetzung in Kleinbuchstaben:** Der Inhalt von *operand1* wird in Kleinbuchstaben umgesetzt.
`TRANSLATE USING operand4`	**Zu benutzende Umsetzungstabelle:** *operand4* ist die Umsetzungstabelle, die für die Zeichenumsetzung verwendet werden soll. Die Tabelle muss Format/Länge A2 oder B2 haben. **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 `INVERTED` angeben, wird die Umsetzungstabelle (*operand4*) in umgekehrter Richtung verwendet, d.h. die Umsetzungsrichtung wird umgekehrt.

Syntax 3 - EXAMINE für Unicode-Grapheme

```
EXAMINE [FULL [VALUE [OF]]]  { operand1
                             { SUBSTRING(operand1,operand2,operand3) }
[POSITION-clause]
[FOR]                        { CHARPOSITION operand4 CHARLENGTH operand5
                             { CHARPOSITION operand4
                             { CHARLENGTH operand5 }
[GIVING] POSITION IN operand6 [[GIVING] LENGTH IN operand7]
```

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- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*, Unterabschnitt *EXAMINE* in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

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	C	S					N	P	I							ja	nein
operand5	C	S					N	P	I							ja	nein

Operand	Mögliche Struktur						Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand6	C	S					N	P	I								ja	nein
operand7	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:

Syntax-Element	Beschreibung
FULL	**FULL-Option:** 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.
SUBSTRING *operand1 operand2 operand3*	**SUBSTRING-Option:** Normalerweise wird der ganze Inhalt des Feldes untersucht, und zwar vom Anfang des Feldes bis zum Ende bzw. bis zum letzten signifikanten Zeichen. Die Option SUBSTRING ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes zu untersuchen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) des Feldteils an, der untersucht werden soll. *operand2* und *operand3* werden als Code-Einheiten angegeben. Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A zu untersuchen, geben Sie folgendes an: `EXAMINE SUBSTRING (#A,5,8)` **Anmerkung:** 1. Wenn Sie *operand2* weglassen, wird ab Anfang des Feldes (Position 1) untersucht. 2. Wenn Sie *operand3* 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 (Systemvariable *LENGTH) ändert sich nicht als Ergebnis der EXAMINE-Operation, ungeachtet der Tatsache, ob eine DELETE- oder REPLACE-Operation ausgeführt wurde oder nicht.
POSITION-clause	**POSITION-Klausel:** Bereichswerte für FROM und THRU werden in Form von Code-Einheiten angegeben. Weitere Informationen siehe ***POSITION-Klausel*** unter Syntax 1.

Syntax-Element	Beschreibung
`CHARPOSITION` *operand4*	**CHARPOSITION-Klausel:** *operand4* legt die Startposition (als Unicode-Graphem) der Graphem-Sequenz fest. Die entsprechende Position wird in *operand6* 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` *operand5*	**CHARLENGTH-Klausel:** *operand5* legt die Länge der Graphem-Sequenz (als Unicode-Graphem) fest. Die Länge der Graphem-Sequenz wird in Form von Code-Einheiten in *operand7* 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` *operand6*	**GIVING POSITION-Klausel:** *operand6* enthält (als Code-Einheiten) die Startposition der von *operand4* und *operand5* definierten Graphem-Sequenz. Wenn *operand1* weniger als *operand4* Grapheme hat, wird Null (0) zurückgegeben. Diese Klausel kann weggelassen werden, wenn die `GIVING LENGTH`-Klausel angegeben wird.
`GIVING LENGTH IN` *operand7*	**GIVING LENGTH-Klausel:** *operand7* enthält (als Code-Einheiten) die Länge der mit *operand4* und *operand5* definierten Graphem-Sequenz. Wenn *operand1* weniger als *operand4*+*operand5* 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 EXAMINE

- Beispiel 1 - EXAMINE
- Beispiel 2 - EXAMINE SUBSTRING, PATTERN, TRANSLATE
- Beispiel 3 - EXAMINE und REPLACE mit mehreren Werten

- Beispiel 4 - EXAMINE für Unicode-Grapheme

Beispiel 1 - EXAMINE

```
** Example 'EXMEX1': EXAMINE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT   (A45)
1 #ARRAY  (A5/1:3)
1 #A      (A3)
1 #START  (N2)
1 #NUM    (N2)
1 #NUM1   (N2)
1 #NUM2   (N2)
1 #NUM3   (N2)
1 #POS    (N2)
1 #POS1   (N2)
1 #LENG   (N2)
1 #INDEX  (N2)
END-DEFINE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
WRITE / 'EXAMPLE 1  (DELIMITER, GIVING NUMBER)'
WRITE NOTITLE '#TEXT: ' #TEXT
EXAMINE #TEXT FOR 'A' GIVING NUMBER #NUM1
EXAMINE #TEXT FOR 'A' WITH DELIMITER GIVING NUMBER #NUM2
EXAMINE #TEXT FOR 'A' WITH DELIMITER '.' GIVING NUMBER #NUM3
WRITE 'EXAMINE #TEXT FOR "A" ' 57T 'Number found:' #NUM1
WRITE 'EXAMINE #TEXT FOR "A" WITH DELIMITER' 57T 'Number found:' #NUM2
WRITE 'EXAMINE #TEXT FOR "A" WITH DELIMITER "."'
  57T 'Number found:' #NUM3
*
WRITE / 'EXAMPLE 2  (DELIMITER, REPLACE, GIVING NUMBER)'
WRITE 'EXAMINE #TEXT FOR "A" WITH DELIMITER "-" REPLACE WITH "*"'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR 'A' WITH DELIMITER '-' REPLACE WITH '*'
        GIVING NUMBER #NUM
WRITE 'After: ' #TEXT 57T 'Number found:' #NUM
*
*
NEWPAGE
*
WRITE / 'EXAMPLE 3  (REPLACE, GIVING NUMBER)'
WRITE 'EXAMINE      #TEXT FOR " " REPLACE WITH "+"'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR ' ' REPLACE WITH '+' GIVING NUMBER #NUM
WRITE 'After: ' #TEXT 57T 'Number found:' #NUM
*
WRITE / 'EXAMPLE 4  (FULL, REPLACE, GIVING NUMBER)'
WRITE 'EXAMINE FULL #TEXT FOR " " REPLACE WITH "+"'
```

```
WRITE 'Before:' #TEXT
EXAMINE FULL #TEXT FOR ' ' REPLACE WITH '+' GIVING NUMBER #NUM
WRITE 'After: ' #TEXT 57T 'Number found:' #NUM
*
WRITE / 'EXAMPLE 5  (DELETE, GIVING POSITION)'
WRITE 'EXAMINE #TEXT FOR "+" DELETE GIVING POSITION #POS'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR '+' DELETE GIVING POSITION #POS
WRITE 'After: ' #TEXT 57T 'Position found:' #POS
*
WRITE / 'EXAMPLE 6  (DELETE, GIVING LENGTH)'
WRITE 'EXAMINE #TEXT FOR "A" DELETE GIVING LENGTH #LENG'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR 'A' DELETE GIVING LENGTH #LENG
WRITE 'After: ' #TEXT 57T 'Length found:' #LENG
*
*
NEWPAGE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
WRITE / 'EXAMPLE 7  (PATTERN, REPLACE, GIVING NUMBER)'
WRITE 'EXAMINE #TEXT FOR         ".A." AND REPLACE "***"'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR         '.A.' AND REPLACE '***' GIVING NUMBER #NUM
WRITE 'After: ' #TEXT 57T 'Number found:' #NUM
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
WRITE 'EXAMINE #TEXT FOR PATTERN ".A." AND REPLACE "***"'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR PATTERN '.A.' AND REPLACE '***' GIVING NUMBER #NUM
WRITE 'After: ' #TEXT 57T 'Number found:' #NUM
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
*
#A   := 'B C'
#POS := 6
#LENG:= 25
*
WRITE / 'EXAMPLE 8  (SUBSTRING, REPLACE, GIVING POSITION)'
WRITE '#A := "B C" ; #POS := 6 ; #LENG:= 25 '
WRITE 'EXAMINE SUBSTRING(#TEXT,#POS,#LENG) FOR #A AND REPLACE "***"'
WRITE 'Before:' #TEXT
EXAMINE SUBSTRING(#TEXT,#POS,#LENG) FOR #A AND REPLACE '***'
        GIVING POSITION #POS1
WRITE 'After: ' #TEXT 57T 'Position found:' #POS1
*
*
NEWPAGE
*
MOVE 'ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C- ' TO #TEXT
```

```
*
WRITE / 'EXAMPLE 9  (DELETE, GIVING NUMBER, GIVING POSITION, '-
        'GIVING LENGTH)'
WRITE 'EXAMINE #TEXT FOR "." DELETE GIVING NUMBER   #NUM'
WRITE 30T 'GIVING POSITION #POS'
WRITE 30T 'GIVING LENGTH   #LENG'
WRITE 'Before:' #TEXT
EXAMINE #TEXT FOR '.' DELETE GIVING NUMBER #NUM
                           GIVING POSITION #POS
                           GIVING LENGTH   #LENG
WRITE 'After: ' #TEXT
WRITE 'Number found:  ' #NUM
WRITE 'Position found:' #POS
WRITE 'Length found:  ' #LENG
*
*
*
MOVE 'ABC  ' TO #ARRAY (1)
MOVE '.A.B.' TO #ARRAY (2)
MOVE '-A-B-' TO #ARRAY (3)
*
WRITE / 'EXAMPLE 10 (GIVING NUMBER, GIVING POSITION, GIVING INDEX)'
WRITE '#ARRAY(1):' #ARRAY(1)
WRITE '#ARRAY(2):' #ARRAY(2)
WRITE '#ARRAY(3):' #ARRAY(3)
WRITE 'EXAMINE #ARRAY(*) FOR "B" GIVING NUMBER   #NUM'
WRITE 27T 'GIVING POSITION #POS'
WRITE 27T 'GIVING INDEX    #INDEX'
EXAMINE #ARRAY(*) FOR 'B' GIVING NUMBER   #NUM
                          GIVING POSITION #POS
                          GIVING INDEX    #INDEX
WRITE 'Number found:  ' #NUM
WRITE 'Position found:' #POS
WRITE 'Index found:   ' #INDEX
END ↩
```

Ausgabe des Programms `EXMEX1`:

```
EXAMPLE 1  (DELIMITER, GIVING NUMBER)
#TEXT:  ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
EXAMINE #TEXT FOR 'A'                                Number found:   4
EXAMINE #TEXT FOR 'A' WITH DELIMITER                 Number found:   3
EXAMINE #TEXT FOR 'A' WITH DELIMITER '.'             Number found:   1

EXAMPLE 2  (DELIMITER, REPLACE, GIVING NUMBER)
EXAMINE #TEXT FOR 'A' WITH DELIMITER '-' REPLACE WITH '*'
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC   A B C   .A.  .B.  .C.    -*-  -B-  -C-    Number found:   1

EXAMPLE 3  (REPLACE, GIVING NUMBER)
EXAMINE      #TEXT FOR ' ' REPLACE WITH '+'
Before: ABC   A B C   .A.  .B.  .C.    -*-  -B-  -C-
```

```
After:  ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-++-C-    Number found:  20

EXAMPLE 4  (FULL, REPLACE, GIVING NUMBER)
EXAMINE FULL #TEXT FOR ' ' REPLACE WITH '+'
Before: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-++-C-
After:  ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-++-C-+   Number found:   1

EXAMPLE 5  (DELETE, GIVING POSITION)
EXAMINE #TEXT FOR '+' DELETE GIVING POSITION #POS
Before: ABC+++A+B+C+++.A.++.B.++.C.++++-*-++-B-++-C-+
After:  ABCABC.A..B..C.-*--B--C-                         Position found:   4

EXAMPLE 6  (DELETE, GIVING LENGTH)
EXAMINE #TEXT FOR 'A' DELETE GIVING LENGTH #LENG
Before: ABCABC.A..B..C.-*--B--C-
After:  BCBC...B..C.-*--B--C-                            Length found:  21

EXAMPLE 7  (PATTERN, REPLACE, GIVING NUMBER)
EXAMINE #TEXT FOR         '.A.' AND REPLACE '***'
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC   A B C   ***  .B.  .C.    -A-  -B-  -C-    Number found:   1
EXAMINE #TEXT FOR PATTERN '.A.' AND REPLACE '***'
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC  ***B C   ***  .B.  .C.    ***  -B-  -C-    Number found:   3

EXAMPLE 8  (SUBSTRING, REPLACE, GIVING POSITION)
#A := 'B C' ; #POS := 6 ; #LENG:= 25
EXAMINE SUBSTRING(#TEXT,#POS,#LENG) FOR #A AND REPLACE '***'
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC   A ***   .A.  .B.  .C.    -A-  -B-  -C-    Position found:   4

EXAMPLE 9  (DELETE, GIVING NUMBER, GIVING POSITION, GIVING LENGTH)
EXAMINE #TEXT FOR '.' DELETE GIVING NUMBER   #NUM
                             GIVING POSITION #POS
                             GIVING LENGTH   #LENG
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC   A B C   A  B  C    -A-  -B-  -C-
Number found:     6
Position found:  15
Length found:    38

EXAMPLE 10 (GIVING NUMBER, GIVING POSITION, GIVING INDEX)
#ARRAY(1): ABC
#ARRAY(2): .A.B.
#ARRAY(3): -A-B-
EXAMINE #ARRAY(*) FOR 'B' GIVING NUMBER   #NUM
                          GIVING POSITION #POS
                          GIVING INDEX    #INDEX
Number found:     3
Position found:   2
Index found:      1    ↩
```

Beispiel 2 - EXAMINE SUBSTRING, PATTERN, TRANSLATE

```
** Example 'EXMEX2': EXAMINE TRANSLATE
************************************************************************
DEFINE DATA LOCAL
1 #TEXT  (A50)
1 #TAB   (A2/1:10)
1 #POS   (N2)
1 #LENG  (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 NOTITLE / 'EXAMPLE 1  (WITH TRANSLATION TABLE)'
WRITE 'EXAMINE #TEXT TRANSLATE USING #TAB(*)'
WRITE 'Before:' #TEXT
EXAMINE #TEXT TRANSLATE USING #TAB(*)
WRITE 'After: ' #TEXT
*
WRITE / 'EXAMPLE 2  (WITH INVERTED TRANSLATION TABLE)'
WRITE 'EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)'
WRITE 'Before:' #TEXT
```

```
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
WRITE 'After: ' #TEXT                                                          ↩
*                                                                              ↩
#POS := 13                                                                     ↩
#LENG:= 15                                                                     ↩
*                                                                              ↩
WRITE / 'EXAMPLE 3  (WITH LOWER CASE TRANSLATION)'                             ↩
WRITE '#POS := 13 ; #LENG:= 15 '                                               ↩
WRITE 'EXAMINE SUBSTRING(#TEXT,#POS,#LENG) TRANSLATE INTO LOWER CASE'          ↩
WRITE 'Before:' #TEXT                                                          ↩
EXAMINE SUBSTRING(#TEXT,#POS,#LENG) TRANSLATE INTO LOWER CASE
WRITE 'After: ' #TEXT                                                          ↩
*                                                                              ↩
END
```

Ausgabe des Programms `EXMEX2`:

```
EXAMPLE 1  (WITH TRANSLATION TABLE)
EXAMINE #TEXT TRANSLATE USING #TAB(*)
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-

EXAMPLE 2  (WITH INVERTED TRANSLATION TABLE)
EXAMINE #TEXT TRANSLATE USING INVERTED #TAB(*)
Before: XYZ   X Y Z   .X.  .Y.  .Z.    -X-  -Y-  -Z-
After:  ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-

EXAMPLE 3  (WITH LOWER CASE TRANSLATION)
#POS := 13 ; #LENG:= 15
EXAMINE SUBSTRING(#TEXT,#POS,#LENG) TRANSLATE INTO LOWER CASE
Before: ABC   A B C   .A.  .B.  .C.    -A-  -B-  -C-
After:  ABC   A B C   .a.  .b.  .c.    -A-  -B-  -C-          ↩
```

Beispiel 3 - EXAMINE und REPLACE mit mehreren Werten

```
*  EXAMPLE 'EXMEX3': EXAMINE AND REPLACE WITH MULTIPLE VALUES
************************************************************************
* This example shows a translation of the pattern
* 'AA', 'Aa' and 'aA' into '++',
* 'BB', 'Bb' and 'bB' into '--' and
* 'CC', 'Cc' and 'cC' into '**'.
************************************************************************
DEFINE DATA LOCAL
1 #SV     (A2/1:3,1:3) INIT (1,V) <'AA','BB','CC'>
                           (2,V) <'Aa','Bb','Cc'>
                           (3,V) <'aA','bB','cC'>
1 #RV     (A2/1:3)     INIT       <'++','--','**'>
1 #STRING (A20)        INIT <'AAABbbbbBCCCcccCaaaA'>
1 #NUM    (N2)
END-DEFINE
*
*
WRITE NOTITLE / 'EXAMINE #STRING FOR #SV(*,*) AND REPLACE WITH #RV(*)' /
*
WRITE 'Before:' #STRING   /* shows   'AAABbbbbBCCCcccCaaaA'
*
EXAMINE #STRING FOR #SV(*,*) AND REPLACE WITH #RV(*)
        GIVING NUMBER #NUM
*
WRITE 'After: ' #STRING   /* shows   '++A--bb--****c**aa++'
  4OT 'Number found:' #NUM
*
```

Ausgabe des Programms `EXMEX3`:

```
EXAMINE #STRING FOR #SV(*,*) AND REPLACE WITH #RV(*)

Before: AAABbbbbBCCCcccCaaaA
After:  ++A--bb--****c**aa++           Number found:   7 ↩
```

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=O) 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:

z/OS-Umgebungen:	**Windows- und Linux-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.	

z/OS-Umgebungen:	Windows- und Linux-Umgebungen (mit Natural Web I/O Interface):
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.

63 EXPAND

- Funktion EXPAND 468
- Syntax-Beschreibung EXPAND 468

```
EXPAND { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

Dieser Abschnitt 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 EXPAND

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 EXPAND

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S	A			A	U					B						nein	nein
operand2	C	S								I								nein	nein
operand3			A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand4	C	S						N	P	I									nein	nein
operand5		S								I4									nein	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
dynamic-clause	**DYNAMIC-Klausel:** 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. Siehe ***DYNAMIC-Klausel*** weiter unten.
operand1	**Zu erweiternde Variable:** *operand1* ist die dynamische Variable, für die die zugewiesene Länge erweitert werden soll.
operand2	**Erweiterungswert:** *operand2* 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.
array-clause	**Array-Klausel:** Mit dem Statement `EXPAND ARRAY` können Sie Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit `(dim[,dim[,dim]])` angegebene Ober- und Untergrenze erweitern. Siehe ***Array-Klausel*** weiter unten.
operand3	**Zu erweiterndes X-Array:** *operand3* 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.
`dim` *operand4*	**Ober-/Untergrenze der Erweiterung:** Die Notation für die Ober- und Untergrenze (*operand4* 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 *operand4* angegeben werden. Siehe ***Dimension*** weiter unten.
`GIVING` *operand5*	**GIVING-Klausel** Wenn diese Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die Klausel angegeben wird, enthält *operand5* 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 `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 größenmäßig angepassten X-Arrays auf ihren standardmäßigen Nullwert zurück. Als Voreinstellung (keine `RESET`-Option) werden die Direktwerte beibehalten und die erweiterten (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:

```
{ operand4 } : { operand4 }
{ *        }   { *        }
```

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.

VII

- 64 FETCH 475
- 65 FIND 481
- 66 FOR 523
- 67 FORMAT 529
- 68 GET 535
- 69 GET SAME 541
- 70 GET TRANSACTION DATA 545
- 71 HISTOGRAM 549
- 72 IF 563
- 73 IF SELECTION 567
- 74 IGNORE 571
- 75 INCLUDE 573

64 FETCH

- Funktion FETCH 476
- Syntax-Beschreibung FETCH 477
- Beispiel für FETCH-Statement 478

```
FETCH [ { REPEAT } ] operand1 [operand2 [(parameter)]] ...
      [ { RETURN } ]
```

Dieser Abschnitt 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 Subprogrammen***

Funktion FETCH

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 RPC (Remote Procedure Call)*-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 (*Datenbank-Open/Close-Befehlsverarbeitung*) 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 FETCH

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
REPEAT	**Wegfall der Notwendigkeit einer Benutzerinteraktion:** 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.
RETURN	**Aufrufen und Ausführen eines Objekts des Typs Programm als Routine:** 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. Geben Sie FETCH RETURN an, wird das aufrufende Programm nicht beendet, sondern nur unterbrochen, während das aufgerufene Programm als Subprogramm 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.
operand1	**Programm-Name:** 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. Die Groß-/Kleinschreibung des Namens wird nicht verändert. 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. 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.

Syntax-Element	Beschreibung
operand2	**Zu übergebende Parameter-Felder:** Mit dem FETCH-Statement können auch Parameterfelder an das aufgerufene Programm übergeben werden. Ein Parameterfeld kann mit einem beliebigen Format definiert werden; das Format wird dem jeweiligen INPUT-Feld entsprechend umgesetzt. Sämtliche Parameter werden oben auf dem Natural-Stack abgelegt. Das aufgerufene Programm liest die übergebenen Parameterfelder über ein INPUT-Statement. Das erste INPUT-Statement bewirkt, dass die Werte aller Parameterfelder in die mit dem INPUT-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 INPUT-Statement der Session-Parameter SG für die Parameterfelder auf SG=ON gesetzt werden. Werden mehr Parameter übergeben als vom INPUT-Statement gelesen werden können, so werden überschüssige Parameter ignoriert. Die Anzahl der Parameter kann mit Hilfe der Natural-Systemvariablen *DATA ermittelt werden. **Anmerkung:** Wenn *operand2* eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.
parameter	**Datumsformat für Datumsvariable:** Wenn *operand2* eine Datumsvariable ist, können Sie den Session-Parameter DF (Date Format) als *parameter* für diese Variable angeben.

Beispiel für FETCH-Statement

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
```

65 FIND

- Funktion FIND 482
- Einschränkungen FIND 483
- Syntax 1 - FIND-Statement mit Verarbeitungsschleife 484
- Syntax 2 - FIND-Statement ohne Verarbeitungsschleife 484
- Syntax-Beschreibung FIND 485
- Beispiele FIND 510

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 FIND

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 folgende Abschnitte im *Leitfaden zur Programmierung*.

- *FIND-Statement*
- *Schleifenverarbeitung*
- *Datenbankfelder mit der (r)-Notation referenzieren*

Datenbank-spezifische Anmerkungen

Datenbank	Erläuterung
VSAM	Das FIND-Statement ist nur auf VSAM-KSDS (Key-Sequenced Datasets) und -ESDS (Entry-Sequenced Datasets) anwendbar. Bei ESDS muss ein Alternativ-Index für den Basis-Cluster definiert werden.

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 Quellcode-Zeilennummer referenziert werden. Alle drei Systemvariablen haben Format/Länge P10; diese(s) Format/Länge kann nicht geändert werden.

Systemvariable	Erläuterung
*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. ■ **VSAM** Siehe **ISN für VSAM* in der *Systemvariablen*-Dokumentation. ■ **SQL** *ISN ist nicht verfügbar. ■ **Entire System Server** *ISN ist nicht verfügbar. ■ **Natural Messaging** *ISN ist nicht verfügbar.
*NUMBER	Siehe *NUMBER in der *Systemvariablen*-Dokumentation. Bei Entire System Server und Natural Messaging ist *NUMBER nicht verfügbar.
*COUNTER	Der in *COUNTER enthaltene Wert ist die Anzahl, wie oft die Verarbeitungsschleife durchlaufen worden ist.

Siehe auch ***Beispiel 13 - Sytemvariablen 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 FIND

Mit Entire System Server und Natural Messaging sind FIND NUMBER und FIND UNIQUE sowie die Klauseln PASSWORD, CIPHER, COUPLED und RETAIN nicht zulässig.

Syntax 1 - FIND-Statement mit Verarbeitungsschleife

```
FIND      [ { ALL          } ]   [MULTI-FETCH-clause] [RECORDS] [IN] [FILE] view-name
          [ { (operand1)   } ]
          [PASSWORD=operand2]
          [CIPHER=operand3]
          [WITH] [[LIMIT] (operand4)] basic-search-criteria
          [COUPLED-clause] ... 4/42
          [STARTING WITH ISN=operand5]
          [SORTED-BY-clause]
          [RETAIN-clause]
          [[IN] SHARED HOLD [MODE=option]]
          [SKIP [RECORDS] IN HOLD]
          [WHERE-clause]
          [IF-NO-RECORDS-FOUND-clause]
          statement ...
END-FIND  [(r)]                          (structured mode only)
LOOP      [(r)]                          (reporting mode only)
```

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

Syntax 2 - FIND-Statement ohne Verarbeitungsschleife

```
FIND [ { FIRST  } ]   [RECORDS] [IN] [FILE] view-name
     [ { NUMBER } ]
     [ { UNIQUE } ]
     [PASSWORD=operand2]
     [CIPHER=operand3]
     [WITH] [[LIMIT] (operand4)] basic-search-criteria
     [COUPLED-clause] ... 4/42
     [SORTED-BY-clause]    (only for FIND FIRST)
     [RETAIN-clause]
     [WHERE-clause]
```

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

Syntax-Beschreibung FIND

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S					N	P	I		B*						ja	nein
operand2	C	S				A											ja	nein
operand3	C	S					N										ja	nein
operand4	C	S					N	P	I		B*						ja	nein
operand5	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:

Syntax-Element	Beschreibung

ALL/*operand1*	**Begrenzung der Verarbeitung:**
	
	Sie können die Anzahl der ausgewählten Datensätze, die in der FIND-Schleife verarbeitet werden sollen, durch Angabe von *operand1* (in Klammern direkt hinter dem Schlüsselwort READ) entweder als numerische Konstante (im Bereich von 0 bis 4294967295) oder als den Namen einer numerischen Benutzervariable begrenzen.
	
	Andernfalls werden alle gefundenen Sätze weiterverarbeitet, was Sie zusätzlich durch das Schlüsselwort ALL hervorheben können.
	
	Geben Sie mit *operand1* 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.
	
	`FIND (5) IN EMPLOYEES WITH ...`
	
	`MOVE 10 TO #CNT(N2)`
	`FIND (#CNT) EMPLOYEES WITH ...`
	
	Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.

Syntax-Element	Beschreibung
	Ist mit dem LT-Parameter ein kleineres Limit gesetzt, so gilt das LT-Limit. **Anmerkung:** 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. *operand1* hat keinen Einfluss auf die Größe eines ISN-Set, der mittels einer RETAIN-Klausel erzeugt wird. *operand1* wird zu Beginn des ersten FIND-Schleifendurchlaufs ausgewertet. Wird der Wert von *operand1* innerhalb der FIND-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der verarbeiteten Datensätze.
FIND FIRST | FIND NUMBER | FIND UNIQUE	**Suchoptionen:** Diese Optionen dienen dazu ■ 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***). Näheres hierzu finden Sie in den entsprechenden Abschnitten weiter unten.
MULTI-FETCH-clause	**Multi-Fetch-Klausel:** 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 Datenbank-Sätzen verarbeitet wird. Um die Leistung dieser Programme zu verbessern, bietet Natural die MULTI-FETCH-Klausel, die es ermöglicht, mehr als einen Datensatz pro Datenbankzugriff zu lesen. Weitere Informationen, siehe ***MULTI-FETCH-Klausel*** weiter unten.
view-name	**View-Angabe:** Der Name eines Views, der entweder in einem DEFINE DATA-Statement-Block oder in einer separaten Global oder Local Data Area definiert ist. Im Reporting Mode ist *view-name* der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.
PASSWORD=*operand2*	**PASSWORD-Klausel:**

Syntax-Element	Beschreibung
	Die PASSWORD-Klausel gilt nur für Zugriffe auf Adabas- oder VSAM-Datenbanken. Mit Entire System Server und Natural Messaging ist diese Klausel nicht erlaubt. Die PASSWORD-Klausel dient dazu, ein Passwort (*operand2*) anzugeben, um auf Daten einer passwortgeschützten Adabas- oder VSAM-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. Wenn das Passwort als Konstante angegeben wird, sollte die PASSWORD-Klausel ganz am Anfang einer Quellcode-Zeile stehen und es sollte sich zwischen dem Schlüsselwort PASSWORD und dem Gleichheitszeichen kein Leerzeichen befinden; dadurch ist gewährleistet, dass das Passwort im Quellcode nicht sichtbar ist. Im TP-Modus können Sie die PASSWORD-Klausel unsichtbar machen, indem Sie dann zuerst das Terminalkommando %* eingeben, bevor Sie das Passwort eintippen. Wenn Sie die PASSWORD-Klausel weglassen, gilt das mit dem PASSW-Statement angegebene Passwort. Während der Ausführung einer Verarbeitungsschleife kann das Passwort nicht geändert werden. Siehe auch ***Beispiel 1 - PASSWORD-Klausel***.
CIPHER=*operand3*	**CIPHER-Klausel:** Die CIPHER-Klausel gilt nur für Zugriffe auf Adabas-Dateien. Mit Entire System Server und Natural Messaging ist diese Klausel nicht erlaubt. Die CIPHER-Klausel dient dazu, einen Chiffrierschlüssel (*operand3*) anzugeben, um in chiffrierter Form gespeicherte Daten von Adabas-Dateien in entschlüsselter Form zu erhalten. Wollen Sie auf eine chiffrierte Datei zugreifen, sollten Sie sich bezüglich des Chiffrierschlüssels mit Ihrem Datenbank-Security-Administrator in Verbindung setzen. Der Chiffrierschlüssel kann als numerische Konstante (8 Stellen lang) oder als Inhalt einer Benutzervariablen (Format/Länge N8) angegeben werden. Wird der Chiffrierschlüssel als Konstante angegeben, sollte die CIPHER-Klausel ganz am Anfang einer Quellcode-Zeile stehen; dadurch

Syntax-Element	Beschreibung
	ist gewährleistet, dass der Chiffrierschlüssel im Quellcode nicht sichtbar ist. Im TP-Modus können Sie die CIPHER-Klausel unsichtbar machen, indem Sie zuerst das Terminalkommando %* eingeben, bevor Sie den Chiffrierschlüssel eintippen. Während der Ausführung der FIND-Verarbeitungsschleife kann der Chiffrierschlüssel nicht geändert werden. Siehe auch *Beispiel 2 - CIPHER-Klausel*.
WITH LIMIT *operand4* *basic-search-criterion*	**WITH-Klausel:** 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*. Datenbank-spezifische Hinweise: ■ **Für Adabas-Dateien:** Sie können in der WITH-Klausel einen Deskriptor, einen Subdeskriptor, einen Superdeskriptor, einen Hyperdeskriptor oder einen phonetischen Deskriptor angeben. Es kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit N markiert ist) angegeben werden. ■ **Für VSAM-Dateien:** Sie können nur ein VSAM-Schlüsselfeld angeben. 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 (*operand4*). Übersteigt die Anzahl der ausgewählten Datensätze diese Zahl, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen. **Anmerkung:** Wenn das Limit eine vierstellige Zahl sein soll, geben Sie diese mit einer vorangestellten Null an: (0*nnnn*); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.
COUPLED-clause	**COUPLED-Klausel:** Diese Klausel gilt nur für Such-Zugriffe auf Adabas-Dateien. Siehe *COUPLED-Klausel*.
STARTING WITH ISN=*operand5*	**STARTING WITH-Klausel:** Diese Klausel kann zum Repositionieren innerhalb einer FIND-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden.

<table>
<tr><th>Syntax-Element</th><th>Beschreibung</th></tr>
<tr><td></td><td>Siehe STARTING WITH-Klausel.</td></tr>
<tr><td>SORTED-BY-clause</td><td>SORTED BY-Klausel:

Diese Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren.

Siehe SORTED BY-Klausel.</td></tr>
<tr><td>RETAIN-clause</td><td>RETAIN-Klausel:

Mit dieser Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen.

Siehe RETAIN-Klausel.</td></tr>
<tr><td>[[IN] SHARED HOLD [MODE=option]]</td><td>SHARED HOLD-Klausel:

Anmerkung: Diese Klausel gilt nur für Zugriffe auf Adabas Version 8.2 oder höher.

Mit dieser Option können Sie die zurzeit gelesenen Datensätze in einen „Shared Hold“-Status setzen. Ein Datensatz kann durch viele Benutzer gleichzeitig in den „Shared Hold“-Status gesetzt werden. Solange sich ein Datensatz im „Shared Hold“-Status befindet, ist er vor Änderungen geschützt, weil er nicht durch zeitgleich arbeitende Benutzer in einen „Exclusive Hold“-Status gesetzt werden kann. Dadurch wird die Datenkonsistenz für die Daten des Datensatzes sichergestellt, weil niemand den Datensatz ändern kann, während er sich in Verarbeitung befindet.

Insbesondere wenn auf denselben Datensatz mit mehreren Statements zugegriffen wird, um verschiedene MU-/PE-Ausprägungen zu lesen (GET SAME-Statement) oder um ein LOB-Feld segmentweise zu lesen (READLOB-Statement), kann der „Shared Hold“-Status die Stabilität der Daten während dieser Transaktion garantieren, ohne den Datensatz für andere Benutzer zu blockieren.

Obwohl ein solcher Hold-Status eine effiziente Art ist, Leseabfolgen zu schützen, ist es dennoch eine grundlegende und wichtige Angelegenheit, den Datensatz aus dieser sanften Sperre wieder freizugeben. Da dies von individuellen Aspekten der Anwendung abhängt, besteht die Möglichkeit, mit Hilfe der MODE-Subklausel verschiedene Optionen auszuwählen.
<table>
<tr><th>MODE-Option</th><th>Sperrzeitraum</th><th>Erläuterung</th></tr>
<tr><td>C</td><td>Nur zum Zeitpunkt des Lesens des Datensatzes;</td><td>Stellt sicher, dass die zurzeit gelesene Version des Datensatzes durch den letzten Änderer festgeschrieben worden</td></tr>
</table>
</td></tr>
</table>

Syntax-Element	Beschreibung		
			ist. Diese Option setzt nicht wirklich eine Sperre, sondern prüft nur, ob der Datensatz zum Zeitpunkt des Lesens nicht durch einen anderen Benutzer in einem „Exclusive Hold"-Status gehalten wird.
	Q	solange bis der nächste Datensatz in einer Abfolge gelesen ist;	Gibt den Datensatz aus dem „Shared Hold"-Status frei, wenn ■ der nächste Datensatz in der Schleifenabfolge gelesen wird oder ■ die Schleife beendet wird oder ■ ein `END TRANSACTION`- oder `BACKOUT TRANSACTION`-Statement ausgeführt wird.
	S	bis die logische Transaktion beendet ist.	Gibt den Datensatz aus dem „Shared Hold"-Status frei, wenn eine logische Transaktion mit einem `END TRANSACTION`- oder `BACKOUT TRANSACTION`-Statement beendet wird.
	`MODE=Q` und `MODE=S` stellen sicher, dass der zurzeit gelesene Datensatz solange nicht gleichzeitig durch andere Benutzer geändert werden kann, bis er wieder aus dem Hold-Status freigegeben worden ist. Falls die `MODE`-Subklausel weggelassen wird, gilt `MODE=C` als Standardwert. Siehe auch ***Beispiel 15 - SHARED HOLD-Klausel*** weiter unten.		
`SKIP RECORDS IN HOLD`	**SKIP RECORDS-Klausel:** **Anmerkung:** Diese Klausel kann nur für Zugriffe auf Adabas benutzt werden.		

Syntax-Element	Beschreibung
	Immer wenn ein im Hold befindlicher Datensatz gelesen werden soll, kann ein Natural-Fehler NAT3145 (Adabas-Rückmeldeschlüssel 145) auftreten, falls der Datensatz zu diesem Zeitpunkt durch einen anderen Benutzer in den Hold-Status versetzt worden ist. Das geschieht, wenn ein „Shared Hold"-Status angefordert wird und der Datensatz sich im „Exclusive Hold"-Status befindet, oder wenn ein „Exclusive Hold"-Status angefordert wird und der Datensatz sich entweder im „Exclusive Hold"- oder im „Shared Hold"-Status befindet. Gewiss ist der Natural-Fehler NAT3145 die angemessene Reaktion, um eine einwandfreie Datenverarbeitung sicherzustellen, jedoch kann es gelegentlich von Nutzen sein, dass ein im Hold befindlicher Datensatz übersprungen werden kann. Wenn es in Ordnung ist, dass ein solcher Datensatz nicht bearbeitet und die Schleifenverarbeitung fortgesetzt wird, sollte die SKIP RECORDS-Klausel verwendet werden. Wenn die SKIP RECORDS-Klausel angewendet wird, versucht Natural zunächst, den Datensatz mit Hold zu lesen. Wenn die Fehlersituation für einen Natural-Fehler NAT3145 eintrifft, dann ■ wird keine Fehlerverarbeitung eingeleitet; ■ der (zurzeit durch einen anderen Benutzer gesperrte) Datensatz wird sofort ohne Hold erneut abgerufen, jedoch nicht im Sinne der Programmlogik verarbeitet; ■ der Datensatz, der nach dem übersprungenen Datensatz an der Reihe ist, wird ohne Hold gelesen und die Verarbeitung wird fortgesetzt. Die SKIP RECORD IN HOLD-Funktionalität greift nur, wenn der Natural-Profilparameter WH auf OFF gesetzt ist. Siehe auch ***Beispiel 16 - SKIP RECORDS-Klausel***.
WHERE-clause	**WHERE-Klausel:** Diese Klausel dient dazu, ein zusätzliches Selektionskriterium (*logical-condition*) anzugeben. Siehe ***WHERE-Klausel***.
IF-NO-RECORDS-FOUND-clause	**IF NO RECORDS FOUND-Klausel:** 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. Siehe ***IF NO RECORDS FOUND-Klausel***.

Syntax-Element	Beschreibung
END-FIND (*r*)	**Ende des FIND-Statements:**

LOOP (*r*)	Im Structured Mode ohne Verarbeitungsschleife muss dass für Natural reservierte Schlüsselwort END-FIND zum Beenden des FIND-Statements verwendet werden.
	
	Im Structured Mode können Sie bei END-FIND Labels oder Zeilennummern angeben.
	
	Beispiel:
	
	`FD. FIND EMPLOYEES-VS WITH  PERSONNEL-ID = '11100112'`
	`DISPLAY PERSONNEL-ID NAME`
	`END-FIND (FD.)`
	
	`0150   FD. FIND MYVIEW2 WITH PERSONNEL-ID = '11000999'`
	`0160    DISPLAY NOTITLE NAME`
	`0170                    FIRST-NAME (0150)`
	`0180                    MAKE       (FD.)`
	`0190   END-FIND (0150)`
	
	Im Reporting Mode mit Verarbeitungsschleife wird das Natural-Statement **LOOP zum Beenden des FIND-Statements verwendet.**
	
	Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

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.

Bei FIND FIRST vorhandene Systemvariablen

Beim FIND FIRST-Statement stehen folgende Natural-Systemvariablen zur Verfügung:

Systemvariable	Erläuterung
*ISN	Die Systemvariable *ISN enthält die Adabas-ISN (Interne Satznummer) des ausgewählten Datensatzes. *ISN enthält den Wert Null (0), wenn kein Datensatz die WITH- und WHERE-Selektionskriterien erfüllt. *ISN steht für VSAM-Datenbanken nicht zur Verfügung. Mit Entire System Server und Natural Messaging ist diese Klausel nicht erlaubt.
*NUMBER	Die Systemvariable *NUMBER enthält die Anzahl der Datensätze, die vor Auswertung des WHERE-Kriteriums das in der WITH-Klausel angegebene primäre Selektionskriterium erfüllt haben. *NUMBER enthält den Wert Null (0), wenn kein Datensatz das WITH-Kriterium erfüllt. Mit Entire System Server und Natural Messaging kann *NUMBER nicht verwendet werden.
*COUNTER	Die Systemvariable *COUNTER 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

- Im Structured Mode darf keine WHERE-Klausel benutzt werden.
- Mit Entire System Server und Natural Messaging kann FIND NUMBER nicht verwendet werden.

Bei FIND NUMBER vorhandene Systemvariablen

Bei FIND NUMBER stehen folgende Natural-Systemvariablen zur Verfügung:

Systemvariable	Erläuterung
*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 und Natural Messaging nicht möglich.
- Bei SQL-Datenbanken ist FIND UNIQUE nicht erlaubt. (Ausnahme: auf z/OS-Computern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)

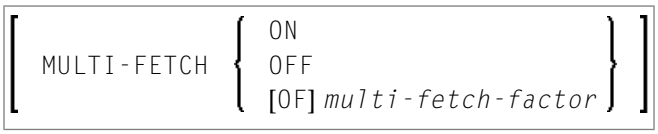
Bei FIND UNIQUE vorhandene Systemvariablen

Systemvariable	Erläuterung
*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).

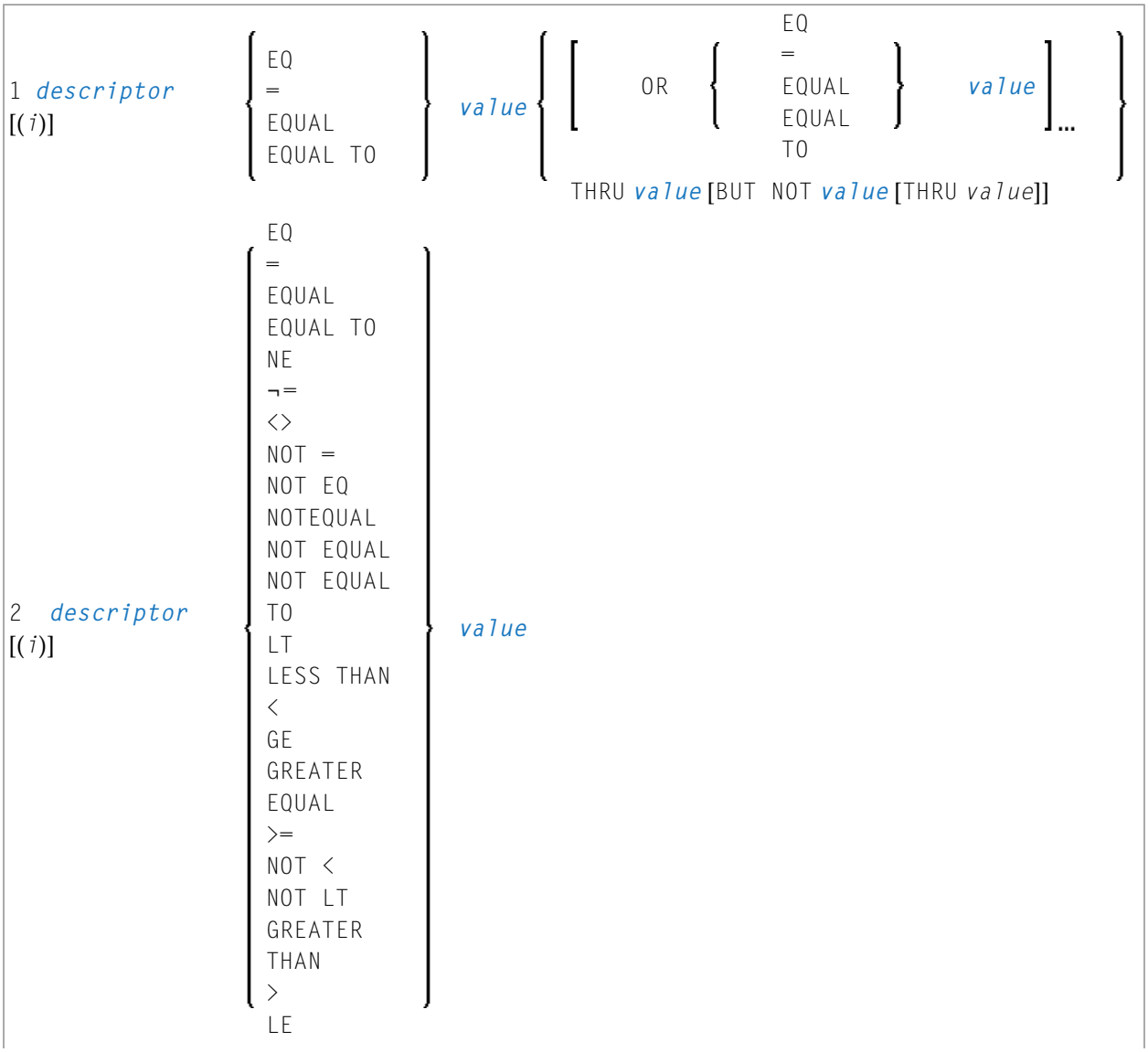
MULTI-FETCH-Klausel

Anmerkung: Diese Klausel kann nur bei Adabas- oder Db2-Datenbanken benutzt werden.



Weitere Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung* oder *Verarbeitung mehrerer Zeilen* (SQL) im *Natural for Db2*-Teil der *Database Management System Interfaces*-Dokumentation.

Suchkriterium bei Adabas-Dateien (basic-search-criterion)



```
                    LESS EQUAL
                    <=
                    NOT >
                    NOT GT
3 set-name
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
descriptor		S	A			A		N	P	I	F	B	D	T	L				nein	nein
value	C	S				A		N	P	I	F	B	D	T	L				ja	nein
set-name	C	S				A													nein	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
descriptor	**Deskriptor-Feld:** Es kann ein Adabas-Deskriptor, -Subdeskriptor, -Superdeskriptor, -Hyperdeskriptor oder phonetischer Deskriptor angegeben werden. Es kann auch ein im DDM als Nicht-Deskriptor markiertes Feld angegeben werden.
(i)	**Index:** 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. 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.
value	**Suchwert:** Der Suchwert, den das Deskriptorfeld haben soll. Die Formate von Suchwert und Deskriptorfeld müssen kompatibel sein.
set-name	**Set-Name:** 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. *set-name* kann entweder als Textkonstante (bis zu 32 Zeichen lang) oder in Form einer alphanumerischen Variablen angegeben werden.

Syntax-Element	Beschreibung
	Mit Entire System Server und Natural Messaging kann *set-name* nicht angegeben werden.

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 { =      } value
               { EQ     }
               { EQUAL  }
               { [TO]   }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
null-indicator		S							I								nein	nein
value	C	S					N	P	I	F	B						ja	nein

Syntax-Element-Beschreibung:

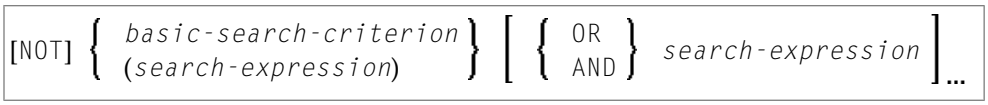
Syntax-Element	Beschreibung	
null-indicator	Der Null-Indikator.	
value	**Mögliche Werte**	**Bedeutung**
	-1	Das entsprechende Feld enthält keinen Wert.
	0	Das entsprechende Feld enthält einen Wert.

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:



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

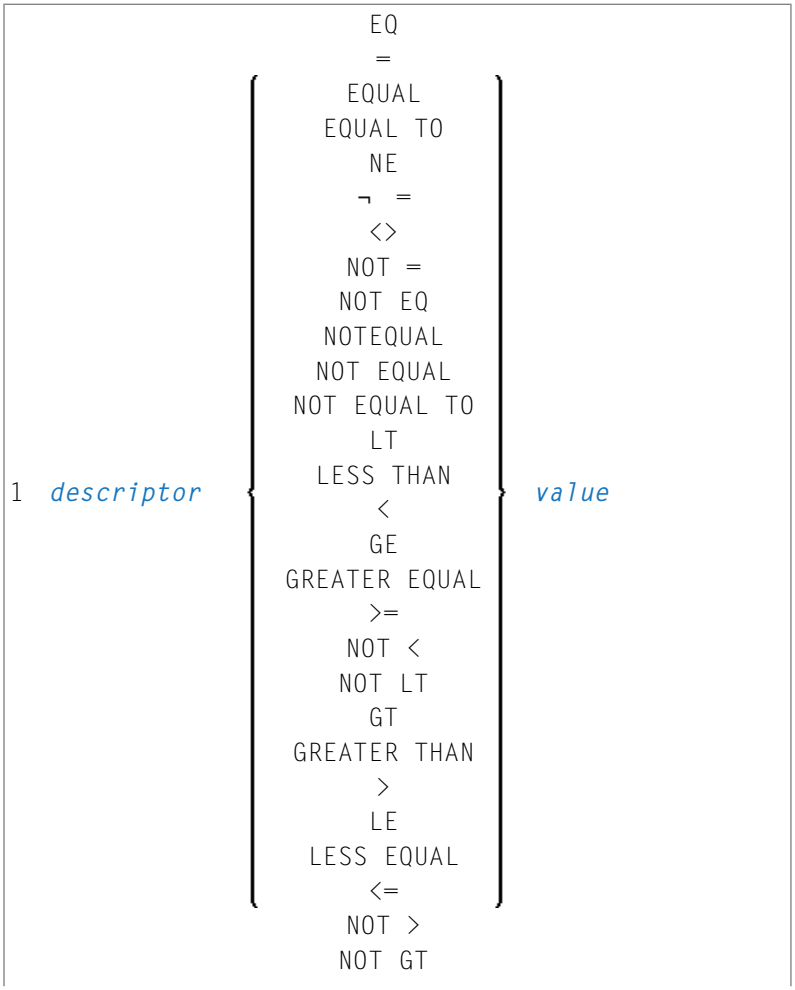
Ein Deskriptor, der Teil eines Datenbank-Arrays ist, kann ebenfalls als Suchfeld verwendet werden. Bei Adabas-Dateien kann dies ein multiples Feld sein oder ein Feld, das in einer Periodengruppe enthalten ist.

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*.

Suchkriterium bei VSAM-Dateien (basic-search-criterion)





Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
descriptor		S	A			A	N	P			B						nein	nein
value	C	S				A	N	P			B						ja	nein

Syntax-Element-Beschreibung:

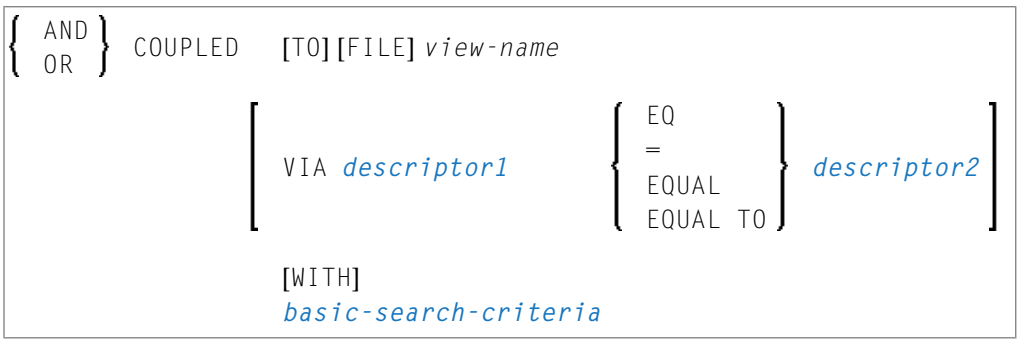
Syntax-Element	Beschreibung
descriptor	**Deskriptor:** Es kann ein Deskriptor angegeben werden, der in einer VSAM-Datei als VSAM-Schlüsselfeld definiert und im entsprechenden Natural-DDM mit `P` (Primärschlüssel) oder `A` (Alternativschlüssel) markiert ist.
value	**Suchwert:** Der Suchwert, den das Schlüsselfeld haben soll.

Die Formate des Deskriptors (*descriptor*) und des Suchwerts (*value*) müssen kompatibel sein.

COUPLED-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-Dateien.

Mit Entire System Server und Natural Messaging ist diese Klausel nicht zulässig.



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- und VSAM-Datenbanken; für VSAM ist sie nur bei ESDS gültig.

Sie können diese Klausel benutzen, um als *operand5* eine Adabas-ISN (Internal Sequence Number) oder VSAM-RBA (relative Byte-Adresse) anzugeben, die als Startwert für die Auswahl von Datensätzen benutzt werden soll. *operand5* muss im Bereich von 0 bis 4294967295 sein. Für Extended ESDS VSAM-Dateien kann dieser Wert bei einer Steuerintervallgröße von 4 KB bis zu 16 TB und bei einer Steuerintervallgröße von 32 KB bis zu 128 TB betragen.

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- und SQL-Datenbanken.

Mit Entire System Server und Natural Messaging ist diese Klausel nicht zulässig.

```
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 in der Liste 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 z/OS-Computern 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*.

Hinweise 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 for z/OS

Bei Adabas for z/OS 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 angegeben 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 for OpenSystems

Bei Adabas for OpenSystems (Linux and Cloud oder Windows) 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:

Wenn man das folgende Programm mit Adabas for z/OS und Adabas for Linux and Cloud/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 for z/OS

```
ISN        NAME         FIRST-NAME       ↩
  CITY
----- --------------- --------------- ↩
---------------
                                        ↩

FIND V1 WITH NAME = 'B' THRU 'BALBIN'
   12 BAILLET         PATRICK         LYS ↩
LEZ LANNOY
   58 BAGAZJA          MARJAN           ↩
MONTHERME
  351 BAECKER          JOHANNES         ↩
FRANKFURT
  355 BAECKER          KARL             ↩
SINDELFINGEN
  370 BACHMANN         HANS             ↩
MUENCHEN
  490 BALBIN           ENRIQUE          ↩
BARCELONA
  650 BAKER           SYLVIA          OAK ↩
BROOK
  913 BAKER            PAULINE          ↩
DERBY
                                        ↩

FIND .. SORTED BY NAME                  ↩

  370 BACHMANN         HANS             ↩
MUENCHEN
  351 BAECKER          JOHANNES         ↩
FRANKFURT
  355 BAECKER          KARL             ↩
SINDELFINGEN
   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
```

Ergebnis bei Natural for OpenSystems

```
ISN        NAME         FIRST-NAME       ↩
   CITY
----- --------------- --------------- ↩
---------------
                                        ↩

FIND V1 WITH NAME = 'B' THRU 'BALBIN'   ↩

   12 BAILLET         PATRICK          ↩
LYS LEZ LANNOY
   58 BAGAZJA          MARJAN           ↩
MONTHERME
  351 BAECKER          JOHANNES         ↩
FRANKFURT
  355 BAECKER          KARL             ↩
SINDELFINGEN
  370 BACHMANN         HANS             ↩
MUENCHEN
  490 BALBIN           ENRIQUE          ↩
BARCELONA
  650 BAKER            SYLVIA           ↩
OAK BROOK
  913 BAKER            PAULINE          ↩
DERBY
                                        ↩

FIND .. SORTED BY NAME                  ↩

  370 BACHMANN         HANS             ↩
MUENCHEN
  351 BAECKER          JOHANNES         ↩
FRANKFURT
  355 BAECKER          KARL             ↩
SINDELFINGEN
   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             ↩
```

Ergebnis bei Natural for z/OS	Ergebnis bei Natural for OpenSystems
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	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 und Natural Messaging ist diese Klausel nicht zulässig.

```
RETAIN AS operand6
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand6	C	S				A										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`RETAIN AS`	**Suchergebnis für weitere Verarbeitung beibehalten:** Mit der `RETAIN`-Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für eine weitere Verarbeitung beizubehalten. Die ausgewählten Datensätze werden als ISN-Set in die Adabas-Arbeitsdatei gestellt. Dieser Set kann in nachfolgenden `FIND`-Statements als Suchkriterium (*`basic-search-criterion`*) angegeben werden, um aus dem Set eine gezieltere Auswahl von Datensätzen für die weitere Verarbeitung zu treffen. 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.
operand6	**Set-Name:** 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.

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 }
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`IF NO RECORDS FOUND`	**IF NO RECORDS FOUND-Klausel:** 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. In einem solchen 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.
`ENTER` *`statement ...`*	**Statements für die Verarbeitung:** 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.
`END-NOREC`	**Ende der IF NO RECORDS FOUND-Klausel:** Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-NOREC` zum Beenden der `IF NO RECORDS FOUND`-Klausel benutzt werden. Im Reporting Mode werden die Statements `DO ... DOEND` benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um die `IF NO RECORDS FOUND`-Klausel zu beenden. Falls Sie nur ein einziges Statement oder nur das Schlüsselwort `ENTER` (s.o.) angeben, können Sie die Statements `DO ... DOEND` weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

`ENTER`
`statement`
`DO` *`statement ...`*
`DOEND`	

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 FIND

- 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
- Beispiel 15 - SHARED HOLD-Klausel
- Beispiel 16 - SKIP RECORDS-Klausel

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'
  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'
  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          PERSONNEL         NAME
                             POSITION             ID
-------------------- ------------------------- --------- --------------------

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         NAME            ISN         NMBR        CNT
   ID
--------- -------------------- ----------- ----------- -----------

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
```

Beispiel 15 - SHARED HOLD-Klausel

```
FIND EMPL-VIEW WITH NAME = ...
    IN SHARED HOLD MODE=Q              /* Record in shared hold until next record ↩
is read.
   ...
   GET EMPL-VIEW  *ISN                    /* The record remains unchanged!
   ...
END-FIND
```

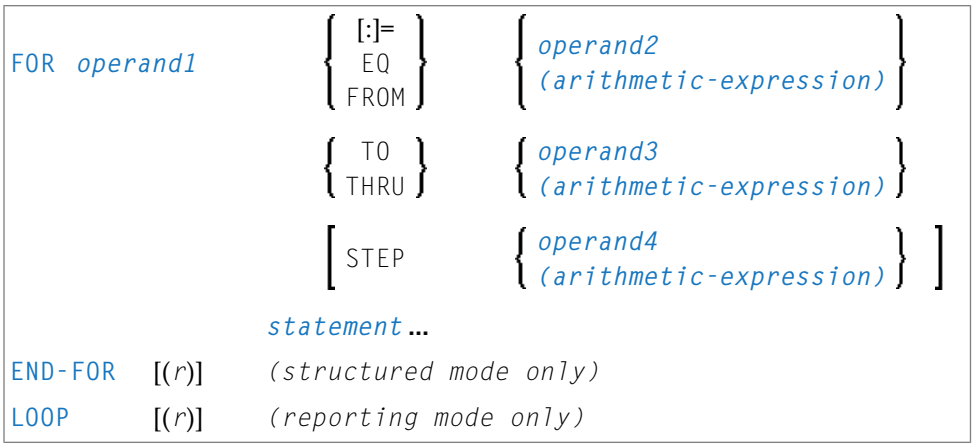
Beispiel 16 - SKIP RECORDS-Klausel

```
FIND EMPL-VIEW WITH NAME = ...             /* Records found are put in hold while ↩
reading.
     SKIP RECORDS IN HOLD                  /* Records already held by other users ↩
are skipped
  ...                                      /* to prevent error NAT3145.
  UPDATE                                                                      ↩

  END TRANSACTION
END-FIND
```

66 FOR

- Funktion FOR 524
- Syntax-Beschreibung FOR 525
- Beispiel für FOR-Statement 526



Dieser Abschnitt 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 FOR

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 FOR

Operanden-Definitionstabelle:

Operand	Mögliche Struktur						Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1		S					N	P	I	F							ja	ja
operand2	C	S			N		N	P	I	F							ja	nein
operand3	C	S			N		N	P	I	F							ja	nein
operand4	C	S			N		N	P	I	F							ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Schleifenkontrollvariable (*operand1*) und Ausgangswert (*operand2*):** *operand1* kann ein Datenbankfeld oder eine Benutzervariable sein und steuert die Anzahl der Schleifendurchläufe. Der nach dem Schlüsselwort FROM angegebene Wert (*operand2*) wird der Kontrollvariablen als Ausgangswert zugeordnet, bevor die Verarbeitungsschleife das erste Mal durchlaufen wird. Dieser Ausgangswert erhöht *operand2* wird. sich mit jedem Schleifendurchlauf um den Wert des mit STEP angegebenen *operand4* (bzw. vermindert sich, wenn *operand4* einen negativen Wert hat). Die Kontrollvariable kann während der Ausführung der Verarbeitungsschleife referenziert werden, um den aktuellen Wert der Kontrollvariablen zu erhalten.
operand2	
operand3	**TO-Wert:** Die Verarbeitungsschleife wird beendet, sobald *operand1* einen Wert enthält, der größer ist als der Wert von *operand3* (oder kleiner, falls der STEP-Wert negativ ist).
operand4	**STEP-Wert:** Der Wert von *operand4* kann positiv oder negativ sein. Ist kein Wert angegeben, wird ein Wert von +1 angenommen. Je nach dem Vorzeichen des STEP-Wertes wird die Vergleichsoperation für *operand3* auf *größer als* oder *kleiner als* gesetzt, wenn die Verarbeitungsschleife zum erstenmal durchlaufen wird. *operand4* darf nicht Null (0) sein.
(*arithmetic-expression*)	**Arithmetischer Ausdruck:**

Syntax-Element	Beschreibung
	Anstelle von *operand2*, *operand3* oder *operand4* kann ein beliebiger arithmetischer Ausdruck angegeben werden. Der arithmetische Ausdruck muss in Klammern angegeben werden. Vorangehende Schlüsselwörter dürfen nicht weggelassen werden. Weitere Informationen siehe *arithmetic-expression* in der Beschreibung des COMPUTE-Statements.
END-FOR (*r*) LOOP (*r*)	**Ende des FOR-Statements:** Im Structured Mode muss das für Natural reservierte Wort END-FOR zum Beenden des FOR-Statements benutzt werden. Im Structured Mode können Sie bei END-FOR Labels oder Zeilennummern angeben. Im Reporting Mode wird das LOOP-Statement zum Beenden des FOR-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

Beispiel für 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.

67 FORMAT

- Funktion FORMAT 530
- Syntax-Beschreibung FORMAT 531
- Parameter beim FORMAT-Statement 532
- Beispiel für FORMAT-Statement 533

```
FORMAT [(rep)]  parameter ...
```

Dieser Abschnitt 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 FORMAT

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 FORMAT

Syntax-Element	Beschreibung
(rep)	**Report-Spezifikation:** Mit der Notation *(rep)* kann ein bestimmter anderer Report angegeben werden, auf den sich das FORMAT-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 FORMAT-Statement auf den ersten Report (Report 0). Informationen zum Steuern des Formats eines mit Natural erzeugten Ausgabe-Reports siehe *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.

parameter	**Parameter:**
	
	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 Quellcode-Zeile hinausgehen.
	
	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.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 VARI (A4)  INIT <'1234'>  /*  Output`
	`END-DEFINE  /*  Produced`
	`FORMAT AD=U  /*  ---------`
	`WRITE 'Text'  VARI  /*  Text 1234`
	`WRITE 'Text' (AD=U)  VARI  /*  Text 1234`
	`END`
	
	Siehe auch *Parameter* weiter unten.

Parameter beim FORMAT-Statement

Die Beschreibungen der Parameter, die Sie beim `FORMAT`-Statement verwenden können, finden Sie in der *Parameter-Referenz*.

Parameter	Beschreibung
`AD`	Attribut-Definition
`AL`	Alphanumerische Länge der Ausgabe
`BX`	Feldumrahmung (Box-Definition)
`CD`	Farbdefinition
`DF`	Datumsformat
`DL`	Ausgabelänge
`EM`	Editiermaske
`ES`	Leerzeilenunterdrückung
`FC`	Füllzeichen für DISPLAY-Statement
`FL`	Gleitkomma-Mantissenlänge
`GC`	Füllzeichen für Gruppenüberschriften
`HC`	Überschriften-Zentrierung
`HW`	Überschriftenbreite
`IC`	Einfügungszeichen
`ICU`	Unicode-Einfügungszeichen
`IP`	Eingabeaufforderungstext
`IS`	Unterdrückung identischer Werte
`KD`	PF-Tasten-Anzeige
`LC`	Vorangestellte Zeichens
`LCU`	Vorangestellte Unicode-Zeichen
`LS`	Zeilenlänge
`MC`	Anzahl multipler Feldwerte (Kann nur im Reporting Mode verwendet werden.)
`MP`	Maximale Seitenzahl eines Reports
`MS`	Manuelle Cursor-Positionierung
`NL`	Numerische Länge der Ausgabe
`PC`	Anzahl der Periodengruppen-Ausprägungen (Kann nur im Reporting Mode verwendet werden.)
`PM`	Druck-Modus
`PS`	Länge einer Reportseite
`SF`	Spaltenabstand
`SG`	Vorzeichen-Stelle

Parameter	Beschreibung
TC	Nachgezogene Zeichen
TCU	Nachgezogene Zeichen (Unicode)
UC	Unterstreichungszeichen
ZP	Anzeige von Nullwerten

Beispiel für FORMAT-Statement

```
** 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  >>
```

68 GET

- Funktion GET 536
- Einschränkungen GET 537
- Syntax-Beschreibung GET 537
- Beispiel für GET-Statement 538

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] ddm-name
      [PASSWORD=operand1]
      [CIPHER=operand2]
      [ { [RECORD]  } ] { operand3   } operand4
      [ { [RECORDS] } ] { *ISN [(r)] } ...
```

Dieser Abschnitt 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 GET

Das Statement `GET` dient dazu, einen Datensatz mit einer bestimmten Adabas-ISN (Interne Satz-Nummer) zu lesen.

Das `GET`-Statement löst keine Verarbeitungsschleife aus.

Einschränkungen GET

- Das GET-Statement kann nicht für SQL-Datenbanken verwendet werden.
- Mit Entire System Server kann das GET-Statement nicht verwendet werden.

Syntax-Beschreibung GET

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A											ja	nein
operand2	C	S					N										nein	nein
operand3	C	S			N		N	P	I		B *						ja	nein
operand4		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:

Syntax-Element	Beschreibung
view-name	**View-Name:** Im Structured Mode sowie im Reporting Mode mit einem DEFINE DATA LOCAL-Statement ist *view-name* der Name eines Views, der entweder direkt in einem DEFINE DATA-Statement oder in einer separaten Global oder Local Data Area definiert ist.
ddm-name	**DDM-Name:** Im Reporting Mode ohne DEFINE DATA LOCAL-Statement ist *ddm-name* der Name eines DDMs.
PASSWORD=*operand1*	**PASSWORD-Klausel/CIPHER-Klausel:** Diese beiden Klauseln sind nur auf Adabas- und VSAM-Datenbanken anwendbar. Die PASSWORD-Klausel dient dazu, ein Passwort anzugeben, um auf Daten einer passwort-geschützten Adabas-Datei zugreifen zu können. Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Adabas-Daten in entschlüsselter Form zu erhalten.
CIPHER=*operand2*	

Syntax-Element	Beschreibung
	Näheres hierzu siehe FIND- und PASSW-Statement.
*ISN / *operand3*	**Interne Sequenz-Nummer:** Die ISN kann als numerische Konstante oder Benutzervariable (*operand3*) oder über die Natural-Systemvariable *ISN angegeben werden. **Anmerkung:** Bei VSAM-Datenbanken: Für VSAM-ESDS muss die RBA in einer numerischen Benutzervariablen enthalten sein oder als Ganzzahl-Konstante angegeben werden. Dasselbe gilt für VSAM-RRDS, außer dass hier statt der RBA die RRN angegeben werden muss.
(*r*)	**Statement-Referenz:** Die Notation (*r*) wird benutzt, um das Statement anzugeben, welches das zum ersten Lesen des Datensatzes benutzte FIND- oder READ-Statement enthält. Wenn (*r*) nicht angegeben wird, bezieht sich das GET-Statement auf die innerste aktive Verarbeitungsschleife. (*r*) kann als Statement-Label oder Quellcode-Zeilennummer angegeben werden.
operand4	**Referenz auf Datenbank-Felder:** Eine spätere Referenzierung von Datenbankfeldern, die mit einem GET-Statement gelesen wurden, kann entweder unter Angabe eines Statement-Labels oder über die Quellcode-Zeilennummer des GET-Statements erfolgen.

Beispiel für GET-Statement

```
** 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         NAME              FIRST-NAME
            ID
-------- --------- -------------------- --------------------

  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
```

69 GET SAME

- Funktion GET SAME 542
- Einschränkungen GET SAME 542
- Syntax-Beschreibung GET SAME 543
- Beispiel für GET SAME-Statement 543

Structured Mode-Syntax

```
GET SAME [(r)]
```

Reporting Mode-Syntax

```
GET SAME [(r)] [operand1 ...]
```

Dieser Abschnitt 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 GET SAME

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 GET SAME

- Das Statement GET SAME ist nur beim Zugriff auf Adabas- oder VSAM-Datenbanken gültig.
- Mit Entire System Server ist dieses Statement nicht verfügbar.
- Bei VSAM-Datenbanken gilt GET SAME nur für ESDS und RRDS. Für ESDS muss die RBA in einer numerischen Benutzervariablen enthalten sein oder als Ganzzahl-Konstante angegeben werden. Dasselbe gilt für RRDS, außer dass hier statt der RBA die RRN angegeben werden muss.
- 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 GET SAME

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1		S	A			A	U	N	P			B						nein	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(r)	**Statement-Referenz:** Die Notation *(r)* wird benutzt, um das Statement anzugeben, das das FIND- oder READ-Statement enthält, mit dem der Datensatz zum erstenmal gelesen wurde. Falls keine Referenzierung erfolgt, bezieht sich das GET SAME-Statement auf die innerste aktive Verarbeitungsschleife. *(r)* kann als ein Statement-Label oder eine Quellcode- Zeilennummer angegeben werden.
operand1	**Angabe der Felder:** Als *operand1* geben Sie das Feld bzw. die Felder an, deren Werte Sie mit dem GET SAME-Statement erhalten wollen. **Anmerkung:** *operand1* kann nicht angegeben werden, wenn das Feld in einem DEFINE DATA-Statement definiert ist.

Beispiel für GET SAME-Statement

```
** 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)
```

```
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
```

70 GET TRANSACTION DATA

- Funktion GET TRANSACTION DATA 546
- Einschränkung GET TRANSACTION DATA 547
- Syntax-Beschreibung GET TRANSACTION DATA 547
- Beispiel für GET TRANSACTION DATA-Statement 547

```
GET TRANSACTION [DATA] operand1 ...
```

Dieser Abschnitt 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 GET TRANSACTION DATA

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 GET TRANSACTION DATA

Das `GET TRANSACTION DATA`-Statement gilt nur für Transaktionen auf Adabas-Datenbanken.

Syntax-Beschreibung GET TRANSACTION DATA

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S				A	U	N	P	I	F	B	D	T				ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Angabe der Felder:** 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.

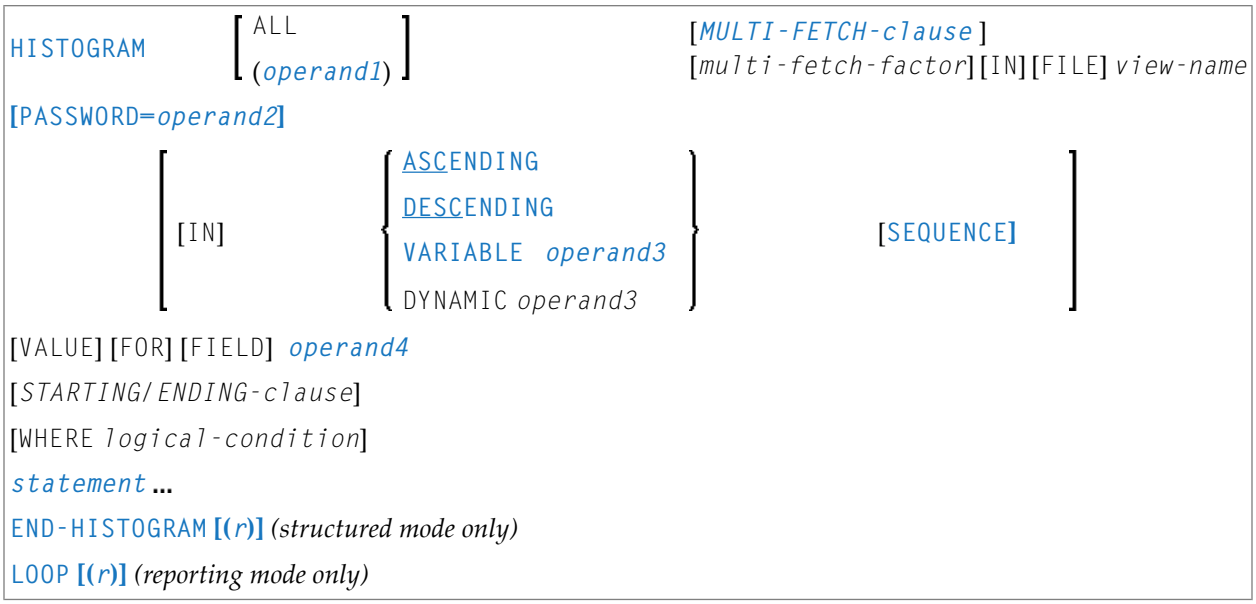
Beispiel für GET TRANSACTION DATA-Statement

```
** 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
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=O)
                 / NAME
                 / FIRST-NAME
                 / CITY
    UPDATE
    END TRANSACTION #PERS-NR
  END-FIND
  /*
END-REPEAT
END
```

71 HISTOGRAM

- Funktion HISTOGRAM 550
- Einschränkungen HISTOGRAM 551
- Syntax-Beschreibung HISTOGRAM 551
- Bei HISTOGRAM verfügbare Systemvariablen 557
- Beispiele HISTOGRAM 557



Dieser Abschnitt 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 HISTOGRAM

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 folgende Abschnitte im *Leitfaden zur Programmierung*.

- *HISTOGRAM-Statement*
- *Schleifenverarbeitung*
- *Datenbankfelder mit der (r)-Notation referenzieren*

Anmerkung: Bei SQL-Datenbanken: Mit `HISTOGRAM` erhalten Sie die Anzahl der Reihen, die in einer bestimmten Spalte den gleichen Wert haben.

Einschränkungen HISTOGRAM

Dieses Statement kann nicht mit Entire System Server verwendet werden.

Bei einer VSAM-Datenbank gilt das `HISTOGRAM`-Statement nur für KSDS und ESDS.

Syntax-Beschreibung HISTOGRAM

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I		B *						ja	nein
operand2	C	S				A												ja	nein
operand3		S				A												ja	nein
operand4		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:

Syntax-Element	Beschreibung
operand1 / ALL	**Anzahl der Deskriptorwerte:** Sie können die Anzahl der Deskriptorwerte, die mit dem `HISTOGRAM`-Statement gelesen werden sollen, auf eine bestimmte Zahl (*operand1*) begrenzen, die Sie entweder als numerische Konstante (`0` bis `99999999`) oder über eine Benutzervariable (die einen Ganzzahlwert enthält) angeben. Andernfalls werden alle Deskriptorwerte gelesen, was Sie zusätzlich durch das Schlüsselwort `ALL` hervorheben können. Das mit *operand1* angegebene Limit hat bei diesem Statement Vorrang vor einem mit einem `LIMIT`-Statement gesetzten Limit. Ist mit dem Profilparameter `LT` ein kleineres Limit gesetzt, so gilt das `LT`-Limit. **Anmerkung:**

Syntax-Element	Beschreibung
	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. *operand1* wird zu Beginn des ersten HISTOGRAM-Schleifendurchlaufs ausgewertet. Wird der Wert von *operand1* innerhalb der HISTOGRAM-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Werte.
MULTI-FETCH-clause	**MULTI-FETCH-Klausel:** Siehe ***MULTI-FETCH-Klausel*** weiter unten.
view-name	**View-Name:** Als *view-name* 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. Der View darf außer dem im HISTOGRAM-Statement verwendeten Feld (*operand4*) keine anderen Felder enthalten. 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. Im Reporting Mode ist *view-name* der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.
PASSWORD=*operand2*	**PASSWORD-Klausel:** Die PASSWORD-Klausel dient dazu, ein Passwort (*operand2*) anzugeben, um auf Daten einer passwort-geschützten Adabas-Datei zugreifen zu können. Weitere Informationen hierzu siehe FIND-Statement und PASSW-Statement.
SEQUENCE	**SEQUENCE-Klausel** Die Klausel gilt nur für Adabas-, VSAM- und SQL-Datenbanken. Mit dieser Klausel können Sie bestimmen, ob die Werte in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen. ■ Standardmäßig werden die Werte in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort ASCENDING auch ausdrücklich angeben können, aber nicht müssen). ■ Wenn die Werte in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort DESCENDING an. ■ Wenn erst zur Laufzeit bestimmt werden soll, ob die Werte in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das

Syntax-Element	Beschreibung
	Schlüsselwort VARIABLE oder DYNAMIC gefolgt von einer Variablen (*operand3*) an. Der Wert von *operand3* zu Beginn der HISTOGRAM-Verarbeitungsschleife bestimmt dann die Reihenfolge. *operand3* muss Format/Länge A1 haben und kann den Wert A (für ASCENDING = aufsteigend) oder D (für DESCENDING = absteigend) enthalten. ■ Wenn das Schlüsselwort VARIABLE benutzt wird, wird die Leserichtung (Wert von *operand3*) beim Start der HISTOGRAM-Verarbeitungsschleife ausgewertet, und bleibt dieselbe, bis die Schleife beendet wird, ungeachtet der Tatsache, ob das Feld für *operand3* in der HISTOGRAM-Schleife geändert wird oder nicht. ■ Wenn das Schlüsselwort DYNAMIC benutzt wird, wird die Leserichtung (Wert von *operand3*) 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. Beispiele für SEQUENCE-Klausel: ■ ***Beispiel 2 - HISTOGRAM-Statement mit in absteigender Reihenfolge gelesenen Sätzen*** ■ ***Beispiel 3 - HISTOGRAM-Statement mit variabler Reihenfolge***
operand4	**Deskriptor:** Als *operand4* kann ein Deskriptor, ein Subdeskriptor, ein Superdeskriptor oder ein Hyperdeskriptor angegeben werden. 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. 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.
STARTING-ENDING-clause	**STARTING/ENDING-Klausel:** 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. Weitere Informationen siehe ***Start-/Endwerte angeben*** weiter unten.

Syntax-Element	Beschreibung
WHERE *logical-condition*	**WHERE-Klausel:** Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (*logical-condition*) angeben. Dies wird ausgewertet, *nachdem* ein Wert gelesen wurde, aber bevor eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich AT BREAK-Verarbeitung) erfolgt. 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. Die Syntax für eine *logical-condition* ist im Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung* beschrieben.
END-HISTOGRAM (*r*)	**Ende des HISTOGRAM-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-HISTOGRAM zum Beenden des HISTOGRAM-Statements benutzt werden. Im Structured Mode können Sie bei END-HISTOGRAM Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des HISTOGRAM-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.
LOOP (*r*)	

MULTI-FETCH-Klausel

Anmerkung: Diese Klausel kann nur bei Adabas- oder Db2-Datenbanken benutzt werden.

```
[ MULTI-FETCH { ON                    } ]
              { OFF                   }
              { [OF] multi-fetch-factor }
```

Weitere Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung* oder *Multiple Row Processing* (SQL) im *Natural for Db2*-Teil der *Database Management System Interfaces*-Dokumentation.

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.

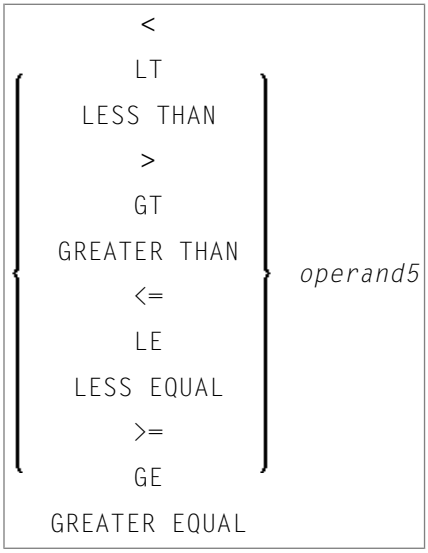
Syntax-Option 1:



Syntax-Option 2:



Syntax-Option 3:



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
operand5	C	S				A	U	N	P	I	F	B	D	T	L				ja	nein
operand6	C	S				A	U	N	P	I	F	B	D	T	L				ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`STARTING FROM ... ENDING AT/TO`	**STARTING FROM / ENDING AT-Klauseln:** Diese Klauseln werden benutzt, um das Lesen auf einen vom Benutzer angegebenen Bereich von Werten einzuschränken. Die `STARTING FROM`-Klausel (= oder `EQ` oder `EQUAL TO` oder `[STARTING] FROM`) 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 `DESCENDING`) vorhanden, wird die `HISTOGRAM`-Schleife nicht ausgeführt. Um die Werte auf einen Endwert zu beschränken, können Sie eine `ENDING AT`-Klausel bei den Schlüsselwörtern `THRU`, `ENDING AT` oder `TO` 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 `TO`, `THRU` und `ENDING AT` ähnlich ist, so unterscheiden sie sich doch intern durch ihre Funktionsweise.
`THRU/ENDING AT`	**THRU / ENDING AT-Option:** Wird `THRU` oder `ENDING AT` benutzt, 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. Die Klauseln `THRU` und `ENDING AT` können für alle Datenbanken benutzt werden, die die `HISTOGRAM`-Statements unterstützen.
`TO`	**Bereich:** Wird das Schlüsselwort `TO` 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 (`ASCENDING`) oder absteigender (`DESCENDING`) Reihenfolge durchführen.

Anmerkung: Das Ergebnis von READ/HISTOGRAM THRU/ENDING AT kann vom Ergebnis bei READ/HISTOGRAM TO abweichen, wenn Natural und die Datenbank, auf die zugegriffen wird, sich auf verschiedenen Plattformen mit unterschiedlichen Sortierfolgen befinden.

Bei HISTOGRAM verfügbare Systemvariablen

Die Natural-Systemvariablen *ISN, *NUMBER und *COUNTER können bei einem HISTOGRAM-Statement verwendet werden.

*NUMBER und *ISN stehen erst nach Auswertung der WHERE-Klausel zur Verfügung. Sie dürfen nicht innerhalb der logischen Bedingung einer WHERE-Klausel eingesetzt werden.

Systemvariable	Erläuterung
*NUMBER	Die Systemvariable *NUMBER enthält die Anzahl der Datensätze, die den zuletzt gelesenen Wert enthalten. Bezüglich SQL-Datenbanken siehe **NUMBER bei SQL Datenbanken* in der *Systemvariablen*-Dokumentation.
*ISN	Die Systemvariable *ISN enthält die Nummer 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- und VSAM-Datenbanken kann *ISN nicht verwendet werden.
*COUNTER	Die Systemvariable *COUNTER enthält die Gesamtzahl der bisher gelesenen Werte (nach Auswertung der WHERE-Klausel).

Beispiele HISTOGRAM

- 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     CNT
                        PERSONS
-------------------- ----------- -----------

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 (9) EMPL FOR NAME FROM #STARTVAL
  WRITE NAME *NUMBER
  IF *COUNTER = 5
    MOVE NAME TO #STARTVAL
  END-IF
END-HISTOGRAM
*
#DIR := 'A'
*
REPEAT
  HISTOGRAM EMPL IN VARIABLE #DIR SEQUENCE
            FOR NAME FROM #STARTVAL
```

```
    MOVE NAME TO #STARTVAL
    INPUT NO ERASE  (IP=OFF AD=O)
          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
```

72 IF

- Funktion IF 564
- Syntax-Beschreibung IF 564
- Beispiel für IF-Statement 565

Structured Mode-Syntax

```
IF logical-condition
[THEN] statement ...
[ELSEstatement ... ]
END-IF
```

Reporting Mode-Syntax

```
IF logical-condition
[THEN]    { statement          }
          { DO statement ... DOEND }
[  ELSE   { statement          }  ]
          { DO statement ... DOEND }
```

Dieser Abschnitt 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 IF

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

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung

`IF` *logical-condition*	**Logische Bedingung:**
	
	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.
	
	Beispiele:
	
	`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'`
	
	Näheres hierzu siehe Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung*.

`THEN` *statement*	**THEN-Klausel:** In der `THEN`-Klausel geben Sie eines oder mehrere *`Statements`* an, die ausgeführt werden sollen, wenn die logische Bedingung erfüllt ist.
`ELSE` *statement*	**ELSE-Klausel:** In der `ELSE`-Klausel geben Sie eines oder mehrere *`Statements`* an, die ausgeführt werden sollen, wenn die logische Bedingung *nicht* erfüllt ist.

`END-IF`
 	
statement
`DO` *statement* `...` `DOEND`
	Ende des IF-Statements:
	
	Im Structured Mode muss das für Natural reservierte Wort `END-IF` zum Beenden des `IF`-Statements benutzt werden.
	
	Im Reporting Mode werden die Statements `DO ... DOEND` benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um die Klauseln und das `IF`-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements `DO ... DOEND` weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Beispiel für 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      ANNUAL     MAKE
                         OF       SALARY
                       BIRTH
-------------------- ---------- ---------- --------

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`.

73 IF SELECTION

- Funktion IF SELECTION 568
- Syntax-Beschreibung IF SELECTION 568
- Beispiel für IF SELECTION-Statement 570

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 }
```

Dieser Abschnitt 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 IF SELECTION

Das Statement `IF SELECTION` dient dazu, zu verifizieren, dass in einer Reihe von alphanumerischen Feldern genau ein Feld einen Wert enthält.

Syntax-Beschreibung IF SELECTION

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1		S	A			A	U							L	C			ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Auswahlfeld(er):** Als *operand1* geben Sie das bzw. die Felder an, die verifiziert werden sollen. Wenn Sie eine Kontrollvariable (Format C) angeben, so wird angenommen, dass sie einen Wert enthält, wenn ihr Status sich auf MODIFIED geändert hat. **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 *geändert* wurde.
THEN *statement* ...	**THEN-Klausel:** Das (oder die) in der THEN-Klausel angegebenen Statement(s) werden ausgeführt, wenn eine der folgenden Bedingungen erfüllt ist: ■ Keines der angegebenen Felder (*operand1*) enthält einen Wert. ■ Mehr als eines der angegebenen Felder (*operand1*) enthält einen Wert. 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. **Anmerkung:** Falls keine Maßnahme ausgeführt werden soll, wenn eine der beiden Bedingungen erfüllt ist, geben Sie das Statement IGNORE in der THEN-Klausel an.
ELSE *statement* ...	**ELSE-Klausel:** 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 *statement* DO *statement* ... DOEND	**Ende des IF-Statements:** Im Structured Mode muss das für Natural reservierte Wort IF SELECTION zum Beenden des IF-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um je nach Situation eines oder mehrere passende Statements anzugeben, und um die Klauseln und das IF SELECTION-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Beispiel für 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.
```

74 IGNORE

- Funktion IGNORE 572
- Beispiel für IGNORE-Statement 572

IGNORE

Dieser Abschnitt behandelt folgende Themen:

Funktion IGNORE

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 für IGNORE-Statement

```
...
...
AT TOP OF PAGE
   IGNORE         /* top-of-page processing still to be coded
END-TOPPAGE
...
...
```

75 INCLUDE

- Funktion INCLUDE 574
- Syntax-Beschreibung INCLUDE 574
- Beispiele INCLUDE 576

```
INCLUDE  copycode-name  [operand1 ... 99]
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion INCLUDE

Das Statement `INCLUDE` dient dazu, den Quellcode eines externen Objekts vom Typ Copycode bei der Kompilierung in ein anderes Objekt einzufügen.

Das `INCLUDE`-Statement wird bei der *Kompilierung* ausgewertet. Die Quellcode-Zeilen des Copycodes werden nicht physisch in den Quellcode des Programms eingefügt, das das `INCLUDE`-Statement enthält, und der eingefügte Copycode ist als Teil des Objektmoduls im kompilierten Programm enthalten.

Anmerkung: Eine Quellcode-Zeile, die ein `INCLUDE`-Statement enthält, darf kein anderes Statement enthalten.

Syntax-Beschreibung INCLUDE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C					A	U									nein	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
copycode-name	**Copycode-Name:** Als *copycode-name* geben Sie den Namen des Copycodes an, dessen Source eingefügt werden soll. Die Groß-/Kleinschreibung des Namens wird nicht verändert. Der *copycode-name* 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.

Syntax-Element	Beschreibung
	Das Objekt, dessen Namen Sie angeben, muss vom Objekttyp Copycode sein. Der Copycode muss entweder in derselben Library gespeichert sein wie das Programm, das das `INCLUDE`-Statement enthält, oder in der betreffenden Steplib (Standard-Steplib ist `SYSTEM`). Wenn der Quellcode 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. Der Quellcode des Copycodes muss aus syntaktisch vollständigen Statements bestehen.
operand1	**Dynamisch einzufügende Werte:** In den einzufügenden Copycode können Sie dynamisch Werte einsetzen. Diese Werte werden mit *operand1* angegeben. Im Copycode werden die Werte mit der folgenden Notation referenziert: `&n&` d.h. Sie markieren die Stelle, an der ein Wert eingesetzt werden soll, mit `&n&`. *n* ist die laufende Nummer des mit dem `INCLUDE`-Statement übergebenen Wertes. Zum Beispiel würde sich `&3&` auf den dritten übergebenen Wert beziehen. Für jede `&n&`-Notation im Copycode müssen Sie im `INCLUDE`-Statement einen Wert angeben. Wenn der Copycode beispielsweise `&5&` enthält, muss *operand1* mindestens fünfmal angegeben werden. Sie können einen Copycode-Parameter (`&n&`) hinter dem anderen ohne Leerzeichen (d.h. `&1&&2&&3&`) schreiben. Diese Methode wird zur Verkettung mehrerer Copycode-Parameter mit einer Source benutzt. Eine Zeichenkette kann einem oder mehreren Copycode- Parametern ohne ein Leerzeichen folgen (d.h. `&1&abc` oder `&1&&2&abc`). Diese Methode wird zur Verkettung einer Zeichenkette mit mehreren Copycode-Parametern benutzt. **Anmerkung:** Da `&n&` ein gültiger Teil eines Namens ist, darf diese Notation nicht als Ersatzzeichen für einen Copycode-Parameter in anderen, oben beschriebenen Positionen (d.h. `abc&1&` oder `&1&abc&2&`) benutzt werden. Mit anderen Worten kann eine Zeichenkette nur hinter Copycode-Parametern auftreten, nicht davor oder dazwischen. Mit dem `INCLUDE`-Statement angegebene Werte, die im Copycode nicht referenziert werden, werden ignoriert.

Beispiele INCLUDE

- 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
```

VIII INPUT

Dieser Abschnitt 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**

Siehe auch *Bildschirmgestaltung / Fenster* im *Leitfaden zur Programmierung*.

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 RPC: Siehe *Notes on Natural Statements on the Server* in der *Natural RPC (Remote Procedure Call)*-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 oder den Terminalkommandos %F und %D 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*
- *INPUT-Statement im Batch-Betrieb*
- *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 Masken-Editor (Map Editor) definiert werden.

Anmerkung: Unter UNIX können keine Auswahlfenster definiert werden. Auswahlfenster, die aus einer Großrechner- oder Windows-Umgebung importiert wurden, werden unter UNIX ignoriert.

Auswahlfenster können an Eingabefelder angehängt werden. Sie sind eine komfortable Alternative zu an Feldern angehängte Helproutinen, da Sie ja ein Auswahlfenster direkt in Ihrem Programm kodieren können. Sie brauchen kein zusätzliches Programm wie bei Helproutinen.

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.

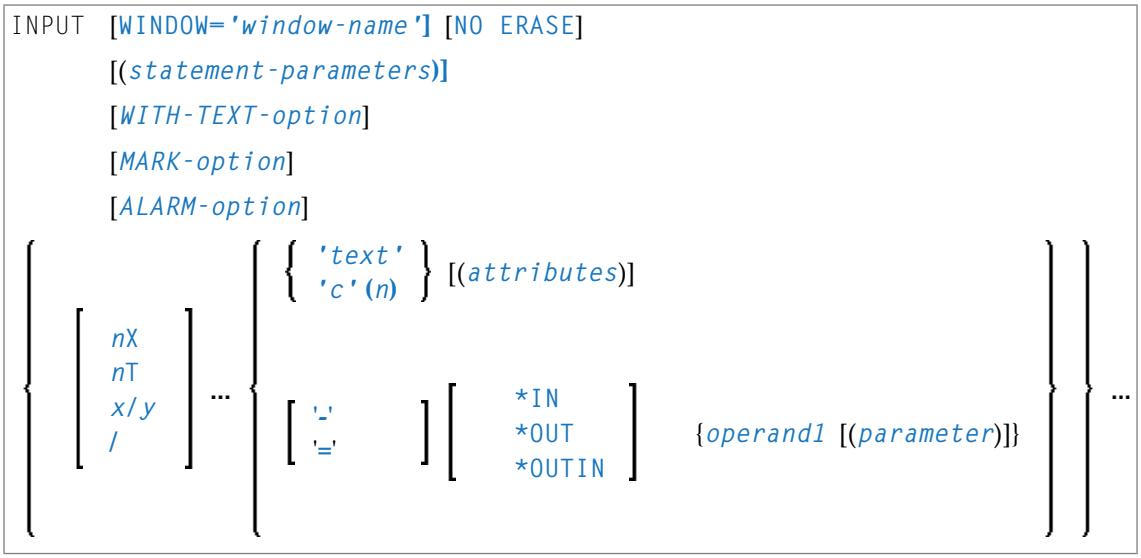
Systemvariablen beim INPUT-Statement

Zu Informationen über die relevanten Systemvariablen siehe Abschnitt *Eingabe/Ausgabebezogene Systemvariablen* in der *Systemvariablen*-Dokumentation.

76 INPUT-Syntax 1 - Dynamisch generierter Eingabeschirm

- INPUT Syntax 1 - Beschreibung 590
- Beispiele - Verwendung von Syntax 1 600

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.



Dieser Abschnitt 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
operand1		S	A	G	N	A	U	N	P	I	F	B	D	T	L		G		ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
INPUT WINDOW='*window-name*'	**Eingabe-Fenster:** 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.

Syntax-Element	Beschreibung
	Das angegebene Fenster ist nur für die Dauer des betreffenden INPUT-Statements aktiv und wird nach Ausführung des INPUT-Statements automatisch deaktiviert. Siehe auch die Statements DEFINE WINDOW und SET WINDOW.
NO ERASE	**Überlagerte Anzeige:** NO ERASE bewirkt, dass die vom INPUT-Statement ausgegebene Schirmanzeige eine bereits vorhandene Anzeige überlagern kann, ohne letztere zu löschen. Schirm bezieht sich in diesem Zusammenhang auf die logische Ausgabe und nicht auf den physischen Bildschirm. 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.

statement-parameters	**Parameter auf Statement-/Feldebene:**
	
	Unmittelbar nach dem Schlüsselwort INPUT oder nach einem der auszugebenden Felder können Sie in Klammern einen oder mehrere Session-Parameter setzen.
	
	Eine Liste der mit dem INPUT-Statement anzugebenden Parameter finden Sie im Abschnitt *Statement-Parameter* weiter unten.
	
	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 Quellcode-Zeilen erstrecken.
	
	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.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 VARI (A4)  INIT <'1234'>  /*  Displays`
	`END-DEFINE  /*  as`
	`FORMAT AD=M  /*  ---------`
	`INPUT  'Text'  VARI  /*  Text 1234`
	`INPUT (PM=I)  'Text'  VARI  /*  Text 4321`
	`INPUT  'Text' (PM=I)  VARI (PM=I)  /*  txeT 4321`
	`INPUT  'Text' (PM=I)  VARI  /*  txeT 1234`
	`END`

Syntax-Element	Beschreibung
	Beispiele für den Einsatz von Parametern auf Statement- und Element-Ebene finden Sie auf den folgenden Seiten.
WITH TEXT-option	**Textausgabe in Meldungszeile:** Diese Option dient dazu, Text anzugeben, der in der Meldungszeile ausgegeben werden soll (siehe den entsprechenden Abschnitt weiter **unten**).
MARK-option	**Cursorposition im Feld:** Siehe Abschnitt ***MARK-Option*** weiter unten.
ALARM-option	**Warntonausgabe:** Siehe Abschnitt ***Alarm-Option*** weiter unten.
Other syntax elements (*nX*, *nT*, *x/y*, *operand1* 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
BX	Box Definition	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

Parameter, die mit dem INPUT-Statement angegeben werden können:		Spezifikation S=auf Statement-Ebene E=auf Element-Ebene
SB	Selection Box	E
SG	Sign Position	SE
ZP	Zero Printing	SE

* Wenn die Anzahl der Ausprägungen eines Arrays den `PS`-Wert überschreitet, wird ein NAT0303-Fehler ausgegeben.

Beschreibungen der einzelnen Session-Parameter sind in der *Parameter-Referenz* enthalten.

WITH TEXT-Option

```
[WITH] TEXT { * operand1 } [(attributes)][,operand3] ... 7
            { operand2   }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S					N	P	I		B*						ja	ja
operand2	C	S				A											ja	ja
operand3	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.

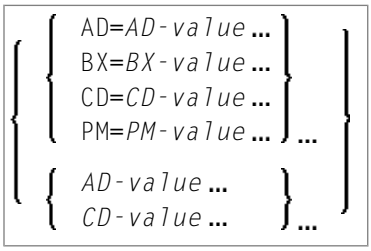
Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Meldungstext aus der Natural-Fehlermeldungsdatei:** Als *operand1* geben Sie eine Natural-Fehlernummer an. Natural liest dann die entsprechende Fehlermeldung von der Natural- Fehlermeldungsdatei. Es können entweder benutzerdefinierte Meldungen oder Natural-Systemmeldungen gelesen werden. ■ Wenn Sie einen positiven Wert von bis zu vier Ziffern (zum Beispiel: `0954`) angeben, werden benutzerdefinierte Meldungen gelesen.

Syntax-Element	Beschreibung
	■ Wenn Sie einen negativen Wert von bis zu vier Ziffern (zum Beispiel `-0954`) angeben, werden Natural-Systemmeldungen gelesen. Siehe auch `REINPUT`-Statement, ***Beispiel 4 - WITH TEXT-Optionen***. Natural-Fehlermeldungsdateien werden mit der `SYSERR`-Utility erstellt und gepflegt. Die Natural-Fehlermeldungen sind in der *Messages and Codes*-Dokumentation enthalten.
operand2	**Eigener Meldungstext:** Als *operand2* geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll. Siehe `REINPUT`-Statement, ***Beispiel 4 - WITH TEXT-Optionen***.
attributes	**Ausgabeattribute:** Als *attributes* können Sie *operand1* oder *operand2* bestimmte Anzeige- und Farbattribute zuordnen. Diese Attribute und die benutzbare Syntax sind im Abschnitt *Ausgabeattribute* weiter unten beschrieben.
operand3	**Dynamische Meldungstext-Komponente:** *operand3* kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden. Der entweder mit *operand1* oder *operand2* angegebene Wert dient dazu, einen Teil der Meldung zu ersetzen und dynamisch zu generieren. Innerhalb der Fehlermeldung dient die Notation `:`*n*`:` zur Referenzierung von *operand3*, wobei n die Ausprägung (`1 - 7`) von *operand3* darstellt. Siehe `REINPUT`-Statement, ***Beispiel 4 - WITH TEXT-Optionen***. **Anmerkung:** 1. Wird *operand3* 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 *operand3* 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 *operand3* 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*
- *BX - Box 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.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand4	C	S				N	P	I								ja	ja
operand1	C	S	A			N	P	I								ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Feldnummer für Referenzierung:** *operand1* gibt die Nummer des Feldes an, in dem der Cursor platziert werden soll: 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.
**fieldname*	**Feldname für Referenzierung:** Anstelle der Feldnummer können Sie den Feldnamen angeben, um den Cursor in ein bestimmtes Feld zu platzieren. Dazu verwenden Sie die Notation **fieldname*.
operand4	**Cursorposition im referenzierten Feld:** Mit MARK POSITION können Sie den Cursor an eine bestimmte Stelle - die Sie mit *operand4* angeben - innerhalb des mit *operand1* oder **fieldname* angegebenen Feldes platzieren. *operand4* darf keine Dezimalstellen enthalten.

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

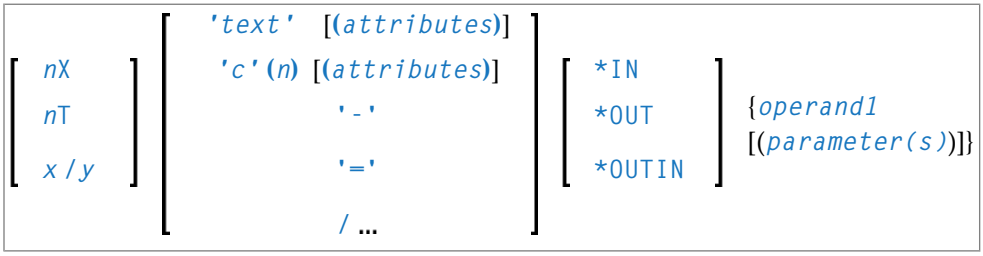
Diese Option bewirkt, dass der Warnton des Terminals ausgelöst wird, wenn das INPUT-Statement ausgeführt wird. Voraussetzung ist, dass die verwendete Terminal-Hardware dies ermöglicht.

```
[[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



Verschiedene Notationen stehen zur Feldpositionierung, Texterstellung und Attributzuweisung zur Verfügung.

Syntax-Element-Beschreibung:

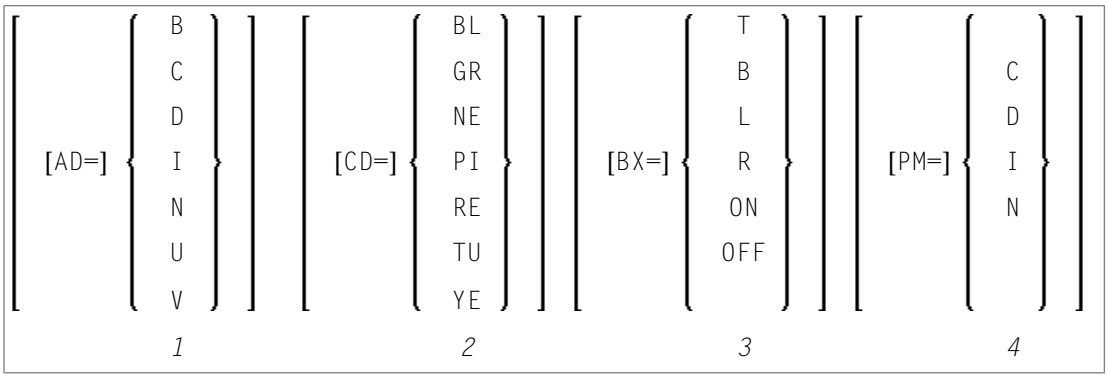
Syntax-Element	Beschreibung
nX	**Leerstellen zwischen Spalten:** Fügt zwischen den Feldern *n* Leerstellen ein. **Anmerkung:** Diese Notation fügt zwischen den Spalten *n* Leerstellen ein. *n* darf nicht `0` sein.
nT	**Tabulator für Feldausgabe:** Setzt einen Tabulator, d.h. die Ausgabe eines Feldes 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 *Text-Notation*, Unterabschnitt *Mit einem Statement zu benutzenden Text definieren*.
`'c' (n)`	**Wiederholungszeichen:** Wie `'text'`. Aber das Zeichen *c* (character) wird *n*-mal ausgegeben. *n* darf `1 - 132` sein. Siehe auch den Abschnitt *Text-Notation*, Unterabschnitt *Vor einem Feldwert n mal anzuzeigendes Zeichen definieren.*.
attributes	**Ausgabeattribute:** Dient zum Setzen der Anzeigeattribute. Siehe ***Attribute*** weiter unten.
`'-'`	**Unterdrückung des Feldnamens:** Die Notation `'-'` unmittelbar vor einem Feld bewirkt, dass die Anzeige des Feldnamens vor dem Feld unterdrückt wird.

Syntax-Element	Beschreibung
	Anmerkung: Eine Textkette vor einem Feld ersetzt den Feldnamen als Eingabeaufforderungstext.
`'='`	**Feldüberschrift-Ausgabe:** Die Notation `'='` unmittelbar vor einem Feld bewirkt, dass unmittelbar vor dem Feldwert die Feldüberschrift ausgegeben wird.
`'/'`	**Zeilenvorschub:** Ein Schrägstrich `'/'` zwischen zwei Feldern oder Textelementen bewirkt einen Zeilenvorschub, d.h. das nachfolgende Element wird in der nächsten Zeile ausgegeben. Felder können als reine Eingabefelder (Session-Parameter `AD=A`), reine Ausgabefelder (`AD=O`) 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. 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. Beispiel: `INPUT #A (AD=A) #B (AD=O) #C (AD=M)` `#A` ist ein Eingabefeld (ungeschützt), in das ein Wert eingegeben werden kann. `#B` ist ein Ausgabefeld (schreibgeschützt), dessen angezeigter Wert nicht überschrieben werden kann. `#C` ist ein Feld, dessen ausgegebener Wert verändert werden kann, indem er durch einen neuen Wert überschrieben wird.
`*IN, *OUT und *OUTIN`	**Feldattribute:** Entspricht den Feldattributen, die mit dem Session-Parameter `AD` gesetzt werden können: `AD=A`, `AD=O` bzw. `AD=M`. **Anmerkung:** Wenn eine nicht änderbare Systemvariable in einem `INPUT`-Statement benutzt wird, wird der Wert als ein reines Ausgabefeld `AD=O` oder `*OUT`-Attribut anzeigt.
operand1	**Zu verwendende Felder:** Als *operand1* geben Sie das zu verwendende Feld an. Sie können Datenbankfelder oder Benutzervariablen angeben. Natural überträgt den Inhalt eines Feldes direkt vom Datenbereich an das `INPUT`-Statement; eine `MOVE`-Operation ist hierzu nicht erforderlich.

Syntax-Element	Beschreibung
	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. Wird in einem INPUT-Statement der Name einer Gruppe von Datenbankfeldern referenziert, so werden alle in der Gruppe enthaltenen Felder einzeln als Eingabefelder verwendet. 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. Auf Großrechnern können keine Arrays mit Bereichen angegeben werden, die es ermöglichen, zur Ausführungszeit die Anzahl der Ausprägungen zu variieren.
parameter(s)	**Statement-Parameter:** Unmittelbar nach *operand1* können Sie in Klammern einen oder mehrere Parameter angeben (siehe Tabelle ***Statement-Parameter*** und folgendes Beispiel). 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 Quellcode-Zeilen erstrecken. 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. Informationen zu den einzelnen Parametern entnehmen Sie der Tabelle im Abschnitt ***Statement-Parameter***. **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.

Ausgabeattribute

Die folgenden Attribute können verwendet werden:



1. Anzeigeattribute - siehe Session-Parameter `AD` in der *Parameter-Referenz*.
2. Farbattribute - siehe Session-Parameter `CD` in der *Parameter-Referenz*.
3. Umrandungen (Box Definition) - siehe Session-Parameter `BX` in der *Parameter-Referenz*.
4. Print-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[]
```

77 INPUT-Syntax 2 - Verwendung einer vordefinierten Eingabemaske

- INPUT USING MAP ohne Parameterliste 604
- Im Programm definierte Eingabefelder 605
- INPUT Syntax 2 - Beschreibung 605
- INPUT-Statement unter Nicht-Screen-Modi 607
- Eingabedaten aus dem Natural-Stack 609
- INPUT-Statement im Batch-Betrieb 610

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     ]
```

Dieser Abschnitt 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 dem Quellcode der Map 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 Maske (Map) verwendeten Feldnamen übereinstimmen.

Die Reihenfolge der Felder im Programm muss allerdings zur Reihenfolge der Felder in der Maske passen. Hierbei ist zu beachten, dass der Masken-Editor die in der Maske definierten Felder in alphabetischer Reihenfolge der Feldnamen sortiert. Näheres hierzu finden Sie in der Masken-Editor-Beschreibung in der *Editoren*-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
map-name	C	S				A	U											ja	nein
operand1		S	A			A	U	N	P	I	F	B	D	T	L	C		ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`INPUT WINDOW='window-name'`	**Eingabe-Fenster:** Siehe ***Syntax 1*** des `INPUT`-Statements.
`WITH TEXT/MARK/ALARM-options`	**Optionen:** Diese Optionen sind unter ***Syntax 1*** des `INPUT`-Statements beschrieben; siehe ***WITH TEXT-Option***, ***MARK-Option***, ***ALARM-Option***.
`USING MAP` *`map-name`*	**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 *`map-name`* 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. Die Groß-/Kleinschreibung des Namens wird nicht verändert. Der Map-Name darf ein Und-Zeichen (&) enthalten; dies wird dann zur Ausführungszeit durch 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.
`NO ERASE`	**Überlagerte Anzeige:** Siehe ***Syntax 1*** des `INPUT`-Statements.
`operand1`	**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.

INPUT-Statement unter Nicht-Screen-Modi

Sie können den Eingabemodus mit dem Session-Parameter IM oder den Terminalkommandos %F und %D ändern.

Forms-Modus

Der Forms-Modus wird mit dem Terminalkommando %F eingeschaltet.

Im Forms-Modus (Profil/Session-Parameter IM=F) zeigt Natural den gesamten Ausgabetext 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

Der Keyword/Delimiter-Modus wird mit dem Terminalkommando %D eingeschaltet.

In diesem Modus (Profil/Session-Parameter IM=D) können Daten unter Verwendung von Schlüsselwörtern oder in Abhängigkeit von der Position/Reihenfolge der Felder eingegeben werden.

Allgemeine Validierungsregeln

Daten, die im Keyword/Delimiter-Modus eingegeben werden, werden wie beim Screen-Modus auf Gültigkeit geprüft. Falls versucht wird, mehr Zeichen einzugeben, als für ein Feld definiert wurden, wird eine Fehlermeldung zurückgegeben.

Wenn das INPUT-Statement auf einem gepufferten Terminal (Typ 3270) oder einer Workstation im Keyword/Delimiter-Modus verarbeitet werden soll, müssen alle Daten, die einem INPUT-Statement zugewiesen werden sollen, auf demselben Bildschirm eingegeben werden. ENTER soll erst dann benutzt werden, nachdem alle Daten für das INPUT-Statement eingegeben worden sind.

Dateneingabe unter Verwendung von Schlüsselwörtern

Bei der Dateneingabe unter Verwendung von Schlüsselwörtern kann der Terminal-Operator Daten für die einzelnen Felder eingeben, indem er den Eingabeaufforderungstext als Schlüsselwort (Keyword) benutzt, der im Forms-Modus vor einem Feld ausgegeben werden würde, um das betreffende Feld zu identifizieren. Nach dem Schlüsselwort muss das Eingabezuweisungszeichen stehen (siehe Session-Parameter IA) und unmittelbar danach die Eingabedaten. Alle Stellen nach dem Eingabe-Zuweisungszeichen bis hin zum Eingabebegrenzungszeichen (siehe Session-Para-

meter ID) werden als Daten genommen. Nach dem letzten Datenelement braucht kein Eingabebegrenzungszeichen zu stehen. Schlüsselwort-Dateneingaben für die einzelnen Felder müssen durch das Eingabebegrenzungszeichen voneinander getrennt sein und können in beliebiger Reihenfolge stehen. Falls der Terminal-Operator ein Schlüsselwort eingibt, das nicht im INPUT-Statement definiert ist, wird eine entsprechende Fehlermeldung zurückgegeben. 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.

Ein Schlüsselwort und das entsprechende Eingabefeld müssen sich in derselben logischen Zeile befinden. Wenn ihre Gesamtlänge länger als die Zeilenlänge ist, passen Sie die Zeilenlänge (siehe Parameter LS) so an, dass Schlüsselwort und Feld in eine Zeile passen.

Dateneingabe unter Verwendung eines Indexes

Bei der Dateneingabe unter Verwendung eines Indexes kann der Terminal-Operator Daten für die einzelnen Felder eingeben, indem er ihre ordinalen Werte mit einem Prozentzeichen (%) als Präfix benutzt. Nach der Indexangabe muss das Eingabezuweisungszeichen stehen (siehe Session-Parameter IA) und unmittelbar danach die Eingabedaten.

Indizierte Daten für die verschiedenen Felder können in beliebiger Reihenfolge stehen. Sie müssen durch Eingabebegrenzungszeichen (siehe Session-Parameter ID) voneinander getrennt sein. Falls der angegeben ordinale Wert keinem der Werte eines existierenden Feldes entspricht, wird eine entsprechende Fehlermeldung zurückgegeben. 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.

Dateneingabe in Abhängigkeit von der Position der Felder

Bei der Dateneingabe in Abhängigkeit von der Position der Felder gibt der Terminal-Operator allein die Daten für die einzelnen Felder ein. Diese müssen durch Eingabebegrenzungszeichen (siehe Session-Parameter ID) voneinander getrennt sein. Die Reihenfolge der Felder bei der Dateneingabe muss der Reihenfolge der Eingabefelder im INPUT-Statement entsprechen.

Der Benutzer kann vom positionsgebundenen Modus in den Schlüsselwortmodus wechseln, indem er zunächst eine Anzahl von durch Eingabebegrenzungszeichen voneinander getrennten Werten im positionsgebundenen Modus eingibt und dann bei ausgewählten Feldern durch Angabe des jeweiligen Schlüsselworts vor den Werten in den Schlüsselwortmodus wechselt. Nachdem zum Positionieren eines Feldes ein Schlüsselwort benutzt worden ist, werden alle nicht auf ein Schlüsselwort bezogenen Eingaben, die auf das Schlüsselwort folgen, als positionsgebundene Eingaben verarbeitet, die zu den Feldern nach dem zuvor ausgewählten Feld im INPUT-Statement folgen.

Beispiel für Schlüsselworteingabe, Index-Eingabe und positionsgebundene Eingabe

Wenn Sie das folgende Programm aus der Kommandozeile ausführen

```
***** Program PGM1 *****
DEFINE DATA LOCAL
1 #F1 (A10)
1 #F2 (A10)
1 #F3 (A10)
END-DEFINE
INPUT (IP=ON) /  'FLD1' #F1
              /  'FLD2' #F2
              /  'FLD3' #F3
WRITE 'FLD1' #F1
    / 'FLD2' #F2
    / 'FLD3' #F3
END
```

und dabei eines der folgenden Kommandos benutzen, wobei das Komma (,) als Eingabebegrenzungszeichen benutzt wird,

`PGM1 FLD1=AA,FLD3=CC`	Schlüsselworteingabe
`PGM1 %1=AA,%3=CC`	Indexeingabe
`PGM1 AA,,CC`	positionsgebundene Eingabe
`PGM1 AA,FLD3=CC`	positionsgebundene Eingabe kombiniert mit Schlüsselworteingabe
`PGM1 AA,FLD2=,CC`	positionsgebundene Eingabe kombiniert mit Schlüsselworteingabe
`PGM1 AA,%3=CC`	positionsgebundene Eingabe kombiniert mit Indexeingabe

dann erhalten Sie immer die folgende Ausgabe:

```
FLD1 AA
FLD2
FLD3 CC
```

Eingabedaten aus dem Natural-Stack

Daten, die mittels eines `FETCH`-, `RUN`- oder `STACK`-Statements auf dem 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 Natural-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.

INPUT-Statement im Batch-Betrieb

Folgende Themen werden behandelt:

- Forms-Modus im Batch-Betrieb
- Keyword/Delimiter-Modus im Batch-Betrieb
- Terminalkommandos im Batch-Betrieb

Forms-Modus im Batch-Betrieb

Im Batch-Forms-Modus wird die Eingabe-Map angezeigt. Für jede Zeile, die ein oder mehrere Eingabefelder (`AD=A` oder `AD=M`) enthält, wird ein Datensatz gelesen, und die in dem Satz enthaltenen Daten werden den entsprechenden Feldern zugeordnet.

Eingabedatenfelder werden als benachbart angesehen. Werden keine Delimiter-Zeichen verwendet, so müssen die Daten genau den intern definierten Feldlängen entsprechend eingegeben werden. Bei numerischen Feldern müssen gegebenenfalls eine Stelle für das Vorzeichen (falls `SG=ON` gesetzt ist) und/oder eine Stelle für das Komma (Dezimalpunkt) berücksichtigt werden.

Werden Delimiter-Zeichen verwendet, um die Eingabedaten der einzelnen Felder voneinander zu trennen, so ist dies nicht erforderlich; die Eingabedaten werden dann jeweils von links nach rechts ab Stelle 1 verarbeitet. Für die Eingabe der Daten gelten die unter *Eingabe von Daten als Reaktion auf ein INPUT-Statement* beschriebenen Regeln. Darüber hinaus kann mit dem Zuweisungszeichen für Eingabe-Parameter (siehe Profil- und Session-Parameter `IA`) bewirkt werden, dass der Inhalt eines `*OUTIN`-Feldes nicht zurückgesetzt wird.

Keyword/Delimiter-Modus im Batch-Betrieb

Im Batch-Betrieb gilt für diesen Modus dasselbe wie im TP-Betrieb, allerdings mit folgenden Ausnahmen:

- Das Drucken der gesamten Eingabemaske kann über das Terminalkommando `%Q` gesteuert werden.
- `*OUTIN`-Felder behalten ihre ursprünglichen Werte, wenn diese nicht explizit geändert werden.

Terminalkommandos im Batch-Betrieb

Folgende Terminalkommandos können verwendet werden, wenn ein INPUT-Statement im Batch-Betrieb auf einem Großrechner verwendet wird:

Terminalkommando	Funktion

%*	**Unterdrückung des nächsten Eingabedatensatzes:**
	
	Wenn in Position 1 oder 2 eines Datensatzes eingegeben, bewirkt %* die Unterdrückung der Ausgabe des nächsten Eingabedatensatzes:
	
	`DATA RECORD`
	`%*`
	`SUPPRESSED DATA RECORD`

%	**Fortsetzung des Datensatzes:**
	
	Als letztes Zeichen eines Datensatzes bewirkt %, dass der nachfolgende Eingabedatensatz als Fortsetzung des aktuellen Datensatzes interpretiert wird.
	
	`DATA, RECORD, WITH,`
	`CONTINUATION, %`
	`CONTINUATION RECORD`
	
	`INPUT V1 V2 V3 V4 V5 V6`
	`DISPLAY V1 V2 V3 V4 V5 V6`
	
	erzeugt folgende Ausgabe:
	
	`DATA RECORD WITH CONTINUATION CONTINUATION RECORD`

%/	**Dateiende-Bedingung:** %/ bewirkt eine Dateiende-Bedingung (End-of-File), wenn es in den ersten zwei Positionen eines Eingabedatensatzes (ohne nachfolgende Nicht-Leerzeichen) eingegeben wird.
%%	**Restart-Punkte setzen:** Sie können mit dem Terminalkommando %% Restart-Punkte in den Eingabedateien setzen und so die Synchronisation der Eingabedateien im Falle eines Fehlers sicherstellen.
%.	**Lesen der Eingabewerte beenden:** Lesen der Eingabewerte für das gerade ausgeführte INPUT-Statement wird beendet.
%K*nn*	**Simulieren von PF-Tasten bzw. PA-Tasten:**

Terminalkommando	Funktion
%KP*n*	Dadurch können PF-Tastenfunktionen auch im Batch-Betrieb verwendet werden.
%Q	**Maskenausgabe im Batch-Betrieb unterdrücken:** Bewirkt, dass Masken, die zum Lesen von Eingabedaten verwendet werden, nicht mit ausgedruckt werden.

Ausführliche Informationen zu den oben genannten Kommandos finden Sie in der *Terminalkommandos*-Dokumentation.

Zur Verwendung des INPUT-Statements im Batch-Betrieb ist zusätzliche JCL erforderlich. Bitte wenden Sie sich an Ihren Natural-Administrator um sicherzustellen, dass diese JCL bereitgestellt ist, bevor Sie eine Batch-Ausführung versuchen.

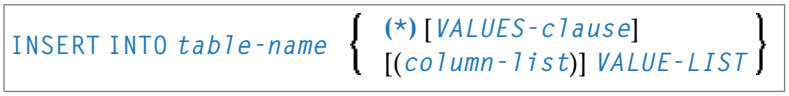
IX

- 78 INSERT (SQL) 615
- 79 INTERFACE 625
- 80 LIMIT 633
- 81 LOOP 637
- 82 MERGE (SQL) 641
- 83 METHOD 651
- 84 MOVE 657
- 85 MOVE INDEXED 687
- 86 MULTIPLY 689
- 87 NEWPAGE 695
- 88 OBTAIN 701
- 89 ON ERROR 711
- 90 OPEN CONVERSATION 717
- 91 OPTIONS 721

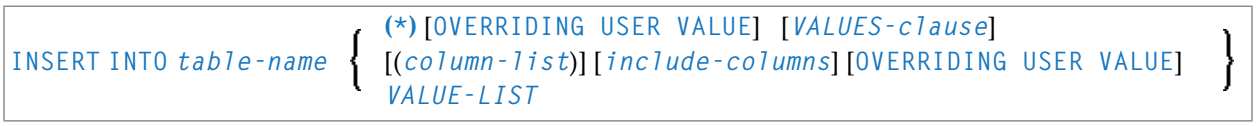
78 INSERT (SQL)

- Funktion INSERT (SQL) 616
- Syntax-Beschreibung INSERT (SQL) 616
- Beispiel für INSERT (SQL) 622

Common Set-Syntax:



Extended Set-Syntax:



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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *INSERT - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion INSERT (SQL)

Das SQL-`INSERT`-Statement dient dazu, in einer Tabelle eine oder mehrere neue Zeilen hinzuzufügen.

Syntax-Beschreibung INSERT (SQL)

Syntax-Element	Beschreibung
`INTO table-name`	**INTO-Klausel:** In der `INTO`-Klausel geben Sie an, in welcher Tabelle neue Zeilen hinzugefügt werden sollen. Siehe weitere Informationen zu *table-name*.
column-list	**Spaltenliste:** Syntax:

Syntax-Element	Beschreibung
	column-name... ↩ Als *column-list* können Sie *column-names* für eine oder mehrere Spalten angeben, die in der hinzugefügten Zeile Werte erhalten sollen. Bei Angabe einer *column-list* muss die Reihenfolge der angegebenen Spalten der Reihenfolge der Werte entsprechen, die entweder in der *insert-item-list* angegeben sind oder im angegebenen View enthalten sind (siehe unten). Wenn Sie bei *column-list* keine Spaltenliste angeben, werden die in der *insert-item-list* bzw. im View angegebenen Werte entsprechend einer impliziten Liste aller Spalten eingefügt, und zwar in der Reihenfolge, in der sie in der Tabelle stehen.
include-columns	**Include Columns-Klausel:** *include-columns* gibt zusätzlich zu den *table-name*-Spalten einen Satz Spalten in der Ergebnistabelle des INSERT-Statement an, wenn dieses in der FROM-Klausel eines SELECT-Statement verschachtelt ist. Weitere Informationen siehe ***include-columns***.
VALUES-clause	**VALUES-Klausel:** Mit der VALUES-Klausel fügen Sie eine *einzelne* Zeile in die Tabelle ein. Siehe ***VALUES-Klausel*** weiter unten.
OVERRIDING USER VALUE	**OVERRIDING USER VALUE-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese Klausel bewirkt, dass der Wert ignoriert wird, der in der VALUES-Klausel angegeben ist oder von einem Fullselect für eine Spalte erzeugt wurde, die als GENERATED ALWAYS definiert worden ist.

VALUES-Klausel

Mit der VALUES-Klausel fügen Sie eine *einzelne* Zeile in die Tabelle ein. Der VALUES-Klausel kann entweder ein Stern (*) oder eine Spaltenliste (*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 benutzen, müssen Sie in der VALUES-Klausel einen View angeben. Mit den Feldwerten des Views wird dann eine neue Zeile in die Tabelle eingefügt, wobei die Feldnamen des Views als Spaltennamen der Zeile verwendet werden.

VALUES-Klausel mit vorangehender Spaltenliste

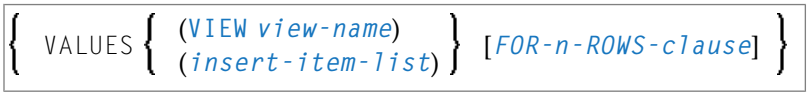
```
[(column-list)] [OVERRIDING USER VALUE] VALUE-LIST
```

Wenn Sie bei *column-list* eine Spaltenliste angeben und in der VALUES-Klausel eine View referenzieren, muss die Anzahl der Spalten in der Spaltenliste der Anzahl der Felder im View innerhalb der VALUE-LIST entsprechen.

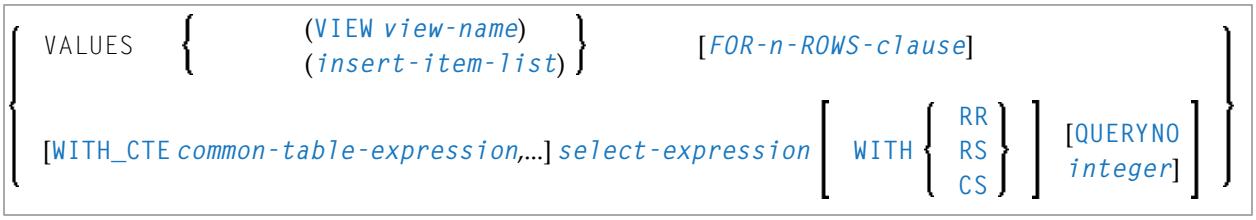
Wenn Sie keine Spaltenliste angeben, werden die im View angegebenen Werte entsprechend der impliziten Liste aller Spalten in der Reihenfolge, in der sie in der Tabelle stehen, eingefügt.

VALUE-LIST

Common Set-Syntax:



Extended Set-Syntax:



Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
VIEW *view-name*	**View-Name:** Mit den Feldwerten dieses Views wird eine neue Zeile in die angegebene Tabelle eingefügt, wobei die Feldnamen des Views als Spaltennamen der Zeile benutzt werden.
insert-item-list	**INSERT Single Row- oder Multiple Row-Klausel:**

Syntax-Element	Beschreibung

	In der *insert-item-list* können Sie einen oder mehrere Zeilenwerte angeben, die den in der *column-list* angegebenen Spalten zugeordnet werden sollen. Die Reihenfolge der angegebenen Werte muss mit der Reihenfolge der Spalten übereinstimmen. Mehrere Zeilen werden in Klammern gesetzt und durch Kommas voneinander getrennt.
	
	Wenn keine *column-list* angegeben wird, werden die Werte in der *insert-item-list* nach einer impliziten Liste mit allen Spalten in der Reihenfolge eingefügt, wie sie in der Tabelle vorkommen.
	
	Die in der *insert-item-list* anzugebenden Werte können Konstanten, Parameter, spezielle Register oder NULL sein.
	
	Informationen zu *view-name*, *constant* und *parameter* siehe ***Grundlegende Syntaxbestandteile*** . Siehe auch die Informationen zu *special-register*.
	
	Wenn der Wert NULL zugewiesen worden ist, bedeutet dies, dass das adressierte Feld keinen Wert erhalten soll (auch nicht den Wert 0 oder Leerzeichen).
	
	Beispiel für INSERT Single Row:
	
	`...`
	`INSERT INTO SQL-PERSONNEL (NAME,AGE)`
	`  VALUES ('ADKINSON',35)`
	`...`
	
	Beispiel für INSERT Multiple Row:
	
	`...`
	`INSERT INTO SQL-PERSONNEL (NAME,AGE)`
	`   VALUES (‘ADKINSON’,35),`
	`  (‘ANDERSON’,’45),`
	`  (‘MILLER’,34),`
	`  (‘SIMONE’,56),`
	`  (‘BECKER’,27)`
	`...`

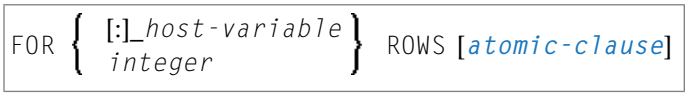
FOR-n-ROWS-clause	**FOR *n* Rows-Klausel:** Optionale Klausel, siehe ***FOR-n-ROWS-Klausel*** weiter unten.
WITH_CTE *common-table-expression*	**WITH_CTE-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese optionale Klausel ermöglicht die Definition einer Ergebnistabelle, die in einer FROM-Klausel des folgenden SELECT-Statements referenziert werden kann. Mehrere *common-table-expressions* können nach dem einzelnen Schlüsselwort WITH_CTE angegeben werden. Jeder

Syntax-Element	Beschreibung
	common-table-expressions kann auch in der FROM-Klausel der nachfolgenden *common-table-expressions* referenziert werden. Weitere Informationen siehe ***WITH_CTE common-table-expression*** beim SELECT-Statement.

select-expression	**INSERT Multiple Rows-Klausel:**
	
	Diese Klausel gehört zum **SQL Extended Set**.
	
	Mit einem *select-expression* können Sie *mehrere* Zeilen in eine Tabelle einfügen. Der *select-expression* wird ausgewertet und jede Zeile der Ergebnistabelle wird so behandelt, als ob die Werte in der Zeile als Werte in einer VALUES-Klausel einer Single-Row-Insert-Operation angegeben werden.
	
	Weitere Informationen siehe ***Select-Ausdrücke***.
	
	Beispiel für Insert Multiple Rows:
	
	`...`
	`INSERT INTO SQL-RETIREE (NAME,AGE,SEX)`
	`  SELECT LASTNAME, AGE, SEX`
	`  FROM SQL-EMPLOYEES`
	`  WHERE AGE > 60`
	`...`
	
	Anmerkung: Die Anzahl der tatsächlich eingefügten Zeilen können Sie mit der Systemvariablen *ROWCOUNT überprüfen (siehe *Systemvariablen*-Dokumentation).

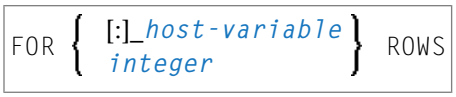
WITH RR/RS/CS	**WITH Isolation Level-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese Klausel ermöglicht die explizite Angabe des zum Auffinden der einzufügenden Zeilen benutzten Isolationsstufe. CS — Cursor Stability RR — Repeatable Read RS — Read Stability
QUERYNO_*integer*	**QUERYNO-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Diese Klausel gibt explizit die bei der EXPLAIN-Ausgabe und zur Ablaufverfolgung der Sätze für dieses Statement zu benutzende Anzahl an.

FOR-*n*-ROWS-Klausel



Diese Klausel setzt sich aus den folgenden Subklauseln zusammen:

FOR [:] *hostvariable*/*integer* ROWS-Klausel



Die Angabe dieser Klausel ist optional. Sie sollte nur angegeben werden, wenn

- die Compiler-Option DB2ARRY angegeben wird und
- mehrere Zeilen aus 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 Zeilen 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 Zeilen aus Natural-Arrays in einer Schleife eingefügt werden. Anhand dieser Klausel können die in den Arrays enthaltenen Zeilen von einem SQL-Statement eingefügt werden.

Siehe Beispiel weiter unten.

Siehe auch den Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

ATOMIC-Klausel



Diese Klausel gibt an, ob die Einfügung mehrerer Zeilen von Db2 als ATOMIC-Operation behandelt werden soll oder nicht.

Sie sollte nur angegeben werden, wenn

- die Compiler-Option DB2ARRY angegeben wird und
- mehrere Zeilen aus Arrays eingefügt werden sollen, die in der *insert-item-list* der ***VALUES-Klausel*** angegeben worden sind.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ATOMIC	Gibt an, dass im Falle eines Fehlers keine Zeile in die Zieltabelle eingefügt wird. Dies ist der Standardwert.
NOT ATOMIC CONTINUE ON SQLEXCEPTION	Gibt an, dass im Falle von Fehlern alle Zeilen eingefügt werden, bei denen keine Fehler aufgetreten sind, während diejenigen Zeilen, bei denen Fehler aufgetreten sind, von Db2 entfernt werden.

Die in solchen Fällen zurückgegebenen sqlcodes entnehmen sie der *Db2 SQL REFERENCE*-Dokumentation.

Beispiel für INSERT (SQL)

```
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
```

79 INTERFACE

- Funktion INTERFACE 626
- Syntax-Beschreibung INTERFACE 627

```
INTERFACE  interface-name
[property-definition]
[method-definition]
END-INTERFACE
```

Dieser Abschnitt 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 INTERFACE

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 INTERFACE

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
interface-name	**Interface-Name:** 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 *Namenskonventionen für Benutzervariablen*. Er muss pro Klasse eindeutig und nicht mit dem Klassen-Namen identisch sein. 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.
property-definition	**Property-Definition für das Interface:** Die *property-definition* dient zur Definition von Eigenschaften für das Interface. Siehe *Property-Definition* weiter unten.
method-definition	**Method-Definition für das Interface:** Die *method-definition* dient zur Definition einer Method für das Interface. Siehe ***Method-Definition*** weiter unten.
END-INTERFACE	Das für Natural reservierte Wort END-INTERFACE muss zum Beenden des INTERFACE-Statements benutzt werden.

Property-Definition

Die Property-Definition dient zur Definition von Eigenschaften für das Interface.

```
PROPERTY property-name
[(format-length/array-definition)]
[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 Method aufzurufen, die den Wert zurückgibt, und eine andere Method 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.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
property-name	**Property-Name:** 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 *Namenskonventionen für Benutzervariablen*). 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.
format-length/array-definition	**Property-Format:** Damit wird das Format der Property definiert, wie es von den Clients erkannt wird. Wenn *format-length/array-definition* weggelassen wird, wird *format-length* und *array-definition* aus der in der IS-Klausel zugewiesenen Objektdaten-Variable genommen. Wenn *format-length/array-definition* angegeben wird, muss diese Angabe unbedingt datenübertragungskompatibel sein, und zwar zu dem und von dem in *operand* in der IS-Klausel angegebenen Format der Objektdaten-Variable. 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. 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.

Syntax-Element	Beschreibung
READONLY	**Schreibschutz für Property-Wert:** Wenn dieses Schlüsselwort angegeben wird, kann der Wert der Property nur gelesen und nicht gesetzt werden. Das in *operand* in der IS-Klausel angegebene Format der Objektdaten-Variable muss datenübertragungskompatibel mit dem in *format-length/array-definition* angegebenen Format sein. Es muss nicht datenübertragungskompatibel in der umgekehrten Richtung sein. Wenn das Schlüsselwort READONLY weggelassen wird, kann der Property-Wert sowohl gelesen als auch gesetzt werden.
IS *operand*	**IS-Klausel:** Der *operand* in der IS-Klausel weist eine Objektdaten-Variable als Adresse zu, um den Property-Wert zu speichern. Die zugewiesene Objektdaten-Variable muss nicht unbedingt eine Gruppe sein. Die Variable wird in normaler Operanden-Syntax referenziert. Dies bedeutet, dass wenn die Objektdaten-Variable ein Array ist, sie mit Index-Notation referenziert werden muss. Nur die vollständige Indexbereichs-Notation und Stern-Notation ist zulässig. 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-Variable in einem PROPERTY-Statement außerhalb des INTERFACE-Statements zuweisen. Wenn die IS-Klausel weggelassen wird, wird die Property mit der Objektdaten-Variable 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.
END-PROPERTY	Das für Natural reservierte Wort END-PROPERTY muss zum Beenden des Interfaces PROPERTY-Definition benutzt werden.

Beispiele Property-Definition

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 Method für das Interface.

```
METHOD method-name
[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 Method in verschiedenen Klassen anders implementieren.

Syntax-Element	Beschreibung
method-name	**Method-Name:** Dies ist der der Method zuzuweisende Name. Der Method-Name kann maximal bis zu 32 Zeichen enthalten und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen entnehmen Sie dem Abschnitt *Namenskonventionen für Benutzervariablen*). Er muss pro Interface eindeutig sein. Wenn die Method 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.
`IS` *subprogram-name*	**Name des Subprogramms:** Dies ist der Name des die Method implementierenden Subprogramms. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist *method-name* (wenn die `IS`-Klausel nicht angegeben wird).
`PARAMETER`	**Parameter-Definition:** Mit dieser Klausel werden die Parameter der Method angegeben; sie hat dieselbe Syntax wie die `PARAMETER`-Klausel vom `DEFINE DATA`-Statement. 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. In der Parameter Data Area als `BY VALUE` markierte Parameter sind Eingabe-Parameter der Methode.

Syntax-Element	Beschreibung
	Nicht als BY VALUE markierte Parameter werden referenziert (By Reference) übergeben und sind Eingabe-/Ausgabe-Parameter. Dies ist die Voreinstellung. Der als BY VALUE RESULT markierte erste Parameter wird als Rückgabewert für die Method zurückgegeben. Wenn mehr als ein Parameter so markiert ist, werden die anderen als Eingabe/Ausgabe-Parameter behandelt. Als OPTIONAL markierte Parameter müssen nicht angegeben werden, wenn die Method aufgerufen wird. Auf ihre Angabe können Sie verzichten, indem Sie die **Notation** *n*X im SEND METHOD-Statement verwenden.
END-METHOD	Das für Natural reservierte Wort END-METHOD muss zum Beenden der METHOD-Definition für das Interface benutzt werden.

80 LIMIT

- Funktion LIMIT 634
- Syntax-Beschreibung LIMIT 635
- Beispiele LIMIT 635

```
LIMIT n
```

Dieser Abschnitt 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 LIMIT

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`LIMIT` *n*	**Limit-Angabe:** Das Limit *n* muss als numerische Konstante angegeben werden und kann einen Wert von `0` bis `4294967295` (führende Nullen sind optional) haben. Ist das Limit auf Null (0) gesetzt, wird die Schleife nicht durchlaufen.

Beispiele LIMIT

- 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         CITY            CNT
                        ID
-------------------- --------- -------------------- -----------

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
```

81 LOOP

- Funktion LOOP 638
- Einschränkung bei LOOP 638
- Syntax-Beschreibung LOOP 639
- Beispiele LOOP 639

```
[CLOSE] LOOP [(r)]
```

Dieser Abschnitt behandelt folgende Themen:

Funktion LOOP

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.

Das Statement LOOP wird bei den folgenden Statements verwendet: CALL FILE, CALL LOOP, FIND, FOR, HISTOGRAM, PARSE JSON, PARSE XML, READ, READLOB, READ RESULT SET (SQL), READ WORK FILE, REPEAT, SELECT (SQL), SORT, UPLOAD PC FILE.

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 bei LOOP

- 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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`LOOP (r)`	**Statement-Referenzierung:** Sollen mehrere Schleifen geschlossen werden, so kann ein bestimmtes Statement per Statement-Label oder Quellcode- 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.

Anmerkungen:

1. Im Reporting Mode werden durch ein `END`-Statement alle noch aktiven Verarbeitungsschleifen, die noch nicht explizit durch ein `LOOP`-Statement beendet wurden, automatisch geschlossen.
2. Man kann das `LOOP`-Statement weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

Beispiele LOOP

- Beispiel 1 - LOOP
- Beispiel 2 - LOOP

Beispiel 1 - LOOP

```
0010 FIND ...
0020   READ ...
0030     READ ...
0040 LOOP (0010)    /* closes all loops
```

Beispiel 2 - LOOP

```
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
```

82 MERGE (SQL)

- Funktion MERGE (SQL) 642
- Einschränkung bei MERGE (SQL) 642
- Syntax-Beschreibung MERGE (SQL) 642
- Beispiele MERGE (SQL) 648

```
MERGE INTO table-name [[AS] correlation-name]
[include-columns] USING source-table
ON search-condition
{WHEN matching-condition THEN modification-operation} ...
[ELSE IGNORE]
[NOT ATOMIC CONTINUE ON SQLEXCEPTION]
[QUERYNO integer]
```

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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *MERGE - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

Funktion MERGE (SQL)

Das MERGE-Statement aktualisiert eine Tabelle mit den angegebenen Eingabedaten. Diejenigen Zeilen in der Zieltabelle, die zu den Eingabedaten passen, werden gemäß den Angaben aktualisiert, und Zeilen, welche in der Zieltabelle nicht vorhanden sind, werden eingefügt.

Das MERGE-Statement gehört zum **SQL Extended Set**.

Einschränkung bei MERGE (SQL)

Dieses Statement steht nur bei Natural for Db2 zur Verfügung.

Syntax-Beschreibung MERGE (SQL)

Syntax-Element	Beschreibung
MERGE INTO	**MERGE INTO-Klausel:** MERGE INTO initiiert ein SQL MERGE-Statement, das eine Kombination aus einem SQL INSERT- und einem SQL Searched UPDATE-Statement ist.
table-name	**Tabellenname:** Identifiziert das Ziel der INSERT- oder UPDATE-Operation des MERGE-Statement.

Syntax-Element	Beschreibung
	Weitere Informationen siehe *table-name*.
[AS] *correlation-name*	**[AS] *correlation-name*-Klausel:** Gibt einen alternativen Namen für die Zieltabelle an. Der alternative Name kann als Qualifikationsmerkmal verwendet werden, wenn auf Spalten der Zwischenergebnistabelle verwiesen wird.
include-columns	**Include Columns-Klausel:** Gibt einen Satz von Spalten an, die zusammen mit den Spalten der Zieltabelle in die Ergebnistabelle des MERGE-Statements aufgenommen werden, wenn es in der FROM-Klausel eines SELECT-Statements geschachtelt ist. Die eingeschlossenen Spalten werden am Ende der durch die Zieltabelle identifizierten Spaltenliste angehängt. Weitere Informationen siehe ***include-columns***.
USING *source-table*	**USING *source-table*-Klausel:** Gibt die Werte für die Zeilendaten an, die in der Zieltabelle zusammengeführt werden sollen. Siehe *source-table*
ON *search-condition*	**ON *search-condition*-Klausel:** Gibt Join-Bedingungen zwischen der Quelltabelle (*source-table*) und der Zieltabelle an. Jeder Spaltenname in der Suchbedingung muss eine Spalte der Zieltabelle oder der Quelltabelle bezeichnen.
WHEN *matching-condition*	**WHEN *matching-condition*-Klausel:** Gibt die Bedingung an, für die die in der folgenden THEN-Klausel definierte Änderungsoperation durchgeführt werden soll. Siehe *matching-condition*.
THEN *modification-operation*	**THEN *modification-operation*-Klausel:** Gibt die Operation an, die bei den Übereinstimmungen mit der in der vorangehenden WHEN-Klausel definierten Bedingung durchgeführt werden soll. Siehe *modification-operation*.
ELSE IGNORE	**ELSE IGNORE-Klausel:** Legt fest, dass bei Quellspalten, die die in der WHEN-Klausel angegebene Bedingung nicht erfüllen, keine Aktion durchgeführt wird.
NOT ATOMIC CONTINUE ON SQLEXCEPTION	**NOT ATOMIC CONTINUE ON SQLEXCEPTION-Klausel:** Gibt an, ob die Zusammenführungsverarbeitung fortgesetzt wird, wenn bei der Verarbeitung einer Zeile eines Satzes von Quellzeilen ein Fehler auftritt.
QUERYNO *integer*	**QUERYNO *integer*-Klausel:** Gibt die Nummer für dieses SQL-Statement an, die in der EXPLAIN-Ausgabe und den Db2-Trace-Aufzeichnungen verwendet wird.

source-table

```
table-reference
(VALUES { values-single-row   }        )
        { values-multiple-row }
[AS] correlation-name (column-name,...)
```

Syntax-Element	Beschreibung
table-reference	Gibt die Quelltabelle an, die mit der Zieltabelle zusammengeführt werden soll.
VALUES	Mit dem Schlüsselwort VALUES wird die Angabe von Werten für die Zeilendaten eingeleitet, die in die Zieltabelle eingefügt werden sollen..
values-single-row	Gibt eine einzelne Zeile der Quelldaten an. Siehe *values-single-row*.
values-multiple-row	Gibt mehrere Zeilen der Quelldaten an. Siehe *values-multiple-row*.
[AS] *correlation-name*	Gibt einen Korrelationsnamen für die Quelltabelle an.
column-name	Gibt einen Spaltennamen an, um die Eingabedaten mit der UPDATE SET *assignment-clause* (Zuweisungsklausel) für eine UPDATE-Operation oder mit der VALUES-Klausel für eine INSERT-Operation zu verbinden.

matching-condition

```
[NOT] MATCHED [AND search-condition]
```

Syntax-Element	Beschreibung
[NOT] MATCHED	Gibt die *modification-operation* (Änderungsoperation) an, die durchgeführt werden soll, wenn eine ON *search-condition*-Suchbedingung als wahr oder nicht wahr ausgewertet wird (NOT kann optional angegeben werden).
AND *search-condition*	Gibt eine (optionale) zusätzliche Bedingung an, die als wahr ausgewertet werden muss, bevor die *modification-operation* durchgeführt wird.

modification-operation

```
update-operation)
DELETE
signal-operation
insert-operation
```

Syntax-Element	Beschreibung
update-operation	Gibt an, dass die übereinstimmende Zielzeile mit den in der Zuweisungsklausel (UPDATE SET *assignment-clause*) zugewiesenen Werten geändert wird. Eine UPDATE-Operation ist nur zulässig, wenn die Übereinstimmungsbedingung (*matching-condition*) als wahr ausgewertet wird.
DELETE	Gibt an, dass die übereinstimmende Zielzeile gelöscht wird. Eine DELETE-Operation ist nur zulässig, wenn die Übereinstimmungsbedingung (*matching-condition*) als wahr ausgewertet wird.
signal-operation	Gibt den SQL-Fehler an, der ausgelöst werden soll. Eine SIGNAL-Operation ist nur zulässig, wenn die Übereinstimmungsbedingung (*matching-condition*) als wahr ausgewertet wird.
insert-operation	Gibt die Zeilen an, die in die Zieltabelle eingefügt werden sollen. Eine INSERT-Operation ist nur zulässig, wenn die Übereinstimmungsbedingung (*matching-condition*) als nicht wahr ausgewertet wird.

signal-operation

```
SIGNAL SQLSTATE [VALUE] sqlstate[SET MESSAGE_TEXT = scalar-expression]
```

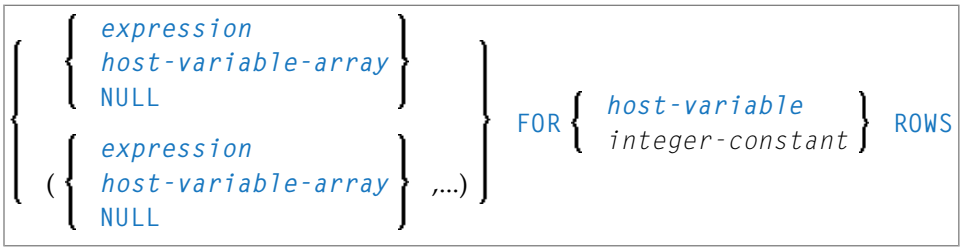
Syntax-Element	Beschreibung
SIGNAL	Gibt die SIGNAL-Operation an, die ausgeführt werden soll, wenn die Übereinstimmungsbedingung (*matching-condition*) als wahr ausgewertet wird. Db2 setzt einen SQLCODE -438, wenn ein Fehler durch das SIGNAL-Statement angezeigt wird.
SQLSTATE [VALUE] *sqlstate*	Gibt den SQLSTATE an, der von Db2 gesetzt werden soll. *sqlstate* ist eine 5 Zeichen lange alphanumerische Konstante oder eine alphanumerische Variable. *sqlstate*-Werte werden SQLSTATE von Db2 zugewiesen. Empfohlene Werte finden Sie in der entsprechenden Db2-Dokumentation.
SET MESSAGE_TEXT= *scalar-expression*	Diese optionale Klausel gibt eine Fehler- oder Warnmeldung an, die in das Feld SQLEERRMC der SQLCA gestellt wird oder die mit dem Statement GET DIAGNOSTICS abgerufen werden kann.

values-single-row



Syntax-Element	Beschreibung
expression	**Skalar-Ausdruck:** Gibt einen Skalar-Ausdruck an. Siehe ***Skalar-Ausdrücke***.
NULL	**NULL-Wert:** Gibt den NULL-Wert an.

values-multiple-row



Syntax-Element	Beschreibung
expression	**Skalar-Ausdruck:** Gibt einen Skalar-Ausdruck an. Siehe ***Skalar-Ausdrücke***.
host-variable-array	**Host-Variablen-Array:** Gibt ein Benutzervariablen-Array oder einen Indexbereich eines Array an. Wenn ein *host-variable-array* angegeben wird, muss die Compiler-Option DB2ARRY auf ON gesetzt werden. Mit dem Schlüsselwort INDICATOR kann ein optionales Indikator-Array angegeben werden: *host-variable-array* INDICATOR [:]*indicator-arry*.
NULL	**NULL-Wert:** Gibt den NULL-Wert an.
FOR ... ROWS	**Anzahl der zusammenzuführenden Zeilen:** Diese Klausel gibt die Anzahl der zusammenzuführenden Zeilen an. *host-variable* oder *integer-constant* wird ein Wert *k* zugewiesen. *k* muss

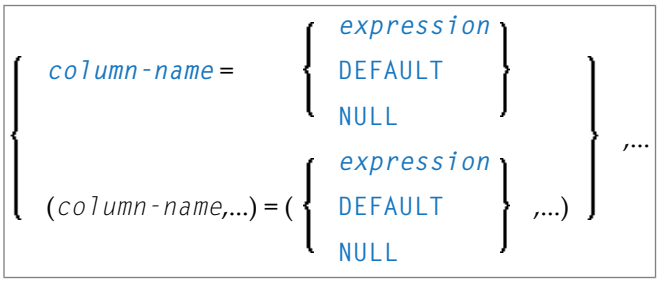
Syntax-Element	Beschreibung
	im Bereich von `0` bis `32767` liegen und muss kleiner als die oder gleich der Minimalgröße aller angegebenen Benutzervariablen-Arrays sein.

update-operation

```
UPDATE SET assignment-clause
```

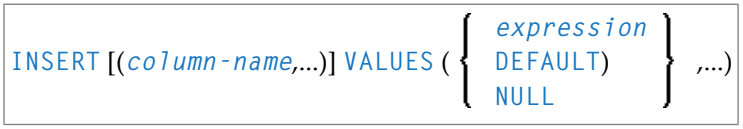
Syntax-Element	Beschreibung
`UPDATE SET`	Gibt an, welche Aktualisierungsoperation ausgeführt werden soll, wenn die Auswertung ergibt, dass die Suchbedingung (*`search-condition`*) wahr ist. Siehe *`assignment-clause`* weiter unten.

assignment-clause



Syntax-Element	Beschreibung
`column-name`	Gibt den Namen der Spalte an, für die ein Einfügewert vorgesehen ist.
`expression`	Gibt den neuen Wert für die Spalte an. Ein Ausdruck (*`expression`*) kann Referenzen auf Spalten der Quellentabelle (*`source-table`*) oder der Zieltabelle enthalten. *`expression`* kann keine Referenzen auf enthaltene Spalten enthalten.
`DEFAULT`	Gibt den Standardwert `DEFAULT` für die Spalte an. Der Wert, welcher zugewiesen wird, hängt davon ab, wie die Spalte definiert ist.
`NULL`	Gibt den Wert `NULL` als neuen Wert für die Spalte an.

insert-operation



Syntax-Element	Beschreibung
INSERT	Gibt die Einfügeoperation an, die ausgeführt werden soll, wenn die Suchbedingung (*search-condition*) nicht wahr ist.
column-name	Gibt die Spalte, für die ein Einfügewert vorgesehen ist.
VALUES	Mit diesem Schlüsselwort werden ein oder mehrere einzufügende Spaltenwerte eingeleitet.
expression	Gibt den neuen Wert für die Spalte an. Ein Ausdruck (expression) kann Referenzen auf Spalten der Quellentabelle enthalten. *expression* kann keine Referenzen auf Spalten der Zieltabelle enthalten.
DEFAULT	Gibt den Standardwert DEFAULT für die Spalte an. Der Wert, welcher zugewiesen wird, hängt davon ab, wie die Spalte definiert ist.
NULL	Gibt den Wert NULL als neuen Wert für die Spalte an.

Beispiele MERGE (SQL)

- Beispiel 1 - MERGE (SQL):
- Beispiel 2 - MERGE (SQL):
- Beispiel 3 - MERGE (SQL):

Beispiel 1 - MERGE (SQL):

Aktualisieren der Bestandsliste eines Kfz-Vertragshändlers. Hinzufügen eines neuen Fahrzeugmodells zur Bestandsliste bzw. Aktualisieren der Informationen über vorhandenes Fahrzeugmodell, welches schon in der Bestandsliste existiert.

```
DEFINE DATA LOCAL
01 #MODEL (A20)
01 #DELTA (I4)
END-DEFINE
* Setup input host variables
ASSIGN #MODEL = 'Grand Turbo'
ASSIGN #DELTA := 5
* Insert/Update into INVENTORY table
MERGE INTO CDS-INVENTORY T
  USING (VALUES (:#MODEL, :#DELTA)) AS S(MODEL, DELTA)
  ON T.MODEL = S.MODEL
```

```
WHEN MATCHED THEN UPDATE SET T.QUANTITY = T.QUANTITY + S.DELTA
WHEN NOT MATCHED THEN INSERT VALUES (S.MODEL, S.DELTA)
END TRANSACTION
END
```

Beispiel 2 - MERGE (SQL):

Aktualisieren der Bestandsliste eines Kfz-Vertragshändlers. Hinzufügen neuer Fahrzeugmodelle zur Bestandsliste und Aktualisieren der Informationen über vorhandenes Fahrzeugmodelle, welche schon in der Bestandsliste existieren. Die Eingabe kommt aus Natural-Arrays. Der spezifische Code der Arrays ist in Fettdruck dargestellt.

```
OPTIONS DB2ARRY ON
DEFINE DATA LOCAL
01 #MODEL_ARR (A20/1:20)
01 #DELTA_ARR (I4/1:20))
01 #ROW-COUNT (I4)
01 #NUM-ERRORS (I4)
01 #SQLCODE    (I4)
01 #SQLSTATE   (A5)
01 #ROW-NUM    (I4)
END-DEFINE
* Setup input host variables
ASSIGN #MODEL_ARR(1) = ‘Grand Turbo’
ASSIGN #DELTA_ARR(1) := 5
ASSIGN #MODEL_ARR(2) = ‘Blue Car’
ASSIGN #DELTA_ARR(2) := 3
. . .
* Insert/Update into INVENTORY table
CALLNAT 'NDBNOERR'
MERGE INTO CDS-INVENTORY T
  USING (VALUES (:#MODEL_ARR(*), :#DELTA_ARR(*))
  FOR 20 ROWS)
  AS S(MODEL, DELTA)
ON T.MODEL = S.MODEL
WHEN MATCHED THEN UPDATE SET T.QUANTITY = T.QUANTITY + S.DELTA
WHEN NOT MATCHED THEN INSERT VALUES (S.MODEL, S.DELTA)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
* Check outcome of MERGE
PROCESS SQL SYSIBM-SYSDUMMY1
  <<GET DIAGNOSTICS
  :#ROW-COUNT  = ROW_COUNT
  ,:#NUM-ERRORS = NUMBER>>
WRITE 'Number of rows merged             :' #ROW-COUNT /
      'NUMBER OF ERRORS                  :' #NUM-ERRORS
IF #NUM-ERRORS > 0
  FOR #I = 1 TO #NUM-ERRORS
    PROCESS SQL SYSIBM-SYSDUMMY1
      <<GET DIAGNOSTICS CONDITION   :#I
      :#SQLCODE   = DB2_RETURNED_SQLCODE,
      :#SQLSTATE  = RETURNED_SQLSTATE,
```

```
      :#ROW_NUM = DB2_ROW_NUMBER>>
    PRINT 'DB2_RETURNED_SQLCODE:' #SQLCODE
      'RETURNED_SQLSTATE:' #SQLSTATE
      'DB2_ROW_NUMBER:' #ROW_NUM (EM=99Z)
  END-FOR
END-IF
END TRANSACTION
END
```

Beispiel 3 - MERGE (SQL):

Zusammenführung von Verkaufsdaten aus der Tabelle `MSALES` in die Tabelle `MPRODUCT`. Demonstration der `MERGE`-Operation mit den Statements `DELETE`, `UPDATE`, `INSERT` und `SIGNAL`.

```
DEFINE DATA
LOCAL USING DEMSQLCA
LOCAL
1 V1 VIEW OF MPRODUCT
2 ID
2 NAME
2 INVENTORY
1  #M_TEXT  (A10) INIT <'Oversold: '>
   END-DEFINE
...
MERGE INTO MPRODUCT AS T
      USING (SELECT MSALES.ID        ,SUM(MSALES.SOLD) AS SOLD,
                     MAX(MCATALOG.NAME) AS NAME
               FROM MSALES, MCATALOG
               WHERE MSALES.ID = MCATALOG.ID
               GROUP BY MSALES.ID) AS S
               (ID,SOLD,NAME)
      ON S.ID = T.ID
      WHEN MATCHED AND T.INVENTORY = S.SOLD
                   THEN DELETE
      WHEN MATCHED AND T.INVENTORY < S.SOLD
                   THEN SIGNAL SQLSTATE '78000'
                        SET MESSAGE_TEXT =:#M_TEXT     || S.NAME
      WHEN MATCHED
                   THEN UPDATE SET T.INVENTORY = T.INVENTORY - S.SOLD
      WHEN NOT MATCHED
                   THEN INSERT VALUES(S.ID, S.NAME, -S.SOLD)
END TRANSACTION
END
```

83 METHOD

- Funktion METHOD 652
- Syntax-Beschreibung METHOD 652
- Beispiel für METHOD-Statement 653

```
METHOD method-name
OF [INTERFACE] interface-name
IS subprogram-name
END-METHOD
```

Dieser Abschnitt 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 METHOD

Das `METHOD`-Statement weist ein Subprogramm als Implementierung zu einer Method 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 METHOD

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
method-name	**Method-Name:** Dies ist der der **Method** zugewiesene Name.
`OF` *interface-name*	**Interface-Name:** Dies ist der dem **Interface** zugewiesene Name.
`IS` *subprogram-name*	**Name des Subprogramms:** Dies ist der Name des Subprogramms, das die Method implementiert. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist *method-name* (wenn die `IS`-Klausel nicht angegeben wird).

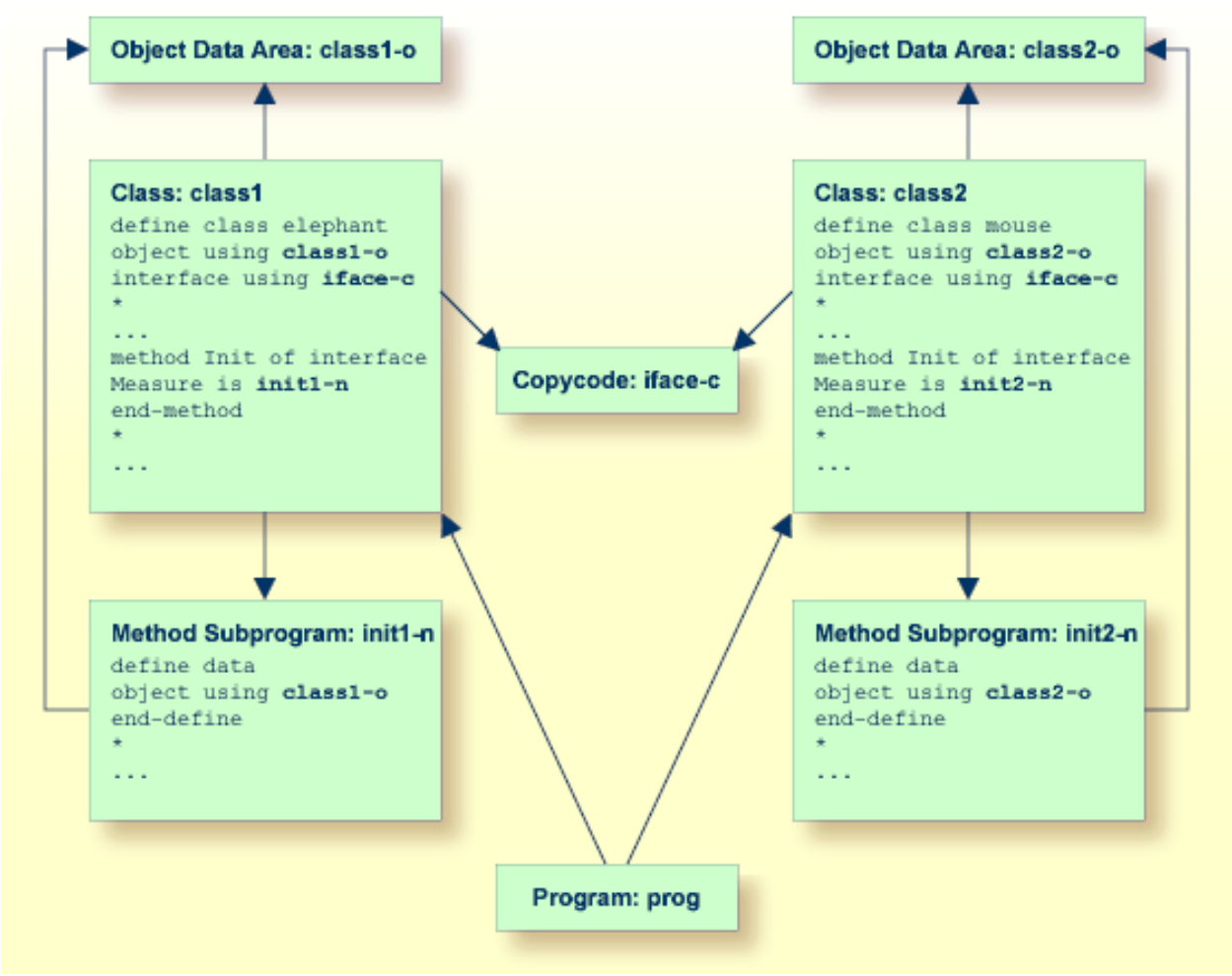
Syntax-Element	Beschreibung
END-METHOD	Das für Natural reservierte Wort END-METHOD muss zum Beenden des METHOD-Statements benutzt werden.

Beispiel für METHOD-Statement

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 Method 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 Method zuzuweisen.

Jetzt kann das Programm prog Objekte beider Klassen erstellen und sie mittels derselben Method 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
```

```
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
```

84 MOVE

- Funktion MOVE 658
- Syntax 1 - MOVE 658
- Syntax 2 - MOVE SUBSTRING 660
- Syntax 3 - MOVE BY NAME / POSITION 662
- Syntax 4 - MOVE EDITED (Editiermaske bei operand2) 663
- Syntax 5 - MOVE EDITED (Editiermaske bei operand1) 665
- Syntax 6 - MOVE LEFT / RIGHT JUSTIFIED 666
- Syntax 7 - MOVE NORMALIZED 667
- Syntax 8 - MOVE ENCODED 669
- Syntax 9 - MOVE ALL 672
- Beispiele MOVE 675

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: ADD | COMPRESS | COMPUTE | DIVIDE | EXAMINE | MULTIPLY | RESET | SEPARATE | SUBTRACT

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion MOVE

Das Statement MOVE dient dazu, den Wert eines Operanden in einen oder mehrere andere Operanden (Feld oder Array) zu übertragen.

Eine Natural-Systemfunktion darf nur verwendet werden, wenn das MOVE-Statement in Verbindung mit einem der folgenden Statements verwendet wird: AT BREAK-, AT END OF DATA oder AT END OF PAGE.

Siehe auch Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Syntax 1 - MOVE

```
MOVE [ROUNDED] operand1 [(parameter)] TO operand2 ...
```

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

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
MOVE ROUNDED	**MOVE ROUNDED-Option:** Diese Option bewirkt, dass *operand2* gerundet wird. ROUNDED wird ignoriert, wenn *operand2* nicht numerisch ist oder wenn der Quelloperand dieselben oder weniger Präzisionsstellen als der Zieloperand hat.

<table>
<tr><th>Syntax-Element</th><th>Beschreibung</th></tr>
<tr><td></td><td>Siehe auch Beispiel 1 - Verschiedene Beispiele für die Benutzung des MOVE-Statements.</td></tr>
<tr><td>operand1
operand2</td><td>Quell- und Zieloperanden:

operand1 ist der Quelloperand, dessen Wert in den Zieloperanden operand2 übertragen wird.

Weitere Informationen zur Datenübertragungs-Kompatibilität und den Regeln für die Datenübertragung finden Sie im Abschnitt Datenübertragung im Leitfaden zur Programmierung.

Ist operand2 eine dynamische Variable, kann ihre Länge mit der MOVE-Operation geändert werden. Die aktuelle Länge einer dynamischen Variable kann mittels der Systemvariablen *LENGTH bestimmt werden. Allgemeine Informationen zu dynamischen Variablen siehe den Abschnitt Dynamische und große Variablen benutzen im Leitfaden zur Programmierung.

Ein MOVE-Statement mit mehreren Zieloperanden ist mit den betreffenden einzelnen MOVE-Statements identisch:

<pre>MOVE #SOURCE TO #TARGET1 #TARGET2</pre>ist identisch mit:

<pre>MOVE #SOURCE TO #TARGET1
MOVE #SOURCE TO #TARGET2</pre></td></tr>
<tr><td>parameter</td><td>Parameter-Option:

Als parameter können Sie den Session-Parameter PM oder den Session-Parameter DF angeben:

<table>
<tr><td>PM=I</td><td>Schreibrichtung-Option:

Zur Unterstützung von Sprachen, deren Schreibrichtung von rechts nach links verläuft, können Sie die Option PM=I angeben, um den Wert von operand1 invers (d.h. von rechts nach links) in operand2 zu übertragen.

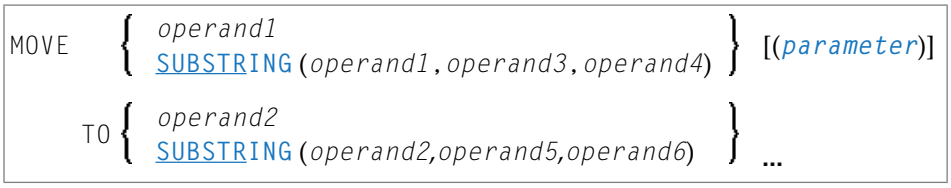
Zum Beispiel hätte als Ergebnis der folgenden Statements das Feld #B den Inhalt ZYX:

<pre>MOVE 'XYZ' TO #A
MOVE #A (PM=I) TO #B</pre>PM=I kann nur angegeben werden, wenn operand2 alphanumerisches Format hat.

Nachfolgende Leerzeichen in operand1 werden entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), dann wird der Wert umgedreht und anschließend in operand2 übertragen. Falls operand1 nicht alphanumerisches Format hat, wird der Wert in alphanumerisches Format umgesetzt, bevor er umgedreht wird.</td></tr>
</table>
</td></tr>
</table>

Syntax-Element	Beschreibung	
		Zur Verwendung von PM=I zusammen mit MOVE LEFT/RIGHT JUSTIFIED siehe MOVE LEFT/RIGHT JUSTIFIED.
	DF=S|I|L	**Datumsformat:** Wenn *operand1* eine Datumsvariable und *operand2* ein alphanumerisches Feld ist, können Sie den Session-Parameter DF als *parameter* für diese Datumsvariable angeben.

Syntax 2 - MOVE SUBSTRING



Diese Syntax gilt, wenn Sie nur einen Teil des Feldinhalts (einen Teil einer Zeichenkette) eines Quell- und/oder Zieloperanden übertragen wollen. Andernfalls gilt ***Syntax 1***.

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

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A			A	U				B						ja	nein
operand2		S	A			A	U				B						ja	nein
operand3	C	S						N	P	I	B*						ja	nein
operand4	C	S						N	P	I	B*						ja	nein
operand5	C	S						N	P	I	B*						ja	nein
operand6	C	S						N	P	I	B*						ja	nein

* Siehe Text.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

MOVE SUBSTRING	**MOVE SUBSTRING:**
	
	Ohne SUBSTRING-Option wird der ganze Inhalt des Feldes übertragen.
	
	Die SUBSTRING-Option ermöglicht es Ihnen, nur einen bestimmten Teil eines alphanumerischen oder eines Unicode-Feldes zu übertragen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand3*) und dann die Länge (*operand4*) des Feldteils an, der übertragen werden soll.
	
	■ Wenn das zugrundeliegende Feldformat von *operand1* alphanumerisch/Unicode (A/U) oder binär (B) ist, gelten die mit *operand3* oder *operand4* angegebenen Werte als Byte-Zahlen;
	■ Wenn das zugrundeliegende Feldformat von *operand1* Unicode (U) ist, gelten die mit *operand3* oder *operand4* als Zahlen in Unicode-Codeeinheiten, d.h. als Doppelbytes.
	
	Um zum Beispiel die 5. bis einschließlich 12. Stelle eines Feldes #A in ein Feld #B zu übertragen, geben Sie Folgendes an:
	
	`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/Unicode oder binären 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/Unicode (A/U) oder binär (B) ist, gelten die mit *operand5* oder *operand6* angegebenen Werte als Byte-Zahlen.
	■ Wenn das zugrundeliegende Feldformat von *operand2* Unicode (U) ist, gelten die mit *operand3* oder *operand4* als Zahlen in Unicode-Codeeinheiten, d.h. als Doppelbytes.
	
	Um zum Beispiel den Wert eines Feldes #A in die 3. bis einschließlich 6. Stelle eines Feld #B zu übertragen, geben Sie Folgendes an:
	
	`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 größere erste Stelle würde einen Laufzeit-Fehler zur Folge haben, weil dies zu einer nicht definierten Lücke im Inhalt von *operand2* führen würde.
	
	Wenn *operand3*/*operand5* oder *operand4*/*operand6* eine binäre Variable ist, kann sie nur mit einer Länge kleiner als oder gleich 4 benutzt werden.

Syntax-Element	Beschreibung
	Wenn Sie *operand3*/*operand5* weglassen, wird ab der ersten Stelle des Feldes übertragen. Wenn Sie *operand4*/*operand6* weglassen, wird ab der angegebenen Stelle 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 erhöhen, muss *operand6* angegeben werden, um die neue Länge der Variablen zu bestimmen. **Anmerkung:** MOVE mit SUBSTRING-Option bewirkt eine Byte-für-Byte-Übertragung (d.h. die unter *Arithmetische Operationen* im *Leitfaden zur Programmierung* beschriebenen Regeln gelten hier **nicht**).
parameter	**Parameter-Option:** Siehe *parameter* unter *Syntax 1*.

Syntax 3 - MOVE BY NAME / POSITION

```
MOVE  BY { [NAME]   } operand1 TO operand2
         { POSITION }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1				G														ja	nein
operand2				G														ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
MOVE BY NAME *operand1* TO *operand2*	**MOVE BY NAME-Option:** 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. 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 Operanden können auch Views sein.

Syntax-Element	Beschreibung
	Anmerkung: Die Reihenfolge der einzelnen Übertragungen ergibt sich aus der Reihenfolge der Felder in *operand1* Siehe auch ***Beispiel 2 - Statement MOVE BY NAME***. **MOVE BY NAME mit Arrays:** Enthalten die Datenstrukturen Arrays, so werden diese bei der Übertragung intern mit dem Index (*) versehen; dies kann zu einem Fehler führen, falls die Arrays nicht den Zuweisungsbedingungen für Arrays entsprechen (siehe Abschnitt *Verarbeitung von Arrays* im *Leitfaden zur Programmierung*). Siehe auch ***Beispiel 3 - MOVE BY NAME mit Arrays***.
`MOVE BY POSITION` *operand1* `TO` *operand2*	**MOVE BY POSITION-Option:** Mit dieser Option können Sie Werte von Feldern einer Gruppe in Felder einer anderen Gruppe übertragen, und zwar unabhängig von den Namen der Felder. 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). 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 Regeln für arithmetische Operationen (siehe *Arithmetische Operationen* im *Leitfaden zur Programmierung*). Die beiden Operanden können auch Views sein. Siehe auch ***Beispiel 4 - MOVE BY POSITION***.

Syntax 4 - MOVE EDITED (Editiermaske bei operand2)

```
MOVE EDITED operand1 TO operand2  { (EM=value)  }
                                  { (EMU=value) }
```

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

MOVE EDITED	**MOVE EDITED-Option - Übertragung mit Editiermaske:**
	
	Ist für *operand2* eine Editiermaske definiert, so wird der Wert von *operand1* unter Verwendung dieser Editiermaske in das Feld *operand2* gestellt.
	
	Die Editiermaske kann als eine Eingabe-Editiermaske für *operand2* angesehen werden, die dazu dient anzugeben, an welchen Stellen in dem alphanumerischen/Unicode-Inhalt von *operand1* die signifikanten Eingabedaten für *operand2* zu finden sind.
	
	Wenn die Editiermaske mehr Zeichen oder Ziffern enthält, als in *operand2* vorhanden sind, erfolgt eine entsprechende Abschneidung. Die Länge von *operand1* darf nicht kleiner sein als die von der Editiermaske dargestellte Länge des Eingabewertes. Wenn die Länge von *operand1* die Länge der Editiermaske übersteigt, werden alle darüber hinaus gehenden Daten ignoriert.
	
	Unter der Voraussetzung, dass die Länge von *operand1* nicht die Länge der Editiermaske übersteigt, kann man die Operation
	
	`MOVE EDITED operand1 TO operand2 (EM=value)`
	
	als Ausführung der folgenden Operation ansehen:
	
	`STACK TOP DATA operand1`
	`INPUT operand2 (EM=value)`
	
	Siehe auch ***Beispiel 1 - Verschiedene Beispiele für die Benutzung des MOVE-Statements***.
	
	Wenn der Inhalt von *operand1* neben dem relevanten Eingabewert für *operand2* Extra-Zeichen enthält, müssen diese zusätzlichen Zeichen in der Editiermaske angegeben werden, damit sie ignoriert werden.
	
	Siehe ***Beispiel 6 – MOVE EDITED mit Editiermaske und Extra-Zeichen*** .
	
	Alternativ kann ein Fragezeichen (?) als Platzhalterzeichen verwendet werden, um diese zusätzlichen Zeichen zu ignorieren.
	
	Siehe ***Beispiel 7 – MOVE EDITED mit Editiermaske und Platzhalterzeichen '?'***.

EM	**Editiermaske:** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der *Parameter-Referenz*.
EMU	**Unicode-Editiermaske:** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EMU in der *Parameter-Referenz*.

Syntax 5 - MOVE EDITED (Editiermaske bei operand1)

```
MOVE EDITED operand1 { (EM=value) } TO operand2
                     { (EMU=value)}
```

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
MOVE EDITED	**MOVE EDITED-Option - Übertragung mit Editiermaske:** Ist für *operand1* eine Editiermaske definiert, wird diese Editiermaske auf *operand1* angewandt und der Wert anschließend in *operand2* übertragen. Die Editiermaske kann als eine *Ausgabe*-Editiermaske für *operand1* angesehen werden, die dazu dient, eine alphanumerische/Unicode-Zeichenkette mit dem/der durch die Editiermaske festgelegten Layout/Länge zu erstellen. Außer den aus *operand1* stammenden Datenzeichen oder -ziffern können Sie zusätzliche ausschmückende Zeichen in die Ausgabe-Zeichenkette aufnehmen. Wenn die Editiermaske mehr Zeichen oder Ziffern referenziert, als in *operand1* vorhanden sind, erfolgt eine entsprechende Abschneidung. Die Länge der erstellten Ausgabe-Zeichenkette (die sich nach Anwendung der Editiermaske aus dem Wert von *operand1* ergibt) darf nicht die Länge von *operand2* übersteigen. Unter der Voraussetzung, dass die Länge von *operand2* nicht kleiner ist als die Länge der Editiermaske, kann man die Operation `MOVE EDITED operand1 (EM=value) TO operand2` als eine Operation

Syntax-Element	Beschreibung
	`WRITE operand1 (EM=value)` betrachten, bei der die Ausgabe nicht auf den Schirm erfolgt, sondern in die Variable *operand2* geschrieben wird. Siehe auch ***Beispiel 1 - Verschiedene Beispiele für die Benutzung des MOVE-Statements***.
EM	**Editiermaske:** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der *Parameter-Referenz*.
EMU	**Unicode-Editiermaske:** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EMU in der *Parameter-Referenz*.

Syntax 6 - MOVE LEFT / RIGHT JUSTIFIED

```
MOVE { LEFT  } [JUSTIFIED] operand1 [(parameter)] TO operand2
     { RIGHT }
```

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
MOVE LEFT / RIGHT JUSTIFIED	**Ausrichtung:** Mit dieser Option können Sie festlegen, ob der übertragene Wert in *operand2* links- oder rechtsbündig ausgerichtet werden soll. MOVE LEFT/RIGHT JUSTIFIED kann nicht benutzt werden, wenn *operand2* eine dynamische Variable ist.
MOVE LEFT JUSTIFIED	**Linksbündig:** Bei MOVE LEFT JUSTIFIED werden vorangestellte Leerzeichen in *operand1* entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), bevor der Wert linksbündig in *operand2* gestellt wird. Der Rest von *operand2* wird dann mit

Syntax-Element	Beschreibung
	Leerzeichen aufgefüllt. Falls der Wert länger als *operand2* ist, wird der Wert rechts abgeschnitten.
MOVE RIGHT JUSTIFIED	**Rechtsbündig:** Bei MOVE RIGHT JUSTIFIED werden nachfolgende Leerzeichen in *operand1* abgeschnitten (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt), bevor der Wert rechtsbündig in *operand2* gestellt wird. Der Rest von *operand2* wird dann mit Leerzeichen aufgefüllt. Falls der Wert länger als *operand2* ist, wird der Wert links abgeschnitten. Siehe auch ***Beispiel 1 - Verschiedene Beispiele für die Benutzung des MOVE-Statements***.
parameter	**Parameter für invertierte Anzeigerichtung:** Wenn Sie MOVE LEFT/RIGHT JUSTIFIED in Verbindung mit PM=I verwenden, erfolgt die Übertragung in folgenden Schritten: 1. Falls *operand1* nicht alphanumerisches/Unicode-Format hat, wird der Wert in alphanumerisches/Unicode-Format umgesetzt. 2. Nachfolgende Leerzeichen in *operand1* werden entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt). 3. Bei LEFT JUSTIFIED werden außerdem vorangestellte Leerzeichen in *operand1* entfernt (auf Großrechnern werden Leerzeichen und binäre Nullen entfernt). 4. Der Wert wird umgedreht und dann in *operand2* übertragen. 5. Gegebenenfalls wird *operand2* 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 siehe *Unicode- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*, Unterabschnitt MOVE NORMALIZED in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Syntax-Diagramm:

```
MOVE NORMALIZED operand1 TO operand2
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A				U											ja	nein
operand2		S	A			A	U											ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
MOVE NORMALIZED	**MOVE NORMALIZED-Option:** Diese Option dient zum Konvertieren von Unicode-Feldern mit potenziell nicht normalisiertem Inhalt in die „Unicode Normalization Form C“ (NFC). Diese zusammengesetzte Form einer Unicode-Zeichenkette enthält keine Kombinationssequenzen für Zeichen, die als vordefinierte Zeichen zur Verfügung stehen. Siehe auch *http://www.unicode.org/reports/tr15/#Canonical_Composition_Examples* (*Normalization Forms D and C Examples*). Beispiel: `MOVE NORMALIZED #SCR TO #TGT`
operand1	**Quelloperand:** Der Quelloperand *operand1* enthält die umzuwandelnde Unicode-Zeichenkette.
operand2	**Zieloperand:** Der Zieloperand *operand2* dient zur Aufnahme der umgewandelten Unicode-Zeichenkette.

Beispiel:

Einige Codepunkte haben unterschiedliche Darstellungen in Unicode. Zum Beispiel der deutsche Buchstabe Ä: die getrennte Darstellung im Unicode ist U+0041, gefolgt von U+0308, bei der ein zusammengesetztes Zeichen (U+0308) benutzt wird; eine andere Darstellung ist das vorher zusammengesetzte Zeichen U+00C4. Das Statement MOVE NORMALIZED wandelt die Unicode-Darstellung mit zusammengesetzten Zeichen in eine standardisierte Unicode-Darstellung mittels vorher zusammengesetzter Zeichen, wo immer dies möglich ist.

Syntax 8 - MOVE ENCODED

Dieser Abschnitt erläutert die Syntax des Statements `MOVE ENCODED`.

Weitere Informationen siehe *Unicode- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*, Unterabschnitt `MOVE ENCODED` in der *Unicode- und Codepage-Unterstützung*-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
operand1	C	S	A			A	U	B									ja	nein
operand2		S				A	U										ja	nein
operand3		S				A	U	B									ja	ja
operand4		S	A			A	U										ja	nein
operand5		S							I4								ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`MOVE ENCODED`	**Codepage-Umsetzung:** Die Option `MOVE ENCODED` setzt eine in einer Codepage kodierte Zeichenkette in die äquivalente Zeichenkette einer anderen Codepage um. **Anmerkung:** Natural verwendet für die Unicode-Umwandlung die „International Components for Unicode (ICU) Library“. Weitere Informationen siehe *http://icu.sourceforge.net/userguide/*.
operand1	**Umzusetzende Zeichenkette:** Der Quelloperand *operand1* enthält die umzusetzende Zeichenkette.
`CODEPAGE` *operand2*	**Codepage der umzusetzenden Zeichenkette:** In *operand2* geben Sie die Codepage von *operand1* an.

Syntax-Element	Beschreibung
	Kann nur angegeben werden, wenn *operand1* das Format A oder B hat. Siehe Anmerkung 1 und 3.
TO *operand3*	**Umgesetzte Zeichenkette:** In den Zieloperanden *operand3* wird die konvertierte Zeichenkette gestellt. Wenn das Ergebnis der Umsetzung nicht in den Zieloperanden (*operand3*) passt, wird das Ergebnis aufgefüllt bzw. abgeschnitten. Als Füllzeichen wird das Leerzeichen der resultierenden Codepage benutzt. Wenn das Zielfeld als eine dynamische Variable definiert wird, ist kein Auffüllen oder Abschneiden erforderlich, da sich die Länge der dynamischen Variablen automatisch an die Länge des Ergebnisses der Konvertierung anpasst.
CODEPAGE *operand4*	**Codepage der umgesetzten Zeichenkette:** In *operand4* geben Sie die Codepage von *operand3* an. Die Codepage von *operand3* kann nur angegeben werden, wenn *operand3* das Format A oder B hat. Siehe Anmerkung 1 und 3.
GIVING *operand5*	**GIVING-Klausel:** Ohne das Schlüsselwort GIVING wird im Falle eines Fehlers eine Natural-Fehlermeldung zurückgegeben. Wenn Sie das Schlüsselwort GIVING angeben, gibt *operand5* eine 0 oder den Natural-Fehlercode anstelle der Natural-Fehlermeldung zurück. Wenn der Zieloperand abgeschnitten wird, wird keine Natural-Fehlermeldung ausgegeben, aber wenn das Schlüsselwort GIVING benutzt wird, enthält *operand5* einen entsprechenden Fehlercode, um auf ein Abschneiden hinzuweisen.

Anmerkungen:

1. Wenn kein Codepage-Operand angegeben wird, dann wird die voreingestellte (Standard-)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.
3. Nur mit dem Makro NTCPAGE im Quellmodul NATCONFG definierte Codepage-Namen können benutzt werden. Andere Codepage-Namen werden mit einem entsprechenden Laufzeitfehler zurückgewiesen.

Beispiele für MOVE ENCODED:

```
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
```

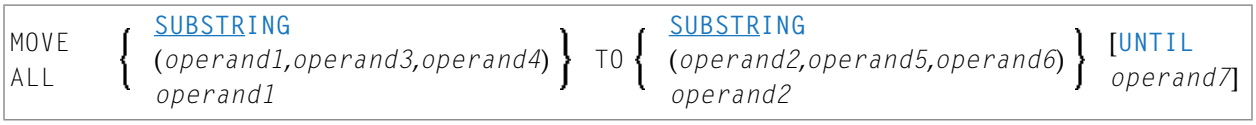
Gültig: Die Konvertierung erfolgt mittels der betreffenden Codepages von `A100-FIELD` (Format/Länge: `A100`) in `A50-FIELD` (Format/Länge: `A50`). Das Ziel wird abgeschnitten. Da eine `GIVING`-Klausel angegeben wird, erhält das `RC-FIELD` einen Fehlercode, der darauf hinweist, dass in diesem Fall Werte abgeschnitten wurden.

Syntax 9 - MOVE ALL

Das Statement `MOVE ALL` ermöglicht es, den Wert von *operand1* so oft nach *operand2* zu übertragen, bis das Zielfeld voll oder bis der `UNTIL`-Wert (*operand7*) erreicht ist.

Mit einer `SUBSTRING`**-Klausel** können Sie die `MOVE ALL`-Operation auf Segmente des Ausgangs- und des Zielfelds begrenzen.

Syntax-Diagram:



Anmerkung: Die `UNTIL`-Option ist nicht erlaubt, wenn eine `SUBSTRING`-Klausel für den Zieloperanden verwendet wird.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A			A	U	N[1]			B						yes	no
operand2		S	A			A	U				B						yes	yes
operand3	C	S						N	P	I	B[2]						yes	no
operand4	C	S						N	P	I	B[2]						yes	no
operand5	C	S						N	P	I	B[2]						yes	no
operand6	C	S						N	P	I	B[2]						yes	no
operand7	C	S						N	P	I							yes	no

[1] Ein numerisches Format (N) für A *operand1* ist nur dann erlaubt, wenn die `SUBSTRING`-Klausel nicht benutzt wird.

[2] Wenn *operand3*/*operand5* oder *operand4*/*operand6* eine binäre Variable ist, kann sie nur mit einer Länge von kleiner als oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Ausgangsoperand:** Dieser Operand enthält den zu übertragenden Wert. Bei einem numerischen Operanden werden alle Stellen einschließlich vorangestellter Nullen übertragen.
TO *operand2*	**Zieloperand:** In diesen Operanden wird der Wert übertragen. Der vorherige Inhalt des Operanden wird vor der MOVE ALL-Operation nicht zurückgesetzt. Dies ist insbesondere bei Verwendung der UNTIL-Option zu beachten, da bereits in *operand2* enthaltene Daten dort behalten werden, es sei denn, sie werden während der MOVE ALL-Operation ausdrücklich überschrieben.
UNTIL *operand3*	**UNTIL-Option:** Mit der UNTIL-Option können Sie die MOVE ALL-Operation auf eine bestimmte Anzahl von Stellen in *operand2* begrenzen. Mit *operand3* geben Sie an, wieviele Stellen in *operand2* gefüllt werden sollen; ist diese Anzahl erreicht, wird die MOVE ALL-Operation beendet. Übersteigt die Länge von *operand7* die Länge von *operand2*, wird die MOVE ALL-Operation beendet, sobald *operand2* vollständig gefüllt ist. Die Option UNTIL kann auch verwendet werden, um einer dynamischen Variable einen Startwert zuzuweisen. Wenn *operand2* eine dynamische Variable ist, entspricht ihre Länge nach der MOVE ALL-Operation dem Wert von *operand7*. Die aktuelle Länge einer dynamischen Variable kann unter Verwendung der Systemvariablen *LENGTH festgestellt werden. Allgemeine Informationen zu dynamischen Variablen finden Sie im Abschnitt *Benutzung dynamischer Variablen*.. **Anmerkung:** Die UNTIL-Option ist nicht erlaubt, wenn für den Zieloperanden eine SUBSTRING-Klausel verwendet wird.
SUBSTRING	**SUBSTR-Klausel:** Mit der SUBSTR-Klausel können Sie ein festes Segment der Ausgangs- oder Zielvariablen in einem MOVE ALL-Statement auswählen, während ohne die SUBSTRING-Klausel der gesamte Inhalt der Ausgangs- oder Zielvariablen verarbeitet wird. *operand3* und *operand4* beschreiben die Startposition und die Länge des als Ausgangswert verwendeten Segments von *operand1*. *operand5* und *operand6* beschreiben die Startposition und die Länge des Segments von *operand2*, das bei der MOVE ALL-Operation gefüllt wird. Wenn die Startposition (*operand3* bzw. *operand5*) weggelassen wird, dann wird standardmäßig Position 1 als Startposition genommen. Wird die Substring-Länge (*operand4* bzw. *operand6*) weggelassen, dann wird die übrig bleibende Länge des Feldes genommen.

Syntax-Element	Beschreibung
	Wenn die SUBSTRING-Klausel für das Ausgangsfeld verwendet wird, müssen der Startwert und die Länge (*operand3* und *operand4*) ein Datensegment beschreiben, das vollständig innerhalb von *operand5* liegt. Wenn die SUBSTRING-Klausel für das Zielfeld verwendet wird, gelten folgende Regeln: ■ Wenn *operand2* eine Variable mit fester Länge ist, muss der durch den Startwert und die Länge beschriebene Bereich (*operand5* und *operand6*) vollständig im Bereich des Feldes liegen. ■ Wenn *operand2* eine dynamische Variable ist, kann der Startwert (*operand5*) entweder in oder unmittelbar hinter die aktuelle Feldlänge (*LENGTH +1) zeigen. Wenn das Ende des SUBSTR-Bereichs innerhalb der zugeordneten Felddaten liegt, erfolgt die Verarbeitung wie bei einem Feld mit fester Länge. Wenn das SUBSTRING-Ende die Größe des aktuellen Felds überschreitet, wird die dynamische Variable im gleichen Maß erweitert. Siehe auch *Beispiele für die Verwendung der SUBSTRING-Klausel* weiter unten.

Beispiele für die Verwendung der SUBSTRING-Klausel

```
DEFINE DATA LOCAL
1 ALFA (A10) INIT <'AAAAAAAAAA'>
1 DYN (A) DYNAMIC INIT <'1234567890'>
1 #VAL (A4) INIT <'1234'>
END-DEFINE
```

Statement	Ergebnis	
	Vorher	**Nachher**
MOVE ALL SUBSTR(#VAL,1,2) TO ALFA	AAAAAAAAAA	1212121212
MOVE ALL '123' TO SUBSTR (ALFA,3,5)	AAAAAAAAAA	AA12312AAA
MOVE ALL 'x' TO SUBSTR (DYN,7,3)	1234567890 (*LENGTH=10)	123456xxx0 (*LENGTH=10)
MOVE ALL 'xyz' TO SUBSTR (DYN,7,6)	1234567890 (*LENGTH=10)	123456xyzxyz (*LENGTH=12)
MOVE ALL 'xyz' TO SUBSTR (DYN,11,4)	1234567890 (*LENGTH=10)	1234567890xyzx (*LENGTH=14)

Beispiele MOVE

- 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 5 - MOVE ALL
- Beispiel 6 – MOVE EDITED mit Editiermaske und Extra-Zeichen
- Beispiel 7 – MOVE EDITED mit Editiermaske und Platzhalterzeichen '?'

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 #FIELDA (A10) INIT <'AAAAAAAAAA'>
  2 #FIELDB (A10) INIT <'BBBBBBBBBB'>
  2 #FIELDC (A10) INIT <'CCCCCCCCCC'>
  2 #FIELDD (A10) INIT <'DDDDDDDDDD'>
1 #TBLOCK
  2 #FIELD1 (A15) INIT <' '>
  2 #FIELDA (A10) INIT <' '>
  2 #FIELD2 (A10) INIT <' '>
  2 #FIELDB (A10) INIT <' '>
  2 #FIELD3 (A20) INIT <' '>
  2 #FIELDC (A10) INIT <' '>
END-DEFINE
*
MOVE BY NAME #SBLOCK TO #TBLOCK
*
WRITE NOTITLE 'CONTENTS OF #TBLOCK AFTER MOVE BY NAME:'
       // '=' #TBLOCK.#FIELD1
        / '=' #TBLOCK.#FIELDA
        / '=' #TBLOCK.#FIELD2
        / '=' #TBLOCK.#FIELDB
        / '=' #TBLOCK.#FIELD3
        / '=' #TBLOCK.#FIELDC
*
END
```

Inhalt von `#TBLOCK` nach der `MOVE BY NAME`-Verarbeitung:

```
CONTENTS OF #TBLOCK AFTER MOVE BY NAME:

#FIELD1:
#FIELDA: AAAAAAAAAA
#FIELD2:
#FIELDB: BBBBBBBBBB
#FIELD3:
#FIELDC: 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 #FIELDA (N1) INIT <1,2,3,4,5>
    2 REDEFINE #FIELDA
      3 #FIELDB (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 #FIELDA (2:4) TO #FIELDA (1:3)
MOVE #FIELDB (2:4) TO #FIELDB (1:3)
```

Zunächst wird der Inhalt der Ausprägungen 2 bis 4 von #FIELDA in die Ausprägungen 1 bis 3 von #FIELDA 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 #FIELDB in die Ausprägungen 1 bis 3 von #FIELDB 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.

Beispiel 5 - MOVE ALL

```
** 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.)
            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
```

Beispiel 6 – MOVE EDITED mit Editiermaske und Extra-Zeichen

```
** Example 'MOVEX3': MOVE EDITED with EM and extra characters
************************************************************************
DEFINE DATA LOCAL
1 #NUM  (N3.2)
1 #A5   (A5)
1 #A8   (A8)
1 #A10  (A10)
1 #A12  (A12)
END-DEFINE
*
*  MOVE EDITED operand1 (EM=value) TO operand2
*
*  The Edit mask will be applied for operand1 and the result is moved
*  to operand2. The created string is described by the edit mask
*  where also extra characters can be included.
*
*  This is similar to the statement 'WRITE operand (EM=value)'.
*
*
#NUM := 23.45
MOVE EDITED #NUM (EM=Z99.99+) TO #A8
WRITE 'MOVE 23.45 TO #NUM'
WRITE 'MOVE EDITED #NUM (EM=Z99.99+) TO #A8'      15X '=' #A8  /
*
*  Show leading zero with filler 'x' defined as first character in EM.
*
#NUM := -78.15
MOVE EDITED #NUM (EM=xFULL-Z99.99) TO #A12
WRITE 'MOVE -78.15 TO #NUM'
WRITE 'MOVE EDITED #NUM (EM=xFULL-Z99.99) TO #A12' 9X '=' #A12 //
*
*  Show leading blank with filler '.' defined as first character in EM.
*  If extra character is equal to edit mask characters, then enclose
*  it within apostrophes.
*
MOVE '  AB ' TO #A5
MOVE EDITED #A5 (EM=.GOXXXXXE'X'TRA) TO #A12
WRITE 'MOVE ''  AB '' TO #A5'
WRITE 'MOVE EDITED #A5 (EM=.GOXXXXXE''X''TRA) TO #A12' 7X '=' #A12 /
*
*  Similar handling for WRITE statement with edit mask.
*
WRITE 'WRITE #A5 (EM=.GOXXXXXNATURAL)' 21X '#A5: '
              #A5 (EM=.GOXXXXXNATURAL) //
*
*
*  MOVE EDITED operand1 TO operand2 (EM=value)
*
*  The value of operand1 will be placed into operand2 using the
```

```
*  edit mask. The edit mask specifies at which positions the contents
*  of operand1 should be used for operand2.
*  Extra characters in operand1 must be specified in the edit mask
*  to be ignored (in operand2).
*
*
#A8 := '+12.40'
MOVE EDITED #A8 TO #NUM (EM=SZ9.99)
WRITE 'MOVE EDITED ''+12.40''      TO #NUM (EM=SZ9.99)'    6X '=' #NUM
*
*  Ignore extra characters of operand1.
*
#A10:= '-12.40EUR'
MOVE EDITED #A10 TO #NUM (EM=-Z9.99EUR)
WRITE 'MOVE EDITED ''-12.40EUR''   TO #NUM (EM=-Z9.99EUR)' 3X '=' #NUM
*
*  Use filler '*' for leading zero and define the filler as first
*  character in EM.
*
#A10 := 'EUR*12.40-'
MOVE EDITED #A10 TO #NUM (EM=*EURZZ9.99-)
WRITE 'MOVE EDITED ''EUR*12.40-''  TO #NUM (EM=*EURZZ9.99-)'
              '=' #NUM /
*
*  Ignore extra characters of operand1 and, if needed, enclose them
*  within apostrophes.
*
MOVE EDITED  'HE4711L-ALOX WO5R-7LD' TO
              #A12 (EM=XX4711X-AXX'X'XXX5X-7XX)
WRITE 'MOVE EDITED ''HE4711L-ALOX WO5R-7LD'' TO'
            '#A12 (EM=XX4711X-AXX''X''XXX5X-7XX)'
WRITE 51X '=' #A12 /
*
*
END
```

Output of Program MOVEX3:

```
MOVE 23.45 TO #NUM
MOVE EDITED #NUM (EM=Z99.99+) TO #A8               #A8:  23.45+

MOVE -78.15 TO #NUM
MOVE EDITED #NUM (EM=xFULL-Z99.99) TO #A12         #A12: FULL-x78.15


MOVE '  AB ' TO #A5
MOVE EDITED #A5 (EM=.GOXXXXXE'X'TRA) TO #A12       #A12: GO..AB EXTRA

WRITE #A5 (EM=.GOXXXXXNATURAL)                     #A5:  GO..AB NATURAL


MOVE EDITED '+12.40'      TO #NUM (EM=SZ9.99)      #NUM:   12.40
```

```
MOVE EDITED '-12.40EUR'   TO #NUM (EM=-Z9.99EUR)   #NUM:  -12.40
MOVE EDITED 'EUR*12.40-'  TO #NUM (EM=*EURZZ9.99-) #NUM:  -12.40

MOVE EDITED 'HE4711L-ALOX WO5R-7LD' TO #A12 (EM=XX4711X-AXX'X'XXX5X-7XX)
                                                    #A12: HELLO WORLD
```

Beispiel 7 – MOVE EDITED mit Editiermaske und Platzhalterzeichen '?'

```
** Example 'MOVEX4': MOVE EDITED with EM and wildcard character '?'
************************************************************************
DEFINE DATA LOCAL
1 #NUM  (N3.2)
1 #NUM2 (N5)
1 #A10  (A10)
1 #A14  (A14)
1 #B4   (B4)
1 #DATE (D)
1 #TIME (T)
END-DEFINE
*
*  MOVE EDITED operand1 TO operand2 (EM=value)
*
*  The value of operand1 will be placed into operand2 using the
*  edit mask. The edit mask specifies at which positions the contents
*  of operand1 should be used for operand2.
*  Extra characters in operand1 must be specified in the edit mask
*  to be ignored (in operand2).
*  Or use the wildcard character '?' instead of any extra character
*  which fits all values and much easier to define.
*
*
*  Numeric edit mask with extra characters or wildcard character
*
#A10 := '-12.40EUR'
MOVE EDITED #A10 TO #NUM (EM=-Z9.99EUR)
WRITE 'MOVE EDITED ''-12.40EUR''  TO #NUM (EM=-Z9.99EUR)' 9X '=' #NUM
*
#A10 := '-12.40EUR'
MOVE EDITED #A10 TO #NUM (EM=-Z9.99???)
WRITE 'MOVE EDITED ''-12.40EUR''  TO #NUM (EM=-Z9.99???)' 9X '=' #NUM //
*
*  Filler '*' is required if started with extra or wildcard character
*
#A10 := 'EUR*5.33-'
MOVE EDITED #A10 TO #NUM (EM=*EURZZ9.99-)
WRITE 'MOVE EDITED ''EUR*5.33-'' TO #NUM (EM=*EURZZ9.99-)' 8X
            '=' #NUM
*
#A10 := 'EUR*5.33-'
MOVE EDITED #A10 TO #NUM (EM=*???ZZ9.99-)
WRITE 'MOVE EDITED ''EUR*5.33-'' TO #NUM (EM=*???ZZ9.99-)' 8X
```

```
            '=' #NUM //
*
*
#A10 := 'ABC987CDEF'
MOVE EDITED #A10 TO #NUM2 (EM=*'ABC'999'CDEF')
WRITE 'MOVE EDITED ''ABC987CDEF'' TO #NUM2 (EM=*''ABC''999''CDEF'')' 2X
            '=' #NUM2
*
#A10 := 'ABC987CDEF'
MOVE EDITED #A10 TO #NUM2 (EM=*???999????)
WRITE 'MOVE EDITED ''ABC987CDEF'' TO #NUM2 (EM=*???999????)' 6X
            '=' #NUM2 //
*
*
*  Alphanumeric edit mask with extra characters or wildcard character
*
MOVE EDITED 'NATURAL-VERSION-9.2' TO
              #A10 (EM=XXXURAL-XXXXION-X.X)
WRITE 'MOVE EDITED ''NATURAL-VERSION-9.2'' TO '-
            '#A10 (EM=XXXURAL-XXXXION-X.X)'
WRITE 56X '=' #A10
*
MOVE EDITED 'NATURAL-VERSION-9.2' TO
              #A10 (EM=XXX????XXXX????X?X)
WRITE 'MOVE EDITED ''NATURAL-VERSION-9.2'' TO '-
            '#A10 (EM=XXX????XXXX????X?X)'
WRITE 56X '=' #A10 /
*
*  Filler '*' is required if started with extra or wildcard character
*
MOVE EDITED   'VENUS-SATURN-GALAXY 2050+' TO
              #A14 (EM=*VEXUS-SXXXXN-GXXA'X'YXXXXXX)
WRITE 'MOVE EDITED ''VENUS-SATURN-GALAXY 2050+'' TO'
WRITE 3X  '#A14 (EM=*VEXUS-SXXXXN-GXXA''X''YXXXXXX)'
            15X '=' #A14
*
MOVE EDITED   'VENUS-SATURN-GALAXY 2050+' TO
              #A14 (EM=*??X????XXXX???XX???XXXXXX)
WRITE 'MOVE EDITED ''VENUS-SATURN-GALAXY 2050+'' TO '
WRITE 3X '#A14 (EM=*??X????XXXX???XX???XXXXXX)'
            17X '=' #A14
*
*
NEWPAGE
*
*
*  Hexadecimal edit mask with extra characters or wildcard character
*
MOVE EDITED 'C1/C2 C3-C4' TO #B4 (EM=H/H' 'H-H)
WRITE 'MOVE EDITED ''C1/C2 C3-C4'' TO '-
            '#B4 (EM=H/H'' ''H-H)' 8X '=' #B4
*
```

```
MOVE EDITED 'C1/C2 C3-C4' TO #B4 (EM=H?H?H?H)
WRITE 'MOVE EDITED ''C1/C2 C3-C4'' TO '-
            '#B4 (EM=H?H?H?H)' 10X '=' #B4 //
*
*  Date and time edit mask with extra characters or wildcard character
*
MOVE EDITED 'DATE-07/10/2026-CREATED' TO
              #DATE (EM='DATE-'DD/MM/YYYY'-CREATED')
WRITE 'MOVE EDITED ''DATE-07/10/2026-CREATED'' TO '-
            '#DATE (EM=''DATE-''DD/MM/YYYY''-CREATED'')'
WRITE '=' #DATE (EM=YYYY-LLL-DD)
            6X '/* date shown with EM=(YYYY-LLL-DD)'/
*
MOVE EDITED 'DATE-07/10/2026-CREATED' TO
              #DATE (EM=?????DD/MM/YYYY????????)
WRITE 'MOVE EDITED ''DATE-07/10/2026-CREATED'' TO '-
            '#DATE (EM=?????DD/MM/YYYY????????)'
WRITE '=' #DATE (EM=YYYY-LLL-DD)
            6X '/* date shown with EM=(YYYY-LLL-DD)'//
*
*
MOVE EDITED 'RUN-TIME-17.37.45' TO #TIME (EM='RUN-TIME-'HH.II.SS)
WRITE 'MOVE EDITED ''RUN-TIME-17.37.45'' TO '-
            '#TIME (EM=''RUN-TIME-''HH.II.SS)'
WRITE '=' #TIME (EM=HH:II:SS)
            9X '/* time shown with EM=(HH:II:SS)'/
*
MOVE EDITED 'RUN-TIME-17.37.45' TO #TIME (EM=?????????HH.II.SS)
WRITE 'MOVE EDITED ''RUN-TIME-17.37.45'' TO '-
            '#TIME (EM=?????????HH.II.SS)'
WRITE '=' #TIME (EM=HH:II:SS)
            9X '/* time shown with EM=(HH:II:SS)'//
*
*
END
```

Output of Program MOVEX4:

```
MOVE EDITED '-12.40EUR'  TO #NUM (EM=-Z9.99EUR)         #NUM:  -12.40
MOVE EDITED '-12.40EUR'  TO #NUM (EM=-Z9.99???)         #NUM:  -12.40


MOVE EDITED 'EUR*5.33-' TO #NUM (EM=*EURZZ9.99-)        #NUM:   -5.33
MOVE EDITED 'EUR*5.33-' TO #NUM (EM=*???ZZ9.99-)        #NUM:   -5.33


MOVE EDITED 'ABC987CDEF' TO #NUM2 (EM=*'ABC'999'CDEF')  #NUM2:    987
MOVE EDITED 'ABC987CDEF' TO #NUM2 (EM=*???999????)      #NUM2:    987


MOVE EDITED 'NATURAL-VERSION-9.2' TO #A10 (EM=XXXURAL-XXXXION-X.X)
                                                        #A10: NATVERS92
```

```
MOVE EDITED 'NATURAL-VERSION-9.2' TO #A10 (EM=XXX????XXXX????X?X)
                                                    #A10: NATVERS92

MOVE EDITED 'VENUS-SATURN-GALAXY 2050+' TO
   #A14 (EM=*VEXUS-SXXXXN-GXXA'X'YXXXXXX)            #A14: NATURAL 2050+
MOVE EDITED 'VENUS-SATURN-GALAXY 2050+' TO
   #A14 (EM=*??X????XXXX???XX???XXXXXX)              #A14: NATURAL 2050+



MOVE EDITED 'C1/C2 C3-C4' TO #B4 (EM=H/H' 'H-H)       #B4: C1C2C3C4
MOVE EDITED 'C1/C2 C3-C4' TO #B4 (EM=H?H?H?H)         #B4: C1C2C3C4


MOVE EDITED 'DATE-07/10/2026-CREATED' TO #DATE (EM='DATE-'DD/MM/YYYY'-CREATED')
#DATE: 2026-Oct-07      /* date shown with EM=(YYYY-LLL-DD)

MOVE EDITED 'DATE-07/10/2026-CREATED' TO #DATE (EM=?????DD/MM/YYYY????????)
#DATE: 2026-Oct-07      /* date shown with EM=(YYYY-LLL-DD)


MOVE EDITED 'RUN-TIME-17.37.45' TO #TIME (EM='RUN-TIME-'HH.II.SS)
#TIME: 17:37:45         /* time shown with EM=(HH:II:SS)

MOVE EDITED 'RUN-TIME-17.37.45' TO #TIME (EM=?????????HH.II.SS)
#TIME: 17:37:45         /* time shown with EM=(HH:II:SS)
```

85 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 liegende 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 wird empfohlen, vorhandene MOVE INDEXED-Statement durch MOVE-Statements zu ersetzen.

Siehe Statement MOVE.

86 MULTIPLY

- Funktion MULTIPLY 690
- Syntax 1 - MULTIPLY-Statement ohne GIVING-Klausel 690
- Syntax 2 - MULTIPLY-Statement mit GIVING-Klausel 691
- Beispiel für MULTIPLY-Statement 692

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: ADD | COMPRESS | COMPUTE | DIVIDE | EXAMINE | MOVE | MOVE ALL | RESET | SEPARATE | SUBTRACT

Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion MULTIPLY

Das Statement MULTIPLY dient dazu, zwei Operanden miteinander zu multiplizieren. Das Ergebnis der Multiplikation kann, je nach verwendeter Syntax, entweder in *operand1* oder in *operand3* gespeichert 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.

Bezüglich Multiplikationen mit Arrays siehe auch *Arithmetische Operationen mit Arrays* im *Leitfaden zur Programmierung*.

Dieses Statement hat zwei verschiedene Syntax-Strukturen.

Syntax 1 - MULTIPLY-Statement ohne GIVING-Klausel

Wenn Syntax 1 benutzt wird, kann das Ergebnis der Multiplikation in *operand1* gespeichert werden.

```
MULTIPLY [ROUNDED] operand1 BY { (arithmetic-expression) }
                               { 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
operand1		S	A		M		N	P	I	F							ja	nein
operand2	C	S	A		N		N	P	I	F							ja	nein

Syntax-Element-Beschreibung (Syntax 1):

Syntax-Element	Beschreibung
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 BY *operand2*	**Operanden:** *operand1* ist der Multiplikand, *operand2* ist der Multiplikator. Das Ergebnis wird in *operand1* gespeichert. Somit ist das Statement gleichbedeutend mit: `operand1 := operand1 * operand2`
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet, bevor es *operand1* oder *operand3* zugewiesen wird. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Syntax 2 - MULTIPLY-Statement mit GIVING-Klausel

Wenn Syntax 2 benutzt wird, kann das Ergebnis der Multiplikation in *operand3* gespeichert werden.

```
MULTIPLY      { (arithmetic-expression) }     { (arithmetic-expression) }     GIVING
[ROUNDED]     { operand1                } BY  { operand2                }     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
operand1	C	S	A		M			N	P	I	F							ja	nein
operand2	C	S	A		N			N	P	I	F							ja	nein
operand3		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):

Syntax-Element	Beschreibung
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
operand1 BY *operand2* GIVING *operand3*	**Operanden:** *operand1* ist der Multiplikand, *operand2* ist der Multiplikator. Das Ergebnis wird in *operand3* gespeichert. Somit ist das Statement gleichbedeutend mit: `operand3 := operand1 * operand2`
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet, bevor es *operand1* oder *operand3* zugewiesen wird. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Beispiel für MULTIPLY-Statement

```
** 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`:

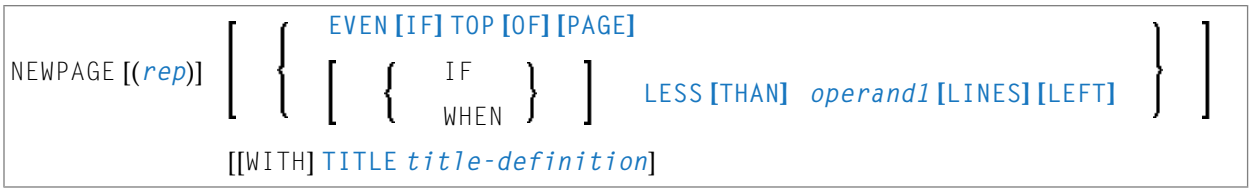
```
MULTIPLY #A BY 3                        #A:   60
MULTIPLY #A BY 3 GIVING #B              #B:    180
MULTIPLY ROUNDED 3 BY 3.5 GIVING #C     #C:   10.5
MULTIPLY  3 BY -4 GIVING #D             #D: -12
MULTIPLY -3 BY -4 GIVING #D             #D:  12
MULTIPLY  3 BY  0 GIVING #D             #D:   0

#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
```

87 NEWPAGE

- Funktion NEWPAGE 696
- Syntax-Beschreibung NEWPAGE 696
- Beispiel für NEWPAGE-Statement 697



Dieser Abschnitt 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 NEWPAGE

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 NEWPAGE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S				N	P	I								ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(rep)`	**Report-Spezifikation:** Mit der Notation `(rep)` kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement `NEWPAGE` 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 mit `(rep)` nichts anderes angegeben wird, bezieht sich das `NEWPAGE`-Statement auf den ersten Report (Report 0). Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, siehe *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
`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` *operand1* `LINES LEFT`	Mit dieser Option bewirken Sie, dass das `NEWPAGE`-Statement ausgeführt wird, wenn auf der aktuellen Seite weniger als *operand1* Zeilen übrig sind. Der interne Zeilenzähler orientiert sich hierbei am Wert des Profil-/Session-Parameters `PS`.
`WITH TITLE` *title-definition*	Diese Option können Sie verwenden, um auf der generierten neuen Seite eine Überschrift auszugeben. Für die *title-definition* gilt dieselbe Syntax wie für das Statement `WRITE TITLE`, außer dass die `SKIP`-Klausel in einem `NEWPAGE WITH TITLE` *title-definition*-Statement nicht erlaubt ist.

Beispiel für NEWPAGE-Statement

```
** 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   CURRENCY
                                            SALARY     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
****************************************
```

88 OBTAIN

- Funktion OBTAIN 702
- Einschränkung bei OBTAIN 703
- Syntax-Beschreibung OBTAIN 703
- Beispiele OBTAIN 707

```
OBTAIN  operand1 ...
```

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

Gehört zur Funktionsgruppe: ***Reporting Mode-Statements***

Dieser Abschnitt behandelt folgende Themen:

Funktion OBTAIN

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 bei OBTAIN

Das `OBTAIN`-Statement gilt nur für den Reporting Mode.

Syntax-Beschreibung OBTAIN

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L				ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	Als *operand1* 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



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         DINERS CLUB
AMERICAN EXPRESS



AVIS                AVIS
AMERICAN EXPRESS



HERTZ               HERTZ
AMERICAN EXPRESS



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 OBTAIN

- 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
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*.

89 ON ERROR

- Funktion ON ERROR 712
- Einschränkung bei ON ERROR 712
- Syntax-Beschreibung ON ERROR 713
- ON ERROR-Verarbeitung in Subprogrammen 713
- Systemvariablen *ERROR-NR und *ERROR-LINE 714
- Beispiel für ON ERROR-Statement 714

Structured Mode-Syntax

```
ON ERROR
statement ...
END-ERROR
```

Reporting Mode-Syntax

```
ON ERROR { statement ...          }
         { DO statement ... DOEND }
```

Dieser Abschnitt 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 ON ERROR

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 bei ON ERROR

Jedes Natural-Objekt darf maximal ein ON ERROR-Statement enthalten.

Syntax-Beschreibung ON ERROR

Syntax-Element	Beschreibung
statement...	**Festlegung der ON ERROR-Verarbeitung:** Um die Verarbeitung zu definieren, die stattfinden soll, wenn eine ON ERROR-Bedingung aufgetreten ist, können Sie eines oder mehrere Statements angeben. **Verlassen eines ON ERROR-Blocks:** Ein ON ERROR-Statement-Block kann mit einem der Statements FETCH, STOP, TERMINATE, RETRY, ESCAPE ROUTINE oder ESCAPE MODULE verlassen werden. Wird der Block nicht mit einem dieser Statements verlassen, gibt Natural eine entsprechende Fehlermeldung aus und bricht die Ausführung des Programms ab.
END-ERROR *statement ...* DO *statement ...* DOEND	**Ende des ON-ERROR-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-ERROR zum Beenden des ON-ERROR-Statements benutzt werden. Im Reporting Mode werden die Statements DO ... DOEND benutzt, um eines oder mehrere Statements anzugeben, und um das ON-ERROR-Statement zu beenden. Falls Sie nur ein einziges Statement angeben, können Sie die Statements DO ... DOEND weglassen. Das ist jedoch nicht im Sinne einer guten Kodierpraxis und wird nicht empfohlen.

ON ERROR-Verarbeitung in Subprogrammen

Wird mittels CALLNAT, PERFORM oder FETCH RETURN eine Subprogramm-Struktur aufgebaut, so darf jedes Modul dieser Struktur ein ON ERROR-Statement enthalten.

Tritt ein Fehler auf, so verfolgt Natural die Subprogramm-Struktur automatisch zurück und wählt das erste in einem Subprogramm 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 Quellcode-Zeile, in der das Statement steht, das den Fehler verursacht hat.

Beispiel für ON ERROR-Statement

```
** 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
```

90 OPEN CONVERSATION

- Funktion OPEN CONVERSATION 718
- Syntax-Beschreibung OPEN CONVERSATION 718
- Weitere Informationen und Beispiele zu OPEN CONVERSATION 719

```
OPEN CONVERSATION USING[SUBPROGRAMS] {operand1}...
```

Dieser Abschnitt 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 OPEN CONVERSATION

Das Statement `OPEN CONVERSATION` wird im Zusammenhang mit dem Natural RPC (Remote Procedure Call) verwendet. Damit ist es möglich, eine Konversation von der Client-Seite aus zu öffnen und die entfernten (remote) Subprogramme anzugeben, die an der Konversation beteiligt sein sollen.

Wenn das `OPEN CONVERSATION`-Statement ausgeführt wird, weist es der Systemvariablen `*CONVID` eine eindeutige Konversationskennung zu, die die Konversation identifiziert.

Syntax-Beschreibung OPEN CONVERSATION

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A			A	ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Namen der beteiligten Subprogramme:** Als *operand1* geben Sie die Namen der Subprogramme an, die an der Konversation beteiligt sein sollen. Der Name eines Subprogramms kann entweder als 1 bis 8 Zeichen lange Konstante oder als alphanumerische Variable mit Länge von 1 bis 8 angegeben werden.

Weitere Informationen und Beispiele zu OPEN CONVERSATION

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

- *Natural RPC-Betrieb im konversationellen Modus*
- *Verwendung des Natural RPC im konversationellen Modus*

91 OPTIONS

- Funktion OPTIONS 722
- Verarbeitung von mehreren OPTIONS-Statements 722

```
OPTIONS parameter...
```

Dieser Abschnitt behandelt folgende Themen:

Funktion OPTIONS

Das Statement `OPTIONS` kann verwendet werden, um Kompilierungsoptionen für das aktuelle Natural-Objekt anzugeben. Es handelt sich um dieselben Optionen, die an folgenden Stellen angegeben werden können:

- statisch mit dem Makro `NTCMPO`,
- dynamisch mit dem Parameter `CMPO`,
- innerhalb einer Natural-Session mit dem Systemkommando `COMPOPT`.

Darüber hinaus kann das Statement `OPTIONS` verwendet werden, um Optionen für den Natural Optimizer Compiler anzugeben. Diese Optionen sind in der *Natural Optimizer Compiler*-Dokumentation beschrieben.

Mit der Option `MCG` angegebene Natural Optimizer Compiler-Optionen werden auf Gültigkeit hin überprüft, auch wenn der Natural Optimizer Compiler nicht installiert ist.

Verarbeitung von mehreren OPTIONS-Statements

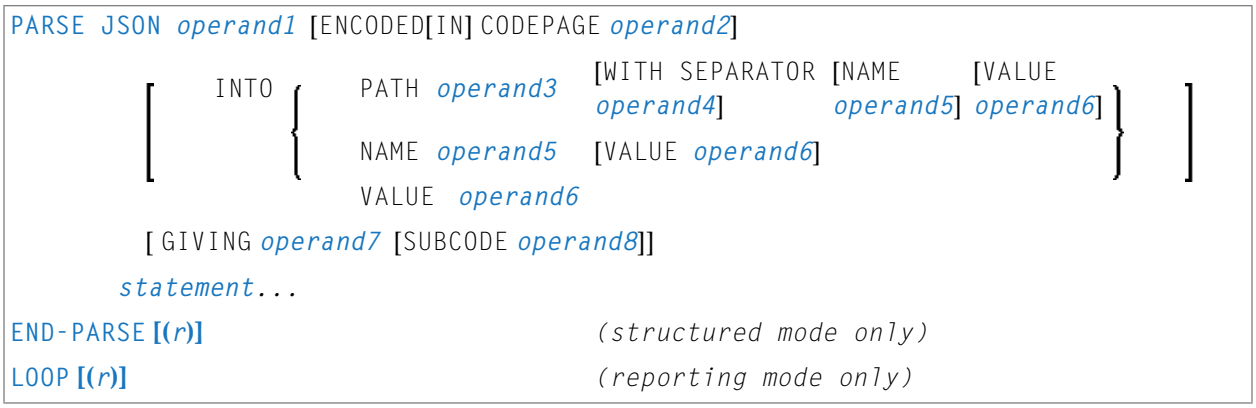
Wenn mehrere `OPTIONS`-Statements innerhalb desselben Natural-Objekts angegeben sind, erlangen die Optionseinstellungen sofort Gültigkeit. Dies ist allerdings nicht der Fall bei den Optionen `PSIGNF`, `TSENABL`, `GFID`, `DB2BIN` und `DB2PKYU`. Bei diesen Optionen gilt der mit dem *letzten* `OPTIONS`-Statement angegebene Optionswert.

X

- 92 PARSE JSON 725
- 93 PARSE XML 737
- 94 PASSW 751
- 95 PERFORM 755
- 96 PERFORM BREAK PROCESSING 763
- 97 PRINT 767
- 98 PROCESS 777
- 99 PROCESS COMMAND 781
- 100 PROCESS PAGE 799
- 101 PROCESS SQL (SQL) 813
- 102 PROPERTY 817

92 PARSE JSON

- Funktion PARSE JSON 726
- Syntax-Beschreibung PARSE JSON 727
- Beispiele PARSE JSON 730



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

Verwandtes Statement: `REQUEST DOCUMENT`.

Gehört zur Funktionsgruppe: ***Internet und Parsing***.

Funktion PARSE JSON

Das Statement `PARSE JSON` ermöglicht es Ihnen, XML-Dokumente aus einem Natural-Programm zu parsen. Siehe auch *Statements für den Internet- und JSON-XML-Zugriff* im *Leitfaden zur Programmierung*.

Es empfiehlt sich, dynamische Variablen zu benutzen, wenn Sie das Statement `PARSE JSON` verwenden. Der Grund dafür ist, dass es unmöglich ist, die Länge einer statischen Variablen zu ermitteln. Der Einsatz von statischen Variablen könnte zum Abschneiden des Wertes führen, der in die Variable geschrieben werden soll.

Informationen zur Unicode-Unterstützung siehe `PARSE JSON` in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Markup

Die folgenden Markierungen werden in Pfadzeichenketten verwendet, um die verschiedenen Datenstrukturen und ihre Zustände in einem JSON-Dokument darzustellen:

Markierung	JSON-Daten	Position in der Pfad-Zeichenkette
<	Anfang eines Objekts	Am Anfang, am Ende oder zwischen Feldnamen
>	Ende eines Objekts	Am Ende
(	Anfang eines Arrays	Am Anfang, am Ende oder zwischen Feldnamen.
)	Ende eines Arrays	Am Ende
/	Separator **Anmerkung:** Die Trennmarkierung können Sie im *operand2* anpassen.	
$	Geparste Daten - Datenzeichenkette	Am Ende

Durch die Verwendung dieses zusätzlichen Markups im Pfadstring können Sie die Elemente des JSON-Dokuments in der Ausgabe leichter unterscheiden.

Zugehörige Systemvariablen

Die folgenden Natural-Systemvariablen werden für jedes ausgeführte `PARSE JSON`-Statement automatisch erstellt:

- `*PARSE-TYPE`
- `*PARSE-LEVEL`
- `*PARSE-INDEX`

Durch Angabe der Notation `(r)` hinter `*PARSE-TYPE`, `*PARSE-LEVEL` und `*PARSE-INDEX` können Sie das Statement-Label bzw. die Statement-Nummer des Statements angeben, in dem das `PARSE`-Statement abgesetzt wurde. Wenn `(r)` nicht angegeben wird, stellt die betreffende Systemvariable die Systemvariable der JSON-Daten dar, die gerade in der zurzeit aktiven `PARSE`-Verarbeitungsschleife abgearbeitet werden.

Weitere Informationen über diese Systemvariablen finden Sie in der *Systemvariablen*-Dokumentation.

Syntax-Beschreibung PARSE JSON

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A	U	B									ja	nein
operand2	C	S				A											ja	nein
operand3		S				A	U	B									ja	ja
operand4	C	S				A	U	B									ja	nein
operand5		S				A	U	B									ja	ja
operand6		S				A	U	B									ja	ja
operand7		S							I4								ja	ja
operand8		S							I4								ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**JSON-Dokument:** *operand1* stellt das zu parsende JSON-Dokument dar. Das JSON-Dokument kann nicht geändert werden, während es vom Parser abgearbeitet wird. Wenn Sie versuchen, während des Parse-Vorgangs das JSON-Dokument zu ändern, hat die Änderung keine Auswirkung auf den PARSE-Vorgang.
ENCODED [IN] CODEPAGE *operand2*	**ENCODED [IN] CODEPAGE:** *operand2* bezeichnet die Codepage des in *operand1* dargestellten Dokuments.
PATH *operand3*	**Pfad:** *operand3* stellt den Pfad der Daten im JSON-Dokument dar. Der Pfad (PATH) enthält den Namen (NAME) des identifizierten JSON-Teils, die Namen aller übergeordneten Elemente (Parents), sowie den Typ des XML-Teils. Jeder Name in PATH wird durch das in *operand4* angegebene, „Separator“ genannte Zeichen abgetrennt. **Anmerkung:** Die mit PATH angegebenen Informationen erleichtern den Aufbau einer Baumstruktur. Siehe auch ***Beispiel 1 - Benutzung der PATH-Option***.
SEPARATOR *operand4*	**Separator:** *operand4* stellt das Trennzeichen (Separator) dar, das zur Trennung von NAME und Mark-Up im PATH (*operand3*) verwendet wird. Das im PATH (*operand3*) verwendete Standard-Trennzeichen ist ein Schrägstrich (/). Siehe auch ***Beispiel 2 - Benutzung der PATH-Option mit SEPARATOR***.
NAME *operand5*	**Datenelementname:** *operand5* stellt den Namen (NAME) eines Datenelements in dem JSON-Dokument dar.

Syntax-Element	Beschreibung
	Wenn NAME keinen Wert hat, dann wird die damit verbundene dynamische Variable auf *LENGTH()=0 gesetzt, welche eine mit einem Leerzeichen gefüllte statische Variable ist. Siehe auch ***Beispiel 3 - Benutzung der NAME-Option***.
VALUE *operand6*	**Datenelementinhalt:** *operand6* stellt den Inhalt (VALUE) eines Datenelements in dem JSON-Dokument dar. Wenn kein Wert vorhanden ist, dann wird die damit verbundene dynamische Variable auf *LENGTH()=0 gesetzt, welche eine mit einem Leerzeichen gefüllte statische Variable ist. Siehe auch ***Beispiel 4 - Benutzung der VALUE-Option***.
GIVING *operand7*	**GIVING-Parameter:** Wenn der PARSE-Prozess zur Laufzeit auf einen Fehler stößt, enthält *operand7* die 4-stellige Natural-Fehlernummer. *operand7* wird am Ende jeder Iteration eines Parse-Prozesses aktualisiert. Wenn der Parameter GIVING angegeben ist, gibt *operand7* 0 zurück, wenn kein Fehler vorliegt. Tritt jedoch ein PARSE-Fehler auf, gibt *operand7* eine entsprechende Natural-Fehlernummer zurück. Wenn ein Fehler auftritt und der GIVING-Parameter angegeben ist, wird die PARSE-Schleife beendet und die Kontrolle an das auf END-PARSE folgende Statement zurückgegeben. Wenn GIVING nicht gesetzt ist und ein Fehler auftritt, wird eine Natural-Fehlermeldung zurückgegeben und die Programmausführung unterbrochen, sofern nicht der ON ERROR- Statement-Block verwendet wird. Siehe auch ***Beispiel 5 - Benutzung der GIVING- und SUBCODE-Klauseln***.
SUBCODE *operand8*	**SUBCODE-Parameter:** Wenn der PARSE-Prozess zur Laufzeit auf einen Fehler stößt, enthält *operand8* den dreistelligen Reason Code, der mit der Fehlernummer in *operand7* korrespondiert. *operand8* wird am Ende jeder Iteration eines Parse-Prozesses aktualisiert. Wenn der SUBCODE-Parameter angegeben ist, gibt *operand8* 0 zurück, wenn kein Fehler vorliegt. Wenn jedoch ein PARSE-Fehler auftritt, gibt *operand8* einen entsprechenden Reason Code zurück. SUBCODE betrifft nur Reason Codes für den Fehler NAT8331. Wenn der SUBCODE-Operand zusammen mit GIVING angegeben wird und ein Fehler auftritt, wird die PARSE-Schleife abgebrochen und die Kontrolle an das auf END-PARSE folgende Statement zurückgegeben.

Syntax-Element	Beschreibung
	Anmerkung: Eine Liste aller Reason Codes für den Natural-Fehler NAT8331 finden Sie im Kapitel *JSON-Related Reason Codes for Error NAT8331* in der *Messages and Codes*-Dokumentation. Siehe auch ***Beispiel 5 - Benutzung der GIVING- und SUBCODE-Klauseln***.
`END-PARSE` (*r*) `LOOP` (*r*)	**Ende des PARSE JSON-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-PARSE` zum Beenden des `PARSE JSON`-Statements benutzt werden. Im Structured Mode können Sie bei `END-PARSE` Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement `LOOP` zum Beenden des `PARSE JSON`-Statements benutzt. Im Reporting Mode können Sie bei `LOOP` Labels oder Zeilennummern angeben.

Beispiele PARSE JSON

- Beispiel 1 - Benutzung der PATH-Option
- Beispiel 2 - Benutzung der PATH-Option mit SEPARATOR
- Beispiel 3 - Benutzung der NAME-Option
- Beispiel 4 - Benutzung der VALUE-Option
- Beispiel 5 - Benutzung der GIVING- und SUBCODE-Klauseln

Beispiel 1 - Benutzung der PATH-Option

Der folgende Code

```
** Example 'PAJSNEX1': PARSE JSON (with PATH and CODEPAGE)
**
** Note: Definition of variable MYJSON needs TQMARK set to OFF.
************************************************************************
OPTIONS TQMARK=OFF          /* Translate quotation mark
*
DEFINE DATA LOCAL
1 MYJSON      (A) DYNAMIC
1 MYCODEPAGE  (A) DYNAMIC
1 MYPATH      (A) DYNAMIC
END-DEFINE
*
COMPRESS '{'
         '  "employee": {'
         '    "@personnel-id": "30016315",'
         '    "full-name": {'
         '      "first-name": "RICHARD",'
```

```
          '      "name": "FORDHAM"'
          '    }'
          '  }'
          '}'
INTO MYJSON LEAVING NO
*
MYCODEPAGE := *CODEPAGE
*
PARSE JSON MYJSON ENCODED IN CODEPAGE MYCODEPAGE
                  INTO PATH MYPATH
   PRINT MYPATH
END-PARSE
END
```

erzeugt die folgende Ausgabe:

```
<
</employee
</employee/<
</employee/</@personnel-id
</employee/</@personnel-id/$
</employee/</full-name
</employee/</full-name/<
</employee/</full-name/</first-name
</employee/</full-name/</first-name/$
</employee/</full-name/</name
</employee/</full-name/</name/$
</employee/</full-name/>
</employee/>
>
```

Beispiel 2 - Benutzung der PATH-Option mit SEPARATOR

Der folgende Code

```
** Example 'PAJSNEX2': PARSE JSON (with PATH and SEPARATOR)
**
** Note: Definition of variable MYJSON needs TQMARK set to OFF.
************************************************************************
OPTIONS TQMARK=OFF          /* Translate quotation mark
*
DEFINE DATA LOCAL
1 MYJSON      (A) DYNAMIC
1 MYCODEPAGE  (A) DYNAMIC
1 MYPATH      (A) DYNAMIC
1 MYSEPARATOR (A1)
END-DEFINE
*
COMPRESS '{'
          '  "employee": {'
          '    "@personnel-id": "30016315",'
```

```
          '    "full-name": {'
          '      "first-name": "RICHARD",'
          '      "name": "FORDHAM"'
          '    }'
          '  }'
          '}'
INTO MYJSON LEAVING NO
*
MYCODEPAGE := *CODEPAGE
MYSEPARATOR := '*'
*
PARSE JSON MYJSON ENCODED IN CODEPAGE MYCODEPAGE
                  INTO PATH MYPATH WITH SEPARATOR MYSEPARATOR
   PRINT MYPATH
END-PARSE
END
```

erzeugt die folgende Ausgabe:

```
<
<*employee
<*employee*<
<*employee*<*@personnel-id
<*employee*<*@personnel-id*$
<*employee*<*full-name
<*employee*<*full-name*<
<*employee*<*full-name*<*first-name
<*employee*<*full-name*<*first-name*$
<*employee*<*full-name*<*name
<*employee*<*full-name*<*name*$
<*employee*<*full-name*>
<*employee*>
>
```

Beispiel 3 - Benutzung der NAME-Option

Der folgende Code

```
** Example 'PAJSNEX3': PARSE JSON (with PATH and NAME)
**
** Note: Definition of variable MYJSON needs TQMARK set to OFF.
************************************************************************
OPTIONS TQMARK=OFF          /* Translate quotation mark
*
DEFINE DATA LOCAL
1 MYJSON      (A) DYNAMIC
1 MYPATH      (A) DYNAMIC
1 MYNAME      (A) DYNAMIC
END-DEFINE
*
COMPRESS '{'
```

```
         '  "employee": {'
         '    "@personnel-id": "30016315",'
         '    "full-name": {'
         '      "first-name": "RICHARD",'
         '      "name": "FORDHAM"'
         '    }'
         '  }'
         '}'
INTO MYJSON LEAVING NO
*
PARSE JSON MYJSON INTO PATH MYPATH NAME MYNAME
   DISPLAY (AL=39) MYPATH MYNAME
END-PARSE
END ↩
```

erzeugt die folgende Ausgabe:

```
                MYPATH                                  MYNAME
-----------------------------------    -----------------------------------


<
</employee                               employee
</employee/<                             employee
</employee/</@personnel-id               @personnel-id
</employee/</@personnel-id/$
</employee/</full-name                   full-name
</employee/</full-name/<                 full-name
</employee/</full-name/</first-name      first-name
</employee/</full-name/</first-name/$
</employee/</full-name/</name            name
</employee/</full-name/</name/$
</employee/</full-name/>
</employee/>
>
```

Beispiel 4 - Benutzung der VALUE-Option

Der folgende Code

```
** Example 'PAJSNEX4': PARSE JSON (with PATH and VALUE)
**
** Note: Definition of variable MYJSON needs TQMARK set to OFF.
************************************************************************
OPTIONS TQMARK=OFF          /* Translate quotation mark
*
DEFINE DATA LOCAL
1 MYJSON      (A) DYNAMIC
1 MYPATH      (A) DYNAMIC
1 MYVALUE     (A) DYNAMIC
END-DEFINE
*
```

```
COMPRESS '{'
         '  "employee": {'
         '    "@personnel-id": "30016315",'
         '    "full-name": {'
         '      "first-name": "RICHARD",'
         '      "name": "FORDHAM"'
         '    }'
         '  }'
         '}'
INTO MYJSON LEAVING NO
*
PARSE JSON MYJSON INTO PATH MYPATH VALUE MYVALUE
   DISPLAY (AL=39) MYPATH MYVALUE
END-PARSE
END
```

erzeugt die folgende Ausgabe:

```
                MYPATH                                      MYVALUE
------------------------------------ ----------------------------------------

<
</employee
</employee/<
</employee/</@personnel-id
</employee/</@personnel-id/$            30016315
</employee/</full-name
</employee/</full-name/<
</employee/</full-name/</first-name
</employee/</full-name/</first-name/$   RICHARD
</employee/</full-name/</name
</employee/</full-name/</name/$         FORDHAM
</employee/</full-name/>
</employee/>
>
```

Beispiel 5 - Benutzung der GIVING- und SUBCODE-Klauseln

Das folgende Programm erzeugt einen Laufzeitfehler:

```
** Example 'PAJSNEX5': PARSE JSON (with GIVING and SUBCODE)
**
** Note: Definition of variable MYJSON needs TQMARK set to OFF.
************************************************************************
OPTIONS TQMARK=OFF          /* Translate quotation mark
*
DEFINE DATA LOCAL
1 MYJSON      (A) DYNAMIC
1 MYPATH      (A) DYNAMIC
1 MYNAME      (A) DYNAMIC
1 MYVALUE     (A) DYNAMIC
```

```
1 MYGIVING    (I4)
1 MYSUBCODE   (I4)
END-DEFINE
*
* Produce Natural runtime error with incorrect JSON document
*
COMPRESS '{'
         '  "employee": {'
         '    "@personnel-id": "30016315",'
         '    "full-name": {'
         '      "first-name": "RICHARD",'
         '      "FORDHAM"'     /* here the key 'name' is missing
         '    }'
         '  }'
         '}'
INTO MYJSON LEAVING NO
*
PARSE JSON MYJSON INTO PATH MYPATH NAME MYNAME VALUE MYVALUE
                  GIVING MYGIVING SUBCODE MYSUBCODE
   WRITE (AL=39) MYPATH
END-PARSE
*
IF MYGIVING NE 0
  WRITE / 'Error Number:'  MYGIVING 'Subcode:' MYSUBCODE
END-IF
END
```

Ausgabe des obigen Programms:

```
<
</employee
</employee/<
</employee/</@personnel-id
</employee/</@personnel-id/$
</employee/</full-name
</employee/</full-name/<
</employee/</full-name/</first-name
</employee/</full-name/</first-name/$
</employee/</full-name/</FORDHAM

Error Number:        8331 Subcode:           5
```

93 PARSE XML

- Funktion PARSE XML 738
- Syntax-Beschreibung PARSE XML 740
- Beispiele PARSE XML 743

```
PARSE XML operand1 [INTO [PATH operand2] [NAME operand3] [VALUE operand4]]
[[NORMALIZE] NAMESPACE operand5 PREFIX operand6]
[GIVING operand7 [SUBCODE operand8]]
statement...
END-PARSE [(r)]                          (structured mode only)
LOOP [(r)]                               (reporting mode only)
```

Dieser Abschnitt behandelt folgende Themen:

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

Verwandtes Statement: `REQUEST DOCUMENT`

Gehört zur Funktionsgruppe: ***Internet und XML***

Funktion PARSE XML

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 zum Abschneiden des Wertes führen, der in die Variable geschrieben werden soll.

Informationen zur Unicode-Unterstützung siehe *Unicode- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*, Unterabschnitt `PARSE XML` in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Markup

Im Folgenden finden Sie Markierungen, 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 <?XML...?>)	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 Quellcode-Zeilennummer des Statements angeben, in dem 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 PARSE XML

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A	U	B									ja	nein
operand2		S				A	U	B									ja	ja
operand3		S				A	U	B									ja	ja
operand4		S				A	U	B									ja	ja
operand5		S	A			A	U	B									ja	ja
operand6		S	A			A	U	B									ja	ja
operand7									I4								ja	ja
operand8									I4								ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**XML-Dokument:** *operand1* stellt das betreffende XML-Dokument dar. Das XML-Dokument kann nicht geändert werden, während es vom Parser abgearbeitet wird. Wenn Sie versuchen, während des Parse-Vorgangs das XML-Dokument zu ändern (indem Sie es zum Beispiel überschreiben), wird eine Fehlermeldung angezeigt.
operand2	**Pfad:** *operand2* stellt den Pfad der Daten im XML-Dokument dar. Der Pfad (PATH) enthält den Namen des identifizierten XML-Teils, die Namen aller Parents (übergeordneten Elemente), sowie den Typ des XML-Teils. **Anmerkung:** Die mit dem Pfad angegebenen Informationen erleichtern den Aufbau einer Baumstruktur. Siehe auch ***Beispiel 1 - Benutzung von operand2***.
operand3	**Datenelementname:** *operand3* stellt den Namen (NAME) eines Datenelements im XML-Dokument dar. 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. Siehe auch ***Beispiel 2 - Benutzung von operand3***.

Syntax-Element	Beschreibung
operand4	**Datenelementinhalt:** *operand4* stellt den Inhalt (VALUE) eines Datenelements im XML-Dokument dar. Ist kein Wert vorhanden, wird eine gegebene dynamische Variable auf *LENGTH()=0 gesetzt, welche eine mit einem Leerzeichen aufgefüllte statische Variable ist. Siehe auch ***Beispiel 3 - Benutzung von operand4***.
operand5 und *operand6* NORMALIZE NAMESPACE PREFIX	**Namespace URI und Präfix:** Der eindeutige Namen gewährleistende Namespace-URI oder Uniform Resource Identifier (*operand5*) und das Namespace-PREFIX (*operand6*) werden während der Laufzeit kopiert. Deshalb beeinflusst eine Änderung der Arrays für Namespace-Zuordnungen in der PARSE-Schleife nicht den Parser. *operand5* und *operand6* sind eindimensionale Arrays mit einer gleichen Anzahl von Ausprägungen. Namespace-Normalisierung ist eine Funktion des PARSE-Statements. Mit XML können Namespaces für die Element-Namen definiert werden:. `<myns:myentity xmlns:myns="http://myuri" />` Die Namespace-Definition besteht aus zwei Teilen: ■ einem Namespace-PREFIX (myns in diesem Fall) und ■ ein URI (myuri) zur Definition des Namespace. Der Namespace-PREFIX ist Teil des Elementnamens. Dies bedeutet, dass für das PARSE-Statement, vor allem für *operand2*, die generierten PATH-Strings vom Namespace-PREFIX abhängig sind. Wenn der Pfad in einem Natural-Programm zur Angabe 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:

Syntax-Element	Beschreibung
	``` 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 zum Präfix:** 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 auf einen Doppelpunkt reduziert. ■ Wenn die Präfix-Definition bei der Namespace-Normalisierung leer ist, dann wird das Namespace-Präfix gelöscht.
`GIVING` *operand7*	**GIVING:** Wenn der Parse-Prozess zur Laufzeit auf einen Fehler stößt, enthält *operand7* die 4-stellige Natural-Fehlernummer. *operand7* wird am Ende jeder Iteration eines Parse-Prozesses aktualisiert. `GIVING` betrifft nur den Fehlercode NAT8311. Wenn der `GIVING`-Parameter angegeben ist, gibt *operand7* eine `0` zurück, wenn kein Fehler vorliegt. Tritt jedoch ein Parse-Fehler auf, gibt *operand7* eine entsprechende Natural-Fehlernummer zurück. Wenn ein Fehler auftritt und der `GIVING`-Parameter angegeben ist, wird die Parse-Schleife beendet und die Kontrolle an das auf `END-PARSE` folgende Statement zurückgegeben. Wenn `GIVING` nicht gesetzt ist und ein Fehler auftritt, wird eine Natural-Fehlermeldung zurückgegeben und die Programmausführung unterbrochen, sofern nicht der `ON ERROR`-Statement-Block verwendet wird. Siehe auch ***Beispiel 6 - Benutzung der GIVING- und SUBCODE-Klauseln***.

Syntax-Element	Beschreibung
SUBCODE *operand8*	**SUBCODE-Parameter:** Wenn der Parse-Prozess zur Laufzeit auf einen Fehler stößt, enthält *operand8* den dreistelligen Reason Code, der mit der Fehlernummer in *operand7* korrespondiert. *operand8* wird am Ende jeder Iteration eines Parse-Prozesses aktualisiert. Wenn der SUBCODE-Parameter angegeben ist, gibt *operand8* eine 0 zurück, wenn kein Fehler vorliegt. Wenn jedoch ein PARSE-Fehler auftritt, gibt *operand8* einen entsprechenden Reason Code zurück. SUBCODE betrifft nur den Fehlercode NAT8311. Wenn der SUBCODE-Operand zusammen mit GIVING angegeben wird und ein Fehler auftritt, wird die PARSE-Schleife abgebrochen und die Kontrolle an das auf END-PARSE folgende Statement zurückgegeben. **Anmerkung:** Eine Liste aller Reason Codes zum Fehler NAT8311, die das PARSE XML-Statement betreffen, finden Sie im Kapitel *PARSE XML: Reason Codes for Error Message NAT8311* in der *Messages and Codes*-Dokumentation. Siehe auch ***Beispiel 6 - Benutzung der GIVING- und SUBCODE-Klauseln***.
END-PARSE (*r*) LOOP (*r*)	**Ende des PARSE XML-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-PARSE zum Beenden des PARSE XML-Statements benutzt werden. Im Structured Mode können Sie bei END-PARSE Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des PARSE XML-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

Beispiele PARSE XML

- 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 6 - Benutzung der GIVING- und SUBCODE-Klauseln

Beispiel 1 - Benutzung von operand2

Der folgende XML-Code

```
myxml := '<?xml version="1.0" ?>'-
         '<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" ?>'-
         '<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" ?>'-
         '<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" ?>'-
         '<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" ?>'-
         '<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//
```

Beispiel 6 - Benutzung der GIVING- und SUBCODE-Klauseln

Das folgende Programm erzeugt einen Laufzeitfehler:

```
** Example 'PAXMLEX4': PARSE XML (with GIVING and SUBCODE)
**
** Note: Natural session parameter XML has to be set at least:
**       XML=(ON,PARSE=ON).
************************************************************************
DEFINE DATA LOCAL
1 MYXML     (A) DYNAMIC
1 MYPATH    (A) DYNAMIC
1 MYNAME    (A) DYNAMIC
```

```
1 MYVALUE   (A) DYNAMIC
1 MYGIVING  (I4)
1 MYSUBCODE (I4)
END-DEFINE
*
* Produce Natural runtime error with incorrect XML document
*
COMPRESS '<?xml version="1.0"  ?>'
         '<employee personnel-id="30016315" >'
         '<full-name>'
         '<!--this is just a comment-->'
         '<first-name>RICHARD</first>'  /* here the key 'first' is wrong
         '<name>FORDHAM</name>'
         '</full-name>'
         '</employee>'
INTO MYXML LEAVING NO
*
PARSE XML MYXML INTO PATH MYPATH NAME MYNAME VALUE MYVALUE
                     GIVING MYGIVING SUBCODE MYSUBCODE
   WRITE (AL=39) MYPATH
END-PARSE
*
IF MYGIVING NE 0
  WRITE / 'Error Number:'  MYGIVING 'Subcode:' MYSUBCODE
END-IF
END
```

und erzeugt die folgende Ausgabe:

```

employee
employee/@personnel-id
employee/full-name
employee/full-name/!
employee/full-name/first-name

Error Number:        8311 Subcode:         107
```

94 PASSW

- Funktion PASSW 752
- Einschränkung bei PASSW 753
- Syntax-Beschreibung PASSW 753
- Beispiel für PASSW-Statement 754

```
PASSW=operand1
```

Dieser Abschnitt 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 PASSW

Mit dem Statement PASSW geben Sie ein Passwort an, um auf eine passwortgeschützte Adabas- oder VSAM-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.

Unterdrückung der Passwort-Anzeige

Wird das Passwort als Konstante angegeben, sollte das PASSW-Statement ganz am Anfang einer Quellcode-Zeile stehen und das Gleichheitszeichen (=) unmittelbar auf PASSW folgen. Dadurch ist gewährleistet, dass das Passwort im Quellcode des Programms nicht sichtbar ist.

Im TP-Modus:	Im Online-Betrieb können Sie das PASSW-Statement unsichtbar eingeben, indem Sie zuerst das Terminalkommando %* eingeben, bevor Sie das PASSW-Statement eintippen.

Im Batch-Betrieb:	Im Batch-Betrieb kann ein Passwort durch Angabe der folgenden Statements im Zeileneditor angegeben werden:
	
	`EDT`
	
	` PASSW='password'`
	` END`
	
	`.E`
	`RUN`
	
	Der Wert für das Passwort erscheint nicht in der gedruckten Ausgabe.

Einschränkung bei PASSW

Dieses Statement gilt nicht für Db2-Datenbanken.

Syntax-Beschreibung PASSW

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A										ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Passwort:** Das Passwort (*operand1*) 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. 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.

Syntax-Element	Beschreibung
	Ein mit einem bestimmten Datenbank-Statement angegebenes Passwort gilt nur für das jeweilige Statement, nicht für irgendwelche nachfolgenden Statements.

Beispiel für PASSW-Statement

Beispiel-Programm `PWDEX1` mit Passwort-Anzeigeschutz (siehe **oben**):

```
** Example 'PWDEX1': PASSW
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
END-DEFINE
*
PASSW=                                      <== Password not visible
*
LIMIT 5
READ EMPLOY-VIEW
 DISPLAY NOTITLE PERSONNEL-ID NAME
END-READ
*
END
```

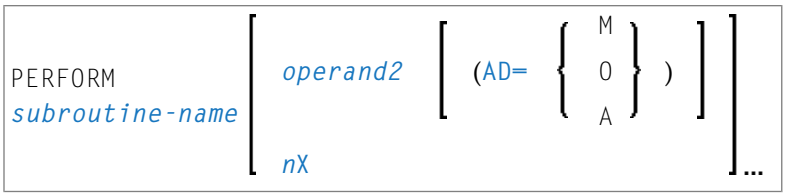
Ausgabe des Programms `PWDEX1`:

```
PERSONNEL         NAME
   ID
--------- --------------------

50005800  ADAM
50005600  MORENO
50005500  BLOND
50005300  MAIZIERE
50004900  CAOUDAL
```

95 PERFORM

- Funktion PERFORM 756
- Syntax-Beschreibung PERFORM 756
- Beispiele PERFORM 759



Dieser Abschnitt 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 Subprogrammen***

Funktion PERFORM

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 PERFORM

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
subroutine-name	**Aufzurufende Subroutine:** Für einen Subroutinen-Namen (maximal 32 Zeichen) gelten dieselben Namenskonventionen wie für Benutzervariablen (siehe *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*). 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). Die aufzurufende Subroutine muss mit einem `DEFINE SUBROUTINE`-Statement definiert werden. Es kann sich dabei um eine interne oder externe Subroutine handeln (siehe `DEFINE SUBROUTINE`-Statement). Innerhalb eines Objekts können nicht mehr als 50 externe Subroutinen referenziert werden. **Von einer Subroutine benutzbare Daten:** ■ **Interne Subroutinen:** Es ist nicht möglich, mit dem `PERFORM`-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 `PERFORM`-Statement Parameter vom aufrufenden Objekt an die aufgerufene Subroutine übergeben (siehe *operand2*); dadurch können Sie die Größe der Global Data Area entsprechend klein halten.
operand2	**Übergabe von Parametern an die externe Subroutine:** Wenn Sie mit dem `PERFORM`-Statement eine externe Subroutine aufrufen, können Sie mit dem `PERFORM`-Statement einen oder mehrere Parameter (*operand2*) vom aufrufenden Objekt an die externe Subroutine übergeben. Für interne Subroutinen kann *operand2* nicht angegeben werden. 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

Syntax-Element	Beschreibung
	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 *operand2* eine Variable ist, können Sie sie folgendermaßen kennzeichnen:

Syntax-Element	Beschreibung	
	AD=O	Nicht modifizierbar, siehe Session-Parameter AD=O. **Anmerkung:** Intern wird AD=O genauso verarbeitet wie BY VALUE (siehe **Anmerkung** unter *operand2*).
	AD=M	Modifizierbar, siehe Session-Parameter AD=M. Dies ist die Standardeinstellung.
	AD=A	Nur für Eingabe, siehe Session-Parameter AD=A.
	Wenn *operand2* eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.	
*n*X	**Angabe zu überspringender Parameter:** Mit der Notation *n*X 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 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 PERFORM

- 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)
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 PEREX2E
  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
```

96 PERFORM BREAK PROCESSING

- Funktion PERFORM BREAK PROCESSING 764
- Syntax-Beschreibung PERFORM BREAK PROCESSING 764
- Beispiel für PERFORM BREAK PROCESSING 765

```
PERFORM BREAK [PROCESSING] [(r)]
  AT BREAK statement ...
```

Dieser Abschnitt 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 PERFORM BREAK PROCESSING

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 PERFORM BREAK PROCESSING

Syntax-Element	Beschreibung
(r)	**Statement-Referenzierung:** Normalerweise wird die PERFORM BREAK PROCESSING-Verarbeitung zum letztenmal ausgeführt, wenn die Ausführung des Programms/Subprogramms bzw. der Subroutine beendet ist. Durch Verwendung eines Statement-Labels oder Angabe der Quellcode-Zeilennummer mittels Notation (r) kann eine bestimmte Verarbeitungsschleife referenziert werden,

Syntax-Element	Beschreibung
	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 BREAK-Verarbeitung wird nach der letzten automatischen Gruppenwechsel-Verarbeitung und vor den AT END OF DATA-Statements ausgeführt.
AT BREAK *statement...*	Siehe Syntax des AT BREAK-Statements.

Beispiel für PERFORM BREAK PROCESSING

```
** 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
```

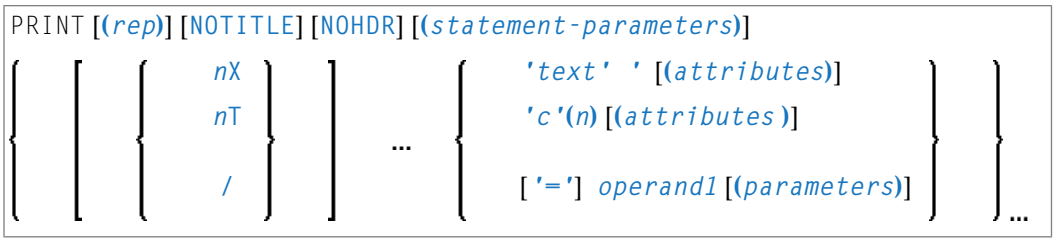
```
_____________________________________________________________________ #LINE:   2
_____________________________________________________________________ #LINE:   3
_____________________________________________________________________ #LINE:   4
_____________________________________________________________________ #LINE:   5
_____________________________________________________________________ #LINE:   6
_____________________________________________________________________ #LINE:   7
_____________________________________________________________________ #LINE:   8
_____________________________________________________________________ #LINE:   9

PLEASE COMPLETE LINES 1-9 ABOVE
```

Äquivalentes Reporting-Mode-Beispiel: PBPEX1R.

97 PRINT

- Funktion PRINT 768
- Syntax-Beschreibung PRINT 769
- Beispiel für PRINT 774



Dieser Abschnitt 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 PRINT

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 PRINT

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(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 `PRINT`-Statement auf den ersten Report (Report 0). 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 *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
`NOTITLE`	**Unterdrückung der Standard-Seitenüberschrift:** 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. Beispiele: ■ Generierte Titelzeile wird ausgegeben:

Syntax-Element	Beschreibung

	`PRINT NAME`
	
	■ Eigene Titelzeile wird ausgegeben:
	
	`PRINT NAME WRITE TITLE 'user-title'`
	
	■ Keine Titelzeile wird ausgegeben:
	
	`PRINT NOTITLE NAME`
	
	Wenn die `NOTITLE`-Option verwendet wird, gilt sie für alle `DISPLAY`-, `PRINT`- und `WRITE`-Statements im selben Objekt, die Daten auf denselben Report schreiben.

`NOHDR`	**Unterdrückung der Spaltenüberschrift:** 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: 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. Ohne `NOHDR`-Option würden auf dieser neuen Seite die `DISPLAY`-Spaltenüberschriften ausgegeben, mit `NOHDR` werden sie dort nicht ausgegeben.
`statement-parameters`	**Parameter-Definition auf Statement-Ebene:** Unmittelbar nach dem Schlüsselwort `PRINT` selbst oder nach einem der auszugebenden Felder können auf Statement-Ebene in Klammern Session-Parameter gesetzt werden. 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 Quellcode-Zeilen erstrecken. 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*.

Syntax-Element	Beschreibung
	Siehe auch: ■ *Liste der Parameter* ■ *Beispiel für Parameter-Benutzung auf Statement- und Element-Ebene.*
*n*X, *n*T, /	Siehe *Feldpositionierung, Text, Attributzuweisung* weiter unten.

Liste der Parameter bei PRINT

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

```
DEFINE DATA LOCAL
1 VARI (A4)     INIT <'1234'>                  /*     Output
END-DEFINE                                     /*    Produced
*                                              /*    ---------
PRINT           'Text'           VARI          /*    Text 1234
PRINT (PM=I)    'Text'           VARI          /*    Text 4321
PRINT           'Text' (PM=I)    VARI (PM=I)   /*    txeT 4321
PRINT           'Text' (PM=I)    VARI          /*    txeT 1234
END
```

Feldpositionierung, Text, Attributzuweisung

```
{ [ nX ]                 { 'text' [(attributes)]                  } }
{ [ nT ] ...             { 'c'(n) [(attributes)]                  } }...
{ [ / ]                  { ['='] operand1 [(parameters)]          } }
```

Feldpositionierungsnotationen

Syntax-Element	Beschreibung
*n*X	**Spaltenabstand:** Mit dieser Notation können Sie zwischen den auszugebenden Werten *n* Leerstellen einfügen. *n* darf nicht 0 sein. Beispiel: `PRINT NAME 5X SALARY`
*n*T	**Setzen von Tabulatoren:** Mit dieser Notation können Sie Tabulatoren setzen, d.h. die Ausgabe eines Wertes beginnt ab Spalte *n*. Wird ein Tabulator gesetzt, dessen Position bereits durch einen anderen ausgegeben Wert besetzt ist, erfolgt ein Zeilenvorschub. Im folgenden Beispiel wird NAME ab Spalte 25 ausgegeben und SALARY ab Spalte 50: `PRINT 25T NAME 50T SALARY`

Syntax-Element	Beschreibung
/	**Zeilenvorschub - Schrägstrich-Notation:** Mit einem Schrägstrich (/) bewirken Sie zwischen zwei Feldern oder Textelementen einen Zeilenvorschub. Beispiel: `PRINT NAME / SALARY`

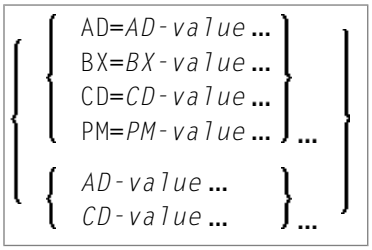
Text-/Attributzuweisung

Syntax-Element	Beschreibung
'*text*'	**Zuweisung von Text:** Eine in Apostrophen angegebene Zeichenkette '*text*' wird als Text ausgegeben. Beispiel: `PRINT 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT`
'*c*' (*n*)	**Wiederholung von Zeichen:** Wie '*text*'. Ausnahme: das Zeichen c wird n-mal unmittelbar vor dem Feldwert ausgegeben. Beispiel: `PRINT '*' (5) '=' NAME`
'='	**Position des Feldinhalts hinter Feldüberschrift:** 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: `PRINT '=' NAME`
operand1	**Auszugebendes Feld:** Als *operand1* geben Sie das auszugebende Feld an.
parameters	**Parameter-Definition auf Elementebene (Feldebene):** Unmittelbar nach *operand1* 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 GLOBALS-Kommando, SET GLOBALS- (nur im Reporting Mode) oder FORMAT-Statement gesetzten Parameter.

Syntax-Element	Beschreibung
	Werden mehrere Parameter angegeben, müssen sie jeweils durch ein oder mehrere Leerzeichen voneinander getrennt werden. Die Angabe eines Parameters darf sich nicht über zwei Quellcode-Zeilen erstrecken. Siehe auch: ■ *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*
- *BX - Box 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 für PRINT

- 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
```

98 PROCESS

- Funktion PROCESS 778
- Einschränkung bei PROCESS 778
- Syntax-Beschreibung PROCESS 779

```
PROCESS  view-name USING {operand1=operand2}, ... [GIVING  operand3...]
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion PROCESS

Das Statement `PROCESS` wird in Verbindung mit Entire System Server und Natural Messaging 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*.

Natural Messaging ist ein Natural-Add-on-Produkt für z/OS, das die Kommunikation mit IBM MQ unterstützt. Es ermöglicht Ihnen den einfachen Zugriff auf Nachrichtenwarteschlangen mithilfe der Statements `PROCESS` und `FIND`. Weitere Informationen siehe *Datenbankmanagementsystem-Schnittstellen* > *Natural Messaging* in der *Natural for z/OS*-Dokumentation.

Einschränkung bei PROCESS

Dieses Statement steht nur bei Entire System Server und Natural Messaging zur Verfügung.

Wenn das Feld `ERROR-CODE` oder `ERROR-TEXT` als Teil der lokalen Daten in die `MQ-QUEUE`-Datensicht (View) aufgenommen wird, werden die standardmäßigen Natural-Systemfehlermeldungen unterdrückt.

Stattdessen enthält `ERROR-CODE` den Fehlercode und `ERROR-TEXT` einen Meldungstext, der die genaue Ursache des Fehlers erläutert.

Dadurch ist das aufrufende Natural-Programm in der Lage, MQ-Fehler programmtechnisch zu behandeln, ohne dass ein Absturz auf Systemebene oder eine Natural-Fehlermeldung ausgelöst wird.

Syntax-Beschreibung PROCESS

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A		N	P			B						ja	nein
operand2	C	S				A	U	N	P			B						ja	nein
operand3		S				A		N	P			B						ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
view-name	Name der von Entire System Server benutzten View (Datensicht).
USING	Mit dieser Klausel können Parameter übergeben werden, indem einem Feld (*operand1*) einer View (Datensicht) ein Wert (*operand2*) zugewiesen wird. **Anmerkung:** Mehrfache Angaben von *operand1=operand2* müssen entweder mit dem Input-Delimiterzeichen (wie mit dem Session-Parameter ID definiert) oder mit einem Komma voneinander getrennt werden.
GIVING	Mit der GIVING-Klausel können Sie Felder (*operand3*) angeben, an die Werte zurückgegeben werden. Jedes dieser Felder muss in einer View (Datensicht) definiert sein.

99 PROCESS COMMAND

- Funktion PROCESS COMMAND 783
- Für die Kommandoverarbeitung verwendetes Datendefinitionsmodul (DDM) 783
- Security-Aspekte 784
- Syntax-Beschreibung PROCESS COMMAND 784
- Beispiele PROCESS COMMAND 796

Structured Mode-Syntax

```
PROCESS COMMAND ACTION
┌─┬─┬──────┬─┐
│ │ │ CHECK│ │
│ │ │ EXEC │ │      USING PROCESSOR-NAME=operand1
│ │ │ TEXT │ │            COMMAND-LINE
│ │ │ HELP │ │            (index[:index])=operand2
│ │ └──────┘ │
│ │          │
│ │ GET USING PROCESSOR-NAME=operand1
│ │           GETSET-FIELD-NAME=operand3
│ │          │
│ │ SET USING PROCESSOR-NAME=operand1
│ │           GETSET-FIELD-NAME=operand3
│ │           GETSET-FIELD-VALUE=operand4
│ │          │
│ │ CLOSE    │
└─┴──────────┘
```

Reporting Mode-Syntax

```
PROCESS COMMAND ACTION
                      USING PROCESSOR-NAME=operand1
┌─┬─┬──────┬─┐            COMMAND-LINE
│ │ │ CHECK│ │            (index[:index])=operand2
│ │ │ EXEC │ │            GIVING RESULT-FIELD
│ │ │ TEXT │ │            (index[:index])
│ │ │ HELP │ │            RETURN-CODE
│ │ └──────┘ │            [NATURAL-ERROR]
│ │          │
│ │ GET USING PROCESSOR-NAME=operand1
│ │           GETSET-FIELD-NAME=operand3
│ │           GIVING GETSET-FIELD-VALUE
│ │           [NATURAL-ERROR]
│ │          │
│ │ SET USING PROCESSOR-NAME=operand1
│ │           GETSET-FIELD-NAME=operand3
│ │           GETSET-FIELD-VALUE=operand4
│ │           [GIVING NATURAL-ERROR]
└─┴──────────┘
```

```
CLOSE [GIVING NATURAL-ERROR]
```

Anmerkung: Die GIVING-Klausel ist nur im Reporting Mode erforderlich und wenn im DEFINE DATA-Statement in der **View-Definition** kein VIEW OF COMMAND definiert worden ist. Die Pflichtfelder der GIVING-Klausel müssen dabei exakt so definiert werden wie die Felder im Datendefinitionsmodul (DDM) COMMAND (**siehe unten**).

Dieses Kapitel behandelt die folgenden Themen:

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

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Subprogrammen*

Funktion PROCESS COMMAND

Sobald ein Kommandoprozessor mit der Utility SYSNCP erstellt worden ist, kann er mit dem Statement PROCESS COMMAND aus einem Natural-Programm heraus aufgerufen werden.

Näheres zur Erstellung eines Natural-Kommandoprozessors finden Sie in der Beschreibung des Dienstprogramms (Utility) SYSNCP in der *Debugger und Dienstprogramme*-Dokumentation.

Für die Kommandoverarbeitung verwendetes Datendefinitionsmodul (DDM)

Wichtig: Das Wort „COMMAND“ im Statement PROCESS COMMAND ist der Name einer View (Datensicht). Der Name der verwendeten View muss nicht notwendigerweise COMMAND sein, aber wir empfehlen die Verwendung von COMMAND, weil ein DDM dieses Namens existiert. Dieses DDM muss im DEFINE DATA-Statement in der **View-Definition** referenziert werden, zum Beispiel: COMMAND VIEW OF COMMAND.

Das Datendefinitionsmodul COMMAND wurde speziell zur Verwendung mit dem PROCESS COMMAND-Statement erstellt:

```
DB:    1 File:    1 - COMMAND                          Default Sequence: ?

  T L   DB  Name                            F Leng  S D   Remark
  - -   --  ------------------------------- - ----  - -   ------------------------
    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 ******
```

Bedeutung der Spaltenüberschriften siehe unter *Spalten für Feld-Attribute* im Kapitel *DDM-Editor benutzen* in der *Editoren*-Dokumentation. Die in dem DDM enthaltenen Felder entsprechen den im `PROCESS COMMAND`-Statement verwendeten DDM-Feldern (Beschreibung siehe ***Im PROCESS COMMAND-Statement verwendete DDM-Felder*** weiter unten).

Anmerkung: Um mögliche Kompilierungs- bzw. Laufzeitfehler zu vermeiden, prüfen Sie bitte, ob das DDM `COMMAND` als Typ `C` (Feld `DDM Type` in dem `SYSDDM`-Menü) katalogisiert ist, bevor Sie es verwenden. (Falls Sie es neu katalogisieren, werden hierbei etwaige Angaben zu Datenbankkennung und Dateinummer (DBID/FNR) in der `SYSDDM`-Utility ignoriert.)

Security-Aspekte

Mit Natural Security können Sie die Verwendung bestimmter in einem Kommandoprozessor definierter Schlüsselwörter und/oder Funktionen einschränken. Schlüsselwörter und/oder Funktionen können für jeden einzelnen Benutzer (oder Gruppen von Benutzern) erlaubt bzw. nicht erlaubt werden.

Weitere Informationen siehe *Funktionssicherheit* in der *Natural Security*-Dokumentation.

Syntax-Beschreibung PROCESS COMMAND

Operanden-Definitionstabelle

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A											nein	nein
operand2	C	S	A	G		A		N									nein	nein
operand3	C	S				A		N									nein	nein
operand4	C	S				A		N	P	I							nein	nein

Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
CHECK	**CHECK-Aktion:** CHECK dient als Vorsichtsmaßnahme, um herauszufinden, ob ein Kommando mit dem Statement PROCESS COMMAND ACTION EXEC ausführbar ist. Für einen bestimmten Prozessor-Namen erfolgt zur Laufzeit eine Prüfung in zwei Schritten: ■ Es wird geprüft, ob der Prozessor in der aktuellen Library oder einer ihrer Steplibs vorhanden ist. ■ Der Inhalt der Kommandozeile COMMAND-LINE (1) wird analysiert, um sicherzustellen, dass er zulässig ist. Außerdem werden die Laufzeit-Aktionsdefinitionen R, M und 1 - 9 in RESULT-FIELD(1:9) geschrieben. Ist das Feld NATURAL-ERROR in der View (Datensicht) oder in der GIVING-Klausel angeben, so wird der Fehlercode in diesem Feld ausgegeben. Ist dieses Feld nicht angegeben und die Kommandoanalyse stößt auf einen Fehler, tritt ein Natural-Systemfehler auf. **Anmerkung:** Da die Funktion der Aktion CHECK auch als Teil der Aktion EXEC ausgeführt wird (siehe unten), ist es nicht nötig, CHECK vor jeder EXEC-Aktion zu verwenden.
EXEC	**EXEC-Aktion:** EXEC funktioniert genau wie CHECK und bewirkt zusätzlich, dass die mit dem **Runtime Action Editor** angegebenen Laufzeit-Aktionen ausgeführt werden. Es wird nur COMMAND-LINE (1) benötigt. Sie können bis zu 9 Ausprägungen von RESULT-FIELD verwenden (im Hinblick auf optimale Verarbeitungszeit sollten Sie jedoch nur soviele Ausprägungen verwenden wie Sie tatsächlich benötigen). **Anmerkung:** EXEC ist die einzige Aktion, mit der das gerade aktive Programm verlassen werden kann. Dies ist der Fall, wenn die Laufzeit-Aktion ein FETCH- oder STOP-Statement enthält. Siehe auch *Beispiel 1 – PROCESS COMMAND ACTION EXEC*.
TEXT	**TEXT-Aktion:** Mit der Aktion TEXT erhalten Sie allgemeine Informationen über den Kommandoprozessor sowie Text zu einem Schlüsselwort bzw. einer Funktion. Der Text ist derselbe, der bei der Definition eines Kommandoprozessors mit der Utility SYSNCP im **Keyword Editor** oder **Action Editor** eingegeben wurde. Weitere Informationen siehe *Eingabewerte für TEXT-Aktionen*.

Syntax-Element	Beschreibung
	Anmerkung: Um auf Text zu Schlüsselwörtern oder Funktionen zugreifen zu können, muss im Feld `Catalog user texts` auf dem Schirm *Processor Header Maintenance 3* der `SYSNCP`-Utility ein `Y` (ja) eingetragen sein, siehe Abschnitt *Verschiedene Optionen - Header 3*.
`HELP`	**HELP-Aktion:** Die Aktion `HELP` liefert 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 `RESULT-FIELD` enthalten. Die Art der erhaltenen Hilfe hängt vom Inhalt der Kommandozeilen ab: ■ `COMMAND-LINE (1)` muss die Suchkriterien enthalten. ■ `COMMAND-LINE (2)` (falls angegeben) muss den Startwert oder einen Suchwert enthalten. ■ `COMMAND-LINE (3)` (falls angegeben) muss einen Startwert enthalten. Weitere Informationen siehe ***Eingabewerte für HELP-Aktionen***. **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.
`GET`	**GET-Aktion** Die Aktion `GET` dient dazu, interne Kommandoprozessor-Informationen und die aktuellen Kommandoprozessor-Einstellungen aus dem dynamisch zugewiesenen `NCPWORK`-Puffer zu lesen. Weitere Informationen siehe ***Eingabewerte für GET- und SET-Aktionen***.
`SET`	**SET-Aktion** Die Aktion `SET` dient zum Ändern interne Einstellungen des Kommandoprozessors im `NCPWORK`-Puffer. Weitere Informationen siehe ***Eingabewerte für GET- und SET-Aktionen***.
`CLOSE`	**CLOSE-Aktion:** Die Aktion `CLOSE` beendet den Einsatz des Kommandoprozessors und gibt den Kommandoprozessor-Puffer wieder frei. Wenn der Kommandoprozessor während einer Session benutzt und nicht mit `CLOSE` freigegeben wird, enthält Ihr Thread einen Puffer namens `NCPWORK`. Die Parameter dieses Puffers können Sie mit dem Systemkommando `BUS` auswerten. Der Puffer wird vom Laufzeit-Teil des Kommandoprozessors benötigt. Er kann mit dem Statement `PROCESS COMMAND ACTION CLOSE` wieder freigegeben werden. Wenn auf dieses Statement ein anderes `PROCESS COMMAND`-Statement folgt, wird der Kommandoprozessor-Puffer erneut geöffnet.

Syntax-Element	Beschreibung
	Siehe auch *Beispiel 2 – PROCESS COMMAND ACTION CLOSE*.
GIVING-Klausel für Reporting Mode	
GIVING	Die GIVING-Klausel ist nur im Reporting Mode erforderlich und wenn im DEFINE DATA-Statement in der **View-Definition** kein VIEW OF COMMAND definiert worden ist. Im Structured Mode ist die GIVING-Klausel nicht vorhanden, 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 die Felder RESULT-FIELD, RETURN-CODE und GETSET-FIELD-VALUE im DEFINE DATA-Statement mit VIEW OF COMMAND definiert sein müssen. Das Feld NATURAL-ERROR kann wahlweise angegeben werden. **Anmerkung:** In der GIVING-Klausel sind Felder angegeben, die vom Kommandoprozessor als Ergebnis der Verarbeitung jeder Aktion ausgefüllt werden müssen.
Im PROCESS COMMAND-Statement verwendete DDM-Felder	
DDM-Feld	**Erläuterung**
PROCESSOR-NAME	Der Name des Kommandoprozessors, für den das PROCESS COMMAND-Statement ausgeführt wird. Der Prozessor muss katalogisiert sein.
COMMAND-LINE	Die Kommandozeile, die vom Kommandoprozessor verarbeitet werden soll (Aktionen CHECK, EXEC) bzw. das Schlüsselwort/Kommando, für das Benutzertext oder Hilfe-Text an das Programm zurückgegeben werden soll (Aktionen TEXT, HELP). Dieses Feld kann aus mehr als einer Zeile bestehen.
RESULT-FIELD	Dieses Feld enthält Informationen, die aus der Verwendung verschiedener Optionen, wie in einer Laufzeit-Aktion angegeben, resultieren. Weitere Informationen siehe *Laufzeit-Aktionen* in der *SYSNCP Utility*-Dokumentation. Achtung, dieses Feld kann aus mehr als einer Zeile bestehen.
GETSET-FIELD-NAME	Dieses Feld wird mit den Aktionen GET und SET verwendet und dient zur Angabe des Namens der Konstanten/Variablen, die gelesen (GET) oder geschrieben (SET) werden soll.
GETSET-FIELD-VALUE	Dieses Feld wird mit den Aktionen GET und SET verwendet und enthält den Wert der im Feld GETSET-FIELD-NAME angegebenen Konstanten/Variablen (siehe oben).
RETURN-CODE	Dieses Feld enthält den Return Code einer Option aufgrund der Aktion EXEC oder CHECK, wie in einer Laufzeit-Aktion angegeben (siehe *SYSNCP-Utility*).
NATURAL-ERROR	Dieses Feld wird mit allen Aktionen verwendet. Wenn es im DEFINE DATA-Statement angegeben wird, enthält es den Natural-Fehlercode für den Kommandoprozessor. Wenn das Feld fehlt, ist die normale Natural-Fehlerbehandlung aktiv, wenn ein Fehler auftritt.

Folgende Themen werden anschließend behandelt:

- Eingabewerte für GET- und SET-Aktionen
- Eingabewerte für TEXT-Aktionen
- Eingabewerte für HELP-Aktionen

Eingabewerte für GET- und SET-Aktionen

Die folgenden Werte können für das Feld `GETSET-FIELD-NAME` (A32) verwendet werden:

Feldname	Format	G/S*	Inhalt
NAME	A8	G	Aktueller Kommandoprozessor-Name.
LIBRARY	A8	G	Geladen aus Library.
FNR	N10	G	Geladen aus Datei.
DBID	N10	G	Geladen aus Datenbank.
TIMESTMP	A8	G	Zeitstempel des aktuellen Kommandoprozessors.
COUNTER	N10	G	Zugriffszähler.
BUFFER-LENGTH	N10	G	Für `NCPWORK` zugeordnete 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.
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.

Feldname	Format	G/S*	Inhalt
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 *Keyword Editor Options - Header 2*
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 Natural-Stack als Daten abgelegtes Kommando.
LAST-ALL	-	G	Letzte oben auf dem Natural-Stack als Daten abgelegte Kommandos.
LAST-COM	-	G	Letztes in die Systemvariable `*COM` gestelltes Kommando.
MULTI	-	G	Legt das letzte von mehreren Kommandos als Daten oben auf dem Natural-Stack ab.
MULTI-COM	-	G	Legt das letzte von mehreren Kommandos in der Systemvariablen `*COM` ab.

*G = Kann mit der Aktion `GET` verwendet werden.

*S = Kann mit der Aktion `SET` verwendet werden.

Eingabewerte für TEXT-Aktionen

Die folgenden Eingabewerte stehen zur Verfügung, um die folgenden Arten von Informationen aus einer TEXT-Aktion zurückzugeben:

- **TEXT für allgemeine Informationen**
- **TEXT für Schlüsselwort-Informationen**
- **TEXT für Funktionsinformationen**

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	Format	Inhalt
1	Text (A40)	Header 1 for User Text (Kopfzeile 1 für Benutzertext)
2	Text (A40)	Header 2 for User Text (Kopfzeile 2 für Benutzertext)
3	Text (A16)	„First Entry used as“ Text (Erster Eintrag benutzt als)
4	Text (A16)	„Second Entry used as“ Text (Zweiter Eintrag benutzt als)
5	Text (A16)	„Third Entry used as“ Text (Dritter Eintrag benutzt als)
6	Numerisch (N3)	Anzahl der Eintrag-1-Schlüsselwörter.
7	Numerisch (N3)	Anzahl der Eintrag-2-Schlüsselwörter.
8	Numerisch (N3)	Anzahl der Eintrag-3-Schlüsselwörter.
9	Numerisch (N7)	Anzahl katalogisierter Funktionen.

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	Format	Inhalt
1	Text (A40)	Schlüsselwort-Kommentartext
2	Text (A16)	Schlüsselwort in voller Länge
3	Text (A16)	Schlüsselwort eindeutig abgekürzt
4	Text (A16)	„Keyword used as“-Eintrag (Schlüsselwort benutzt als)
5	Numerisch (N4)	Interne Schlüsselwortnummer (IKN)
6	Numerisch (N2)	Mindestlänge des Schlüsselworts
7	Numerisch (N2)	Maximallänge des Schlüsselworts
8	Text (A2)	Schlüsselwort-Typ (1, 2, 3, 1S, 2S, 3S, P)

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	Format	Inhalt
1	Text (A40)	Text wie mit Option `T` in Laufzeit-Aktion definiert.
2	Numerisch (N10)	Interne Funktionsnummer (IFN) des angegebenen Platzes.
3	Numerisch (N10)	Interne Funktionsnummer (IFN) der angegebenen Funktion.

Eingabewerte für HELP-Aktionen

Die folgenden Eingabewerte stehen zur Verfügung, um verschiedene Informationen aus einer `HELP`-Aktion zurückzugeben:

- `HELP` **für Schlüsselwörter**
- `HELP` **für Synonyme**
- `HELP` **für globale Funktionen**
- `HELP` **für lokale Funktionen**
- `HELP` **für IKN**
- `HELP` **für IFN**

HELP für Schlüsselwörter

Diese Aktion 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 *Keyword*) 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 Internal Keyword Number (IKN) liefert diese Aktion das originale 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)` wird das originale Schlüsselwort zurückgegeben. In den Feldern `RESULT-FIELD (2:n)` werden die Synonyme des Schlüsselworts zurückgegeben.

Beispiel:

Eingabe:	Ausgabe:
Command Line 1: S Command Line 2: 1003	Result-Field 1: Edit Result-Field 2: Maintain Result-Field 3: Modify

HELP für globale Funktionen

Diese Aktion 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 Command Line 2: ADD	Result-Field 1: ADD CUSTOMER Result-Field 2: ADD FILE Result-Field 3: ADD USER

HELP für lokale Funktionen

Diese Aktion 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 Kommandoprozessors genommen.	
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 Aktion das originale Schlüsselwort.

Kommandozeile	Inhalt	
1	Muss mit `IKN` anfangen.	
	Optionen:	
	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` `Command Line 2:  0000002002`	`Result-Field 1:  CUSTOMER`

HELP für IFN

Für eine bestimmte Interne Funktionsnummer (IFN) liefert diese Aktion die Schlüsselwörter einer Funktion.

Kommandozeile	Inhalt	
1	Muss mit `IFN` anfangen.	
	Option:	
	A	Funktionen mit internen Schlüsselwörtern werden nicht unterdrückt.
2	Die zu übersetzende IFN im Format N10.	
3	**Weitere Optionen:**	
	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

Beispiele PROCESS COMMAND

Nachfolgend finden Sie je ein Anschauungsbeispiel für die Verwendung der Aktion EXEC und der Aktion CLOSE.

- Beispiel 1 - PROCESS COMMAND ACTION EXEC
- Beispiel 2 - PROCESS COMMAND ACTION CLOSE

Beispielprogramme finden Sie in der Library SYSNCP. Die Namen der Beispielprogramme beginnen alle mit EXAM*. Führen Sie das Programm EXAM durch Eingabe des Kommandos EX aus. Es erscheint das Menü **Demonstrate PROCESS COMMAND Statement**, in dem Sie Codes zur Auswahl der einzelnen Aktionsbeispiele eingeben können.

Weitere Informationen finden Sie unter *Online-Tests der Kommandoverarbeitung* im Abschnitt *Merkmale des Kommandoprozessors* in der SYSNCP Utility-Dokumentation.

Beispiel 1 - PROCESS COMMAND ACTION EXEC

```
/* 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
```

Beispiel 2 - PROCESS COMMAND ACTION CLOSE

```
/* 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
```

100 PROCESS PAGE

- Funktion PROCESS PAGE 800
- Syntax 1 - PROCESS PAGE 800
- Syntax 2 - PROCESS PAGE USING 803
- Syntax 3 - PROCESS PAGE UPDATE 806
- Syntax 4 - PROCESS PAGE MODAL 810
- Beispiele 812

Dieser Abschnitt behandelt folgende Themen:

Funktion PROCESS PAGE

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	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A	U											ja	nein
operand2		S	A												C			nein	nein
operand3	C	S				A	U											ja	nein
operand4	C	S	A			A	U	N	P	I	F	B	D	T	L			ja	ja
operand5		S	A												C			nein	nein
operand11		S								I4								ja	ja

Syntax-Element-Beschreibung:

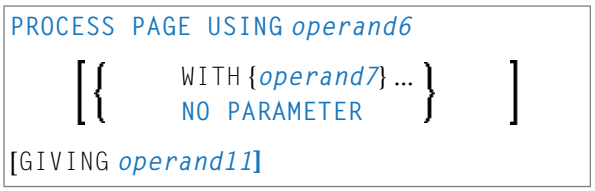
Syntax-Element	Beschreibung	
parameter	**Attribut-Kontrollvariable(n):** Den Parameter CV können Sie in Klammern angeben, um eine oder mehrere der in *operand2* angegebenen Kontrollvariablen zu referenzieren: `(CV=operand2)` Siehe auch *Logische Bedingungen*, *MODIFIED-Option* im *Leitfaden zur Programmierung*.	
operand1	**Name des externen Seiten-Layouts:** *operand1* enthält den Namen des externen Seiten-Layouts.	
operand2	**Name der Attribut-Kontrollvariable(n):** *operand2* enthält den Namen der Kontrollvariablen, muss Format C haben und entweder ein Skalar oder eine einzelne Array-Ausprägung sein.	
operand3	**Namen der externen Datenfelder:** *operand3* enthält den/die Namen des/der externen Datenfelder, in die oder aus denen *operand4* übertragen wird.	
operand4	**Namen der Natural-Datenfelder:** *operand4* enthält den/die Namen des bzw. der externen Natural-Datenfelder, die übertragen werden.	
parameters	**Parameter:** Unmittelbar nach *operand4* können Sie einen oder mehrere Parameter in Klammern angeben:	
	EM oder EMU	Während des Datentransfers verwendete Editiermaske. Weitere Informationen siehe Session-Parameter EM in der *Parameter-Referenz*.

Syntax-Element	Beschreibung	
		Weitere Informationen zu Unicode-Editiermasken siehe Session-Parameter `EMU` in der *Parameter-Referenz*.
	`CV`	Den Parameter `CV`, können Sie angeben, um eine oder mehrere in *`operand5`* angegebene Kontrollvariable zu referenzieren: `(CV=operand5)` Siehe auch *Logische Bedingungen*, *MODIFIED-Option* im *Leitfaden zur Programmierung*.
`operand5`	**Name der Attribut-Kontrollvariable(n):** *`operand5`* enthält den Namen der Kontrollvariablen. Die Variable muss Format C haben. Falls *`operand4`* ein Skalar-Ausdruck oder eine einzelne Array-Ausprägung ist, muss *`operand5`* folgendes sein: ■ ein Skalar-Ausdruck ■ oder eine einzelne Array-Ausprägung. Falls es sich bei *`operand4`* um den vollen Bereich eines Array der Dimension 1 handelt, muss *`operand5`* folgendes sein: ■ ein Skalar-Ausdruck ■ oder eine einzelne Array-Ausprägung ■ oder der volle Bereich eines Array der Dimension 1 mit der gleichen Größe. Falls es sich bei *`operand4`* um den vollen Bereich eines Array der Dimension 2 handelt, muss *`operand5`* folgendes sein: ■ 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 *`operand4`* in Dimension 2 hat.	
`GIVING` *`operand11`*	**GIVING-Klausel:** *`operand11`* enthält den Natural-Fehler, wenn die Anfrage nicht ausgeführt werden konnte.	

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



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	Mögliche Struktur					Mögliche Formate													Referenzierung erlaubt	Dynam. Definition
operand6	C	S				A													ja	nein
operand7		S	A	G		A	U	N	P	I	F	B	D	T	L				ja	ja
operand11		S								I4									ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`USING` *operand6*	**Adapter-Name:** Ruft eine Adapter-Definition auf, die vorher in einer Natural-Systemdatei gespeichert wurde. Siehe auch *Adapter* im *Leitfaden zur Programmierung*. Der Adapter-Name (*operand6*) kann eine 1 bis 8 Zeichen umfassende, alphanumerische Konstante oder Benutzervariable sein. Wenn eine Variable benutzt wird, muss sie vorher definiert worden sein. 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. **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 *Multi Language Management* in der *Natural for Ajax*-Dokumentation.
operand7	**Feld-Spezifikation:** 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. 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

Syntax-Element	Beschreibung
	Inhalt der Datenbank zu ändern, müssen die passenden UPDATE-/STORE-Statements für die Datenbank benutzt werden. Siehe *PROCESS PAGE USING mit im Programm definierten Feldern*.
NO PARAMETER	**NO PARAMETER-Option:** Siehe *PROCESS PAGE USING ohne Parameter-Liste*.
GIVING *operand11*	**GIVING-Klausel:** *operand11* enthält den Natural-Fehler, wenn die Anfrage nicht ausgeführt werden konnte. **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 (*operand11*) übergeben, und die Ausführung wird mit dem nächsten Statement fortgesetzt.

PROCESS PAGE USING ohne Parameter-Liste

Die folgenden Anforderungen müssen erfüllt sein, wenn PROCESS PAGE USING ohne Parameter-Liste benutzt wird:

- Der Adapter-Name (*operand6*) 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 (Datensichten) 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 *Masken-Editor (Map Editor)* in der *Editoren*-Dokumentation.

Das Programmeditorzeilenkommando `.I(adaptername)` kann benutzt werden, um ein vollständiges `PROCESS PAGE USING`-Statement mit einer Parameter-Liste zu erhalten, die aus den im angegebenen Adapter definierten Feldern abstammt.

Wenn das Layout des Adapters geändert wird, braucht das den Adapter verwendende Programm nicht neu katalogisiert zu werden. Wenn aber Feldnamen, Feldformate/längen oder Array-Strukturen im Adapter geändert werden, oder wenn Felder zum Adapter hinzugefügt oder aus ihm gelöscht werden, dann muss das Programm neu katalogisiert werden.

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 ]
    [ DATA ]   [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	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand11	S	I4	ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FULL	**FULL-Option:** Wenn Sie die FULL-Option in einem PROCESS PAGE UPDATE-Statement angeben, wird das entsprechende PROCESS PAGE-Statement vollständig neu ausgeführt: ■ Mit einem normalen PROCESS PAGE UPDATE-Statement (ohne FULL-Option) wird der Inhalt von Variablen, die zwischen dem PROCESS PAGE- und dem PROCESS PAGE

Syntax-Element	Beschreibung
	UPDATE-Statement geändert wurden, nicht angezeigt; das heißt, alle Variablen auf dem Bildschirm zeigen den Inhalt an, den sie hatten, als das PROCESS PAGE-Statement ursprünglich ausgeführt wurde. ■ Mit einem PROCESS PAGE UPDATE FULL-Statement werden alle Änderungen, die nach der anfänglichen Ausführung des PROCESS PAGE-Statements erfolgt sind, auf das PROCESS PAGE-Statement angewandt, wenn es neu ausgeführt wird; das heißt, alle Variablen auf dem Bildschirm enthalten die Werte, die sie hatten, als das PROCESS PAGE UPDATE-Statement ausgeführt wurde. Kennzeichnend für das PROCESS PAGE UPDATE FULL-Statement ist, dass der Status der Attributkontrollvariablen auf NOT MODIFIED zurückgesetzt wird. Dies ist bei einem gewöhnlichen PROCESS PAGE UPDATE-Statement nicht der Fall. Sie können die MODIFIED-Option benutzen, um zu prüfen, ob einer Attributkontrollvariablen der Status MODIFIED zugewiesen worden ist.
DATA	**DATA-Option:** Die DATA-Option verhält sich wie die FULL-Option, jedoch mit der Ausnahme, dass der MODIFIED-Status der Kontrollvariablen *nicht* zurückgesetzt wird.
event-option	**EVENT-Option:** Siehe ***EVENT-Option*** weiter unten.
GIVING (*operand11*)	**GIVING-Klausel:** *operand11* enthält den Natural-Fehler, wenn die Anfrage nicht durchgeführt werden konnte.

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	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand8	C	S				A	U											ja	nein
operand9	C	S				A	U											ja	nein
operand10	C	S	A			A	U	N	P	I	F	B	D	T	L			ja	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`AND SEND EVENT` *operand8*	**Vom externen E/A-System angefordertes Ereignis:** Abhängig von der Implementierung des externen Ein-/Ausgabesystems sind Ereignisse verfügbar, siehe *Sending Events to the User Interface* in der *Natural for Ajax*-Dokumentation.
`WITH PARAMETERS`	**WITH PARAMETERS-Klausel:** Mit dieser Klausel können Sie Folgendes festlegen:
`NAME` *operand9*	**Externer Datenfeld-Name:** *operand9* enthält den externen Namen der Datenfelder, in die/aus denen *operand10* übertragen wird.
`VALUE` *operand10*	**Natural-Datenfelder:** *operand10* enthält die Natural-Datenfelder, die übertragen werden.
`EMU=` `EM=`	**Editiermaske:** Bei der Datenübertragung verwendete Editiermaske. Einzelheiten zu Editiermasken siehe Session-Parameter `EM` in der *Parameter-Referenz*.

Syntax-Element	Beschreibung
	Einzelheiten zu Unicode-Editiermasken siehe Session-Parameter EMU in der *Parameter-Referenz*.
END-PARAMETERS	**Ende der WITH PARAMETERS-Klausel:** Das für Natural reservierte Wort END-PARAMETERS muss benutzt werden, um die WITH PARAMETERS-Klausel zu beenden.

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***.

Verwandtes Statement: PROCESS PAGE

Gehört zur Funktionsgruppe ***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.

Mit der Verarbeitung des PROCESS PAGE MODAL-Statementblocks werden folgende Aktionen ausgeführt:

- Zuerst werden Daten vom Report 0, die noch nicht angezeigt worden sind, angezeigt;
- die Systemvariable *PAGE-LEVEL wird inkrementiert;
- das Öffnen einer modalen Seite wird vorbereitet. Das physische Öffnen der modalen Seite wird mit dem nächsten Statement PROCESS PAGE USING *operand6* WITH ausgeführt, wobei *operand6* der Name des zu verwendenden Adapters ist.

Beim Verlassen des PROCESS PAGE MODAL-Statementblocks werden folgende Aktionen ausgeführt:

- Falls eine modale Seite für diese Ebene geöffnet wurde, wird das Schließen der modalen Seite vorbereitet. Das physische Schließen der modalen Seite erfolgt mit dem nächsten PROCESS PAGE UPDATE [FULL]-Statement;

- 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-Blocks begonnen wurde;
- ein nat:page.default-Ereignis wird in dem Programm ausgelöst, das die modale Seite geöffnet hat.

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.

Das PROCESS PAGE MODAL-Statement ist im Batch-Betrieb nicht gültig.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
statement	**Auszuführende Statements:** 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 IGNORE-Statement angeben.
END-PROCESS	**Ende des PROCESS PAGE MODAL-Statements:** Das für Natural reservierte Wort END-PROCESS muss benutzt werden, um das PROCESS PAGE MODAL-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.

101 PROCESS SQL (SQL)

- Funktion PROCESS SQL 814
- Syntax-Beschreibung PROCESS SQL 814
- Beispiele PROCESS SQL 815

```
PROCESS SQL ddm-name <<statement-string>>
```

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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *PROCESS SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion PROCESS SQL

Das Statement `PROCESS SQL` dient dazu, mit SQL-Statements auf eine zugrundeliegende Datenbank zuzugreifen.

Syntax-Beschreibung PROCESS SQL

Syntax-Element	Beschreibung
ddm-name	**DDM-Name:** Der Name eines Datendefinitionsmoduls (DDM) muss angegeben werden, um die Adresse der Datenbank zu liefern, die die Stored Procedure ausführt. Weitere Informationen siehe *ddm-name*.
statement-string	**Statement-Zeichenkette:** Die Statements, die Sie im *statement-string* angeben können, sind dieselben, die Sie auch mit dem SQL-Statement `EXECUTE` (vgl. ***Flexible SQL***) ausführen können. **Vorsicht:** Um Transaktionssynchronisationsprobleme zwischen der Natural-Umgebung und der zugrundeliegenden Datenbank zu vermeiden, dürfen die Statements `COMMIT` und `ROLLBACK` im `PROCESS SQL`-Statement nicht verwendet werden. Die Statement-Zeichenkette darf über mehrere Zeilen gehen, ohne dass am Zeilenende ein Fortsetzungszeichen erforderlich ist. Sie darf Kommentare am Ende einer Zeile sowie ganze Kommentarzeilen enthalten. In der Statement-Zeichenkette dürfen auch Parameter enthalten sein; siehe ***Parameter in Statement-Zeichenketten*** weiter unten.

Parameter in Statement-Zeichenketten

```
[ :U ]
[ :G ]  :host-variable [INDICATOR:host-variable] [LINIDICATOR:host-variable]
```

Anders als bei den im Abschnitt ***Grundlegende Syntaxbestandteile*** 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

Syntax-Element	Beschreibung
:U:*host-variable*	**USING-Variable:** Das Präfix :U qualifiziert die *host-variable* als sogenannte USING-Variable; d.h. ihr Wert wird an die Datenbank übergeben. :U ist das Standardpräfix.
:G:*host-variable*	**GIVING-Variable:** Das Präfix :G qualifiziert die *host-variable* als sogenannte GIVING-Variable; d.h. sie soll einen Wert von der Datenbank erhalten.

Beispiele PROCESS SQL

Beispiel 1 für Db2:

```
PROCESS SQL DB2_DDM << CONNECT TO :LOCATION >>
```

Beispiel 2 für Db2:

```
PROCESS SQL DB2_DDM << SET :G:LOCATION = CURRENT SERVER >>
```

102 PROPERTY

- Funktion PROPERTY 818
- Syntax-Beschreibung PROPERTY 818
- Beispiel für PROPERTY-Statement 819

```
PROPERTY property-name
OF[INTERFACE] interface-name
IS operand
END-PROPERTY
```

Dieser Abschnitt 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 PROPERTY

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 PROPERTY

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
property-name	Dies ist der der **Property** zugewiesene Name.
`OF` *interface-name*	Dies ist der dem **Interface** zugewiesene Name.
`IS` *operand*	Der *operand* in der `IS`-Klausel weist eine Objektdaten-Variable als Platz zum Speichern des Property-Wertes zu.
`END-PROPERTY`	Das für Natural reservierte Wort `END-PROPERTY` muss zum Beenden des `PROPERTY`-Statements benutzt werden.

Beispiel für PROPERTY-Statement

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.

XI

- 103 READ 823
- 104 READLOB 849
- 105 READ RESULT SET (SQL) 857
- 106 READ WORK FILE 863
- 107 REDEFINE 877
- 108 REDUCE 883
- 109 REINPUT 889
- 110 REJECT 903
- 111 RELEASE 905
- 112 REPEAT 909
- 113 REQUEST DOCUMENT 915
- 114 RESET 933
- 115 RESIZE 939
- 116 RETRY 945
- 117 ROLLBACK (SQL) 949
- 118 RUN 953

103 READ

- Funktion READ 824
- Syntax-Beschreibung READ 825
- Bei READ verfügbare Systemvariablen 839
- Beispiele READ 840

```
{ READ   } [{ ALL        }] [MULTI-FETCH-clause] [RECORDS] [IN] [FILE] view-name
{ BROWSE }  { (operand1) }
[PASSWORD=operand2]
[CIPHER=operand3]
[WITH REPOSITION]
[sequence/range-specification]
[STARTING WITH ISN=operand4]
[[IN] SHARED HOLD [MODE=option]]
[SKIP [RECORDS] IN HOLD]
[WHERE logical-condition]
statement ...
END-READ [(r)]     (structured mode only)
LOOP [(r)]         (reporting mode only)
```

Dieser Abschnitt 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 | READLOB | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Funktion READ

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 folgende Abschnitte im *Leitfaden zur Programmierung*.

- *READ-Statement*
- *Schleifenverarbeitung*
- *Datenbankfelder mit der (r)-Notation referenzieren*

Syntax-Beschreibung READ

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I		B *						ja	nein
operand2	C	S				A												ja	nein
operand3	C	S						N										ja	nein
operand4	C	S						N	P	I		B *						ja	nein

* Format B von *operand1* und *operand4* kann mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

operand1	**Anzahl der zu lesenden Datensätze:**
	
	Sie können die Anzahl der Datensätze, die mit dem READ-Statement gelesen werden sollen, durch Angabe von *operand1* (in Klammern direkt hinter dem Schlüsselwort READ) als numerische Konstante (im Bereich von 0 bis 4294967295) oder als den Namen einer numerischen Benutzervariable begrenzen.
	
	Beispiel:
	
	`READ (5) IN EMPLOYEES ...`
	
	`MOVE 10 TO CNT(N2)`
	`READ (CNT) EMPLOYEES ...`
	
	Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem LIMIT-Statement gesetzten Limit.
	
	Ist mit dem Profil-/Session-Parameter LT ein kleineres Limit gesetzt, so gilt das LT-Limit.
	
	Anmerkung:
	
	1. Wenn Sie eine vierstellige Anzahl von Sätzen lesen möchten, geben Sie diese mit einer vorangestellten Null an: (0*nnnn*); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.

Syntax-Element	Beschreibung
	2. *operand1* wird zu Beginn des ersten READ-Schleifendurchlaufs ausgewertet. Wird der Wert von *operand1* innerhalb der READ-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Datensätze.
ALL	**ALL-Option:** Um hervorzuheben, dass *alle* Datensätze gelesen werden sollen, können Sie optional das Schlüsselwort ALL angeben. Wenn Sie *operand1* und ALL weglassen, wird die ALL-Option standardmäßig verwendet.
MULTI-FETCH-clause	Siehe ***MULTI-FETCH-Klausel***.
view-name	**View-Name:** Als *view-name* geben Sie den Namen einer View (Datensicht) an, der entweder in einem DEFINE DATA-Statement oder in einer programmexternen Global oder Local Data Area definiert ist. Im Reporting Mode ist *view-name* der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.
PASSWORD=*operand2* CIPHER=*operand3*	**PASSWORD- und CIPHER-Klauseln** Diese Klauseln gelten nur für Zugriffe auf Adabas- oder VSAM-Datenbanken. Mit Entire System Server können sie nicht verwendet werden. Die PASSWORD-Klausel dient dazu, ein Passwort anzugeben, um auf Daten einer passwortgeschützten Datei zugreifen zu können. Die CIPHER-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Daten in entschlüsselter Form zu erhalten. Weitere Informationen hierzu siehe Statements FIND und PASSW.
WITH REPOSITION	**WITH REPOSITION-Option:** Diese Option macht das READ-Statement empfänglich für Repositionierungsereignisse. Siehe ***WITH REPOSITION Option***.
sequence/range-specification	**Reihenfolge-/Bereichsangabe:** Diese Option gibt Lese-Reihefolge und -umfang an. Siehe ***Lesereihenfolge und -umfang***.
STARTING WITH ISN=*operand4*	**STARTING WITH ISN-Klausel:** Diese Klausel gilt nur für Adabas- und VSAM-Datenbanken. **Zugriff auf Adabas**

Syntax-Element	Beschreibung
	Diese Klausel kann in Verbindung mit einem READ-Statement in physischer oder logischer Reihenfolge (aufsteigend/absteigend) verwendet werden. Der angegebene Wert (*operand4*) steht für eine Adabas ISN und wird verwendet, um einen bestimmten Datensatz anzugeben, ab dem die READ-Leseschleife gestartet werden soll. **Logische Reihenfolge** 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 im Deskriptorfeld denselben Inhalt haben, werden sie in der Reihenfolge der ISNs sortiert zurückgegeben. Die Klausel STARTING WITH ISN ist so etwas wie ein „Selektionskriterium der zweiten Stufe“, das nur gilt, wenn der Startwert mit dem Deskriptorwert für den ersten Datensatz übereinstimmt. Alle Datensätze mit einem Deskriptorwert, der mit dem Startwert identisch ist, und mit einer ISN, die *kleiner gleich* (*größer gleich* für ein absteigendes READ) der Start-ISN ist, werden von Adabas ignoriert. Der erste in der READ-Schleife zurückgegebene Datensatz ist entweder ■ der erste Datensatz mit Deskriptor = Startwert und einer ISN *größer* (*kleiner* für ein absteigendes READ) als die Start-ISN ■ oder wenn ein solcher Datensatz nicht vorhanden ist, der erste Datensatz mit einem Deskriptor *größer* (*kleiner* für ein absteigendes READ) als der Startwert. **Physische Reihenfolge** Die Datensätze werden in der Reihenfolge zurückgegeben, in der sie physisch gespeichert sind. Wenn eine STARTING WITH ISN-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. **Zugriff auf VSAM** Diese Klausel kann nur in physischer Reihenfolge verwendet werden. Der angegebene Wert (*operand4*) steht für eine VSAM RBA (relative Byte-Adresse von ESDS) oder RRN (relative Datensatznummer von RRDS), die als Startwert für die READ-Leseoperation verwendet werden soll. **Verwendung**

<table>
<tr><th>Syntax-Element</th><th>Beschreibung</th></tr>
<tr><td></td><td>Diese Klausel kann zum Repositionieren innerhalb einer READ-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.

Die Klause 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.

Beispiel: Siehe Programm REASISND weiter unten.</td></tr>
<tr><td>[[IN] SHARED HOLD [MODE=option]]</td><td>SHARED HOLD-Klausel:

Anmerkung: Diese Klausel gilt nur für Zugriffe auf Adabas Version 8.2 oder höher.

Mit dieser Option können Sie die zurzeit gelesenen Datensätze in einen „Shared Hold“-Status setzen. Ein Datensatz kann durch viele Benutzer gleichzeitig in den „Shared Hold“-Status gesetzt werden. Solange sich ein Datensatz im „Shared Hold“-Status befindet, ist er vor Änderungen geschützt, weil er nicht durch zeitgleich arbeitende Benutzer in einen „Exclusive Hold“-Status gesetzt werden kann. Dadurch wird die Datenkonsistenz für die Daten des Datensatzes sichergestellt, weil niemand den Datensatz ändern kann, während er sich in Verarbeitung befindet.

Insbesondere wenn auf denselben Datensatz mit mehreren Statements zugegriffen wird, um verschiedene MU-/PE-Ausprägungen zu lesen (GET SAME-Statement) oder um ein LOB-Feld segmentweise zu lesen (READLOB-Statement), kann der „Shared Hold“-Status die Stabilität der Daten während dieser Transaktion garantieren, ohne den Datensatz für andere Benutzer zu blockieren.

Obwohl ein solcher Hold-Status eine effiziente Art ist, Leseabfolgen zu schützen, ist es dennoch eine grundlegende und wichtige Angelegenheit, den Datensatz aus dieser sanften Sperre wieder freizugeben. Da dies von individuellen Aspekten der Anwendung abhängt, besteht die Möglichkeit, mit Hilfe der MODE-Subklausel verschiedene Optionen auszuwählen.
<table>
<tr><th>MODE-Option</th><th>Sperrzeitraum</th><th>Erläuterung</th></tr>
<tr><td>C</td><td>Nur zum Zeitpunkt des Lesens des Datensatzes;</td><td>Stellt sicher, dass die zurzeit gelesene Version des Datensatzes durch den letzten Änderer festgeschrieben worden</td></tr>
</table>
</td></tr>
</table>

Syntax-Element	Beschreibung		
			ist. Diese Option setzt nicht wirklich eine Sperre, sondern prüft nur, ob der Datensatz zum Zeitpunkt des Lesens nicht durch einen anderen Benutzer in einem „Exclusive Hold“-Status gehalten wird.
	Q	solange bis der nächste Datensatz in einer Abfolge gelesen ist;	Gibt den Datensatz aus dem „Shared Hold“-Status frei, wenn ■ der nächste Datensatz in der Schleifenabfolge gelesen wird oder ■ die Schleife beendet wird oder ■ ein `END TRANSACTION`- oder `BACKOUT TRANSACTION`-Statement ausgeführt wird.
	S	bis die logische Transaktion beendet ist.	Gibt den Datensatz aus dem „Shared Hold“-Status frei, wenn eine logische Transaktion mit einem `END TRANSACTION`- oder `BACKOUT TRANSACTION`-Statement beendet wird.
	`MODE=Q` und `MODE=S` stellen sicher, dass der zurzeit gelesene Datensatz solange nicht gleichzeitig durch andere Benutzer geändert werden kann, bis er wieder aus dem Hold-Status freigegeben worden ist. Falls die `MODE`-Subklausel weggelassen wird, gilt `MODE=C` als Standardwert. Siehe auch ***Beispiel 8 - SHARED HOLD-Klausel*** weiter unten		
`SKIP RECORD(S) IN HOLD`	**SKIP RECORDS IN HOLD-Klausel:** **Anmerkung:** Diese Klausel kann nur für Zugriffe auf Adabas benutzt werden.		

Syntax-Element	Beschreibung
	Immer wenn ein im Hold befindlicher Datensatz gelesen werden soll, kann ein Natural-Fehler NAT3145 (Adabas-Rückmeldeschlüssel `145`) auftreten, falls der Datensatz zu diesem Zeitpunkt durch einen anderen Benutzer in den Hold-Status versetzt worden ist. Das geschieht, wenn ein „Shared Hold“-Status angefordert wird und der Datensatz sich im „Exclusive Hold“-Status befindet, oder wenn ein „Exclusive Hold“-Status angefordert wird und der Datensatz sich entweder im „Exclusive Hold“- oder im „Shared Hold“-Status befindet. Gewiss ist der Natural-Fehler NAT3145 die angemessene Reaktion, um eine einwandfreie Datenverarbeitung sicherzustellen, jedoch kann es gelegentlich von Nutzen sein, dass ein im Hold befindlicher Datensatz übersprungen werden kann. Wenn es in Ordnung ist, dass ein solcher Datensatz nicht bearbeitet und die Schleifenverarbeitung fortgesetzt wird, sollte die `SKIP RECORDS`-Klausel verwendet werden. Wenn die `SKIP RECORDS`-Klausel angewendet wird, versucht Natural zunächst, den Datensatz mit Hold zu lesen. Wenn die Fehlersituation für einen Natural-Fehler NAT3145 eintrifft, dann ■ wird keine Fehlerverarbeitung eingeleitet; ■ der (zurzeit durch einen anderen Benutzer gesperrte) Datensatz wird sofort ohne Hold erneut abgerufen, jedoch nicht im Sinne der Programmlogik verarbeitet; ■ der Datensatz, der nach dem übersprungenen Datensatz an der Reihe ist, wird ohne Hold gelesen und die Verarbeitung wird fortgesetzt. Die `SKIP RECORD IN HOLD`-Funktionalität greift nur, wenn der Natural-Profilparameter `WH` auf `OFF` gesetzt ist. Siehe auch ***Beispiel 9 - SKIP RECORDS-Klausel*** weiter unten
`WHERE` *`logical-condition`*	**WHERE-Klausel:** 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, und *bevor* eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich `AT BREAK`-Verarbeitung) erfolgt. Näheres zu logischen Bedingungen und die Syntax finden Sie unter *Logische Bedingungen* im *Leitfaden zur Programmierung*.

Syntax-Element	Beschreibung
	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.
END-READ (*r*) LOOP (*r*)	**Ende des READ-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-READ zum Beenden des READ-Statements benutzt werden. Im Structured Mode können Sie bei END-READ Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des READ-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

Weitere Syntax-Element-Beschreibungen:

- MULTI-FETCH-Klausel
- WITH REPOSITION-Option
- Lesereihenfolge und -umfang (sequence/range-specification)

MULTI-FETCH-Klausel

Anmerkung: Diese Klausel kann nur bei Adabas- oder Db2-Datenbanken benutzt werden.

```
[ MULTI-FETCH { ON                    } ]
              { OFF                   }
              { [OF] multi-fetch-factor }
```

Ausführliche Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung* oder *Verarbeitung mehrerer Zeilen* (SQL) im *Natural for Db2*-Teil der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

WITH REPOSITION-Option



Anmerkung: Diese Option ist nur beim Zugriff auf Adabas- oder VSAM-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.

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)

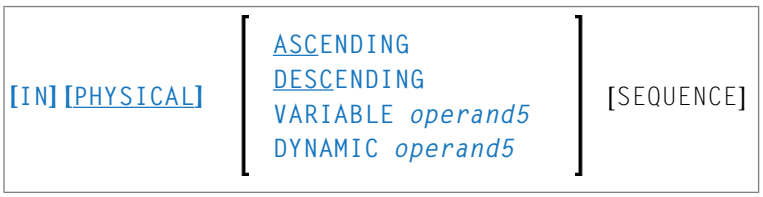
Die folgenden Syntax-Optionen sind verfügbar, um Lese-Reihenfolge und/oder -Bereich anzugeben.

- Syntax-Option 1: READ PHYSICAL
- Syntax-Option 2: READ BY ISN
- Syntax-Option 3: READ BY DESCRIPTOR
- Operanden-Definitionstabelle (sequence/range-specification)
- Syntax-Element-Beschreibung (sequence/range-specification)
- Anmerkungen zu Funktionsunterschieden zwischen THRU/ENDING AT und TO bei Verwendung in READ BY DESCRIPTOR

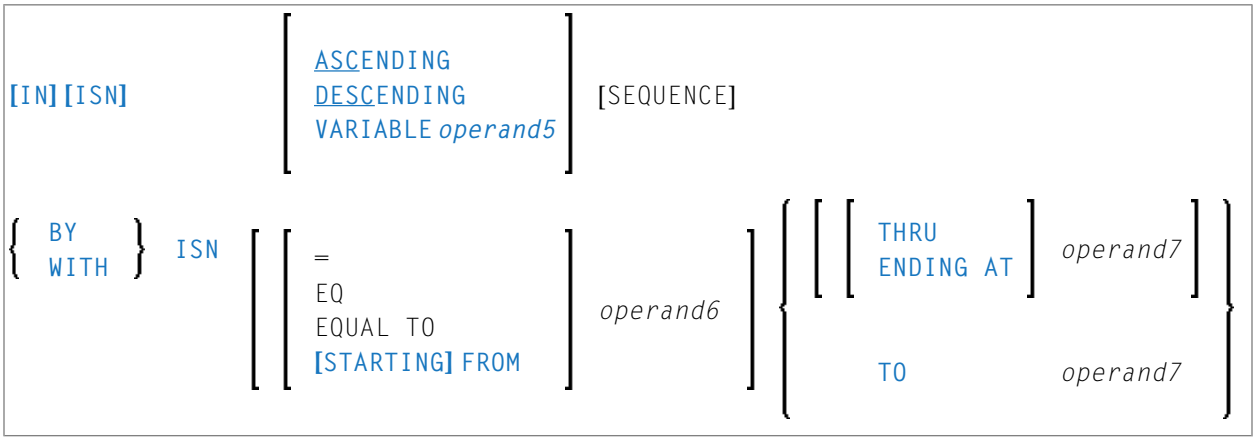
Anmerkungen:

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.

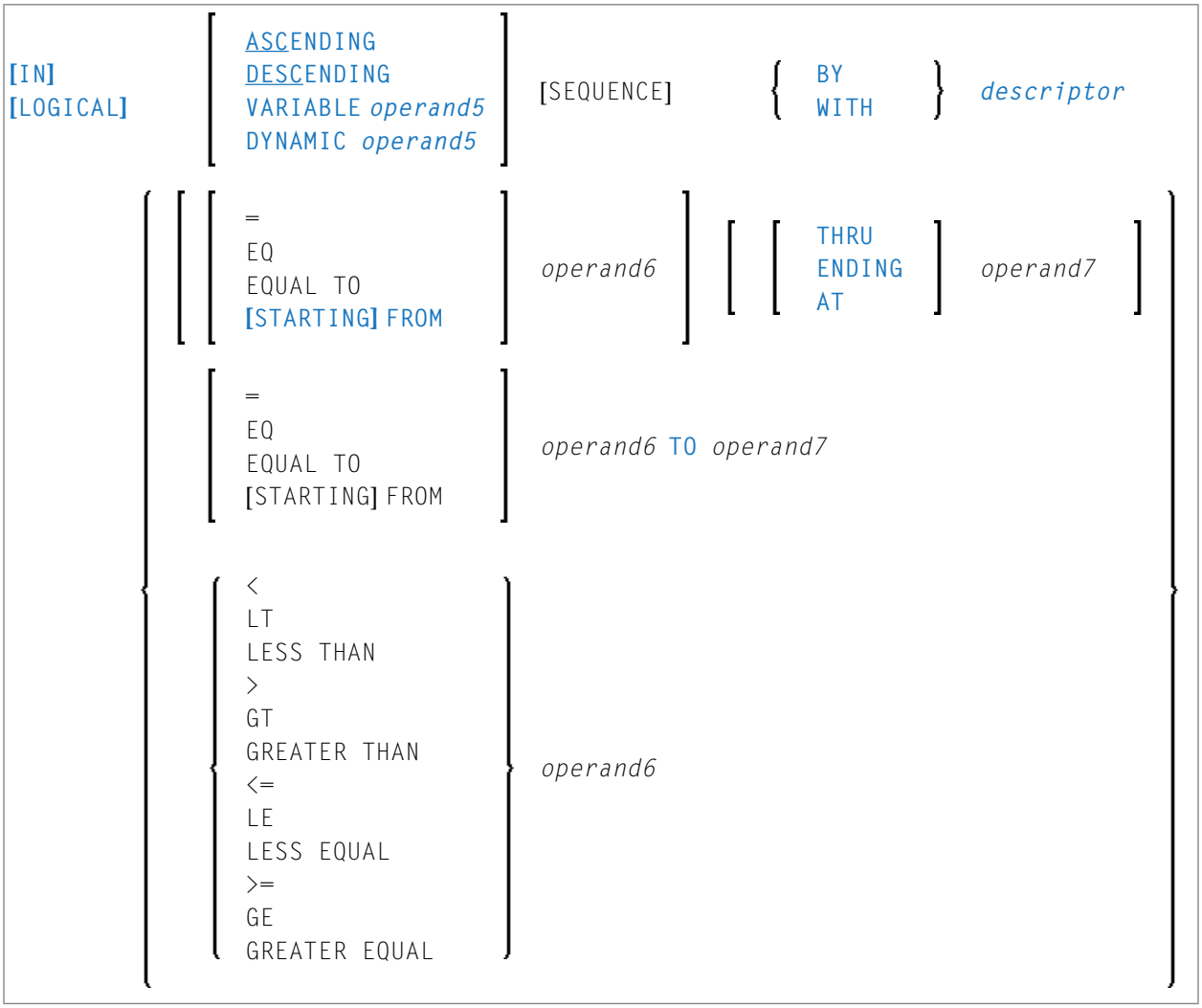
Syntax-Option 1: READ PHYSICAL



Syntax-Option 2: READ BY ISN



Syntax-Option 3: READ BY DESCRIPTOR



Operanden-Definitionstabelle (sequence/range-specification)

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand5		S				A												ja	nein
operand6	C	S				A		N	P	I	F	B *	D	T	L			ja	nein
operand7	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 (sequence/range-specification)

Syntax-Element	Beschreibung
`READ IN PHYSICAL SEQUENCE`	**Lesen in physischer Reihenfolge:** `PHYSICAL SEQUENCE` bedeutet, dass die Datensätze in der physischen Reihenfolge, in der sie auf der Datenbank gespeichert sind, gelesen werden. **Anmerkung:** Bei VSAM-Datenbanken ist `READ PHYSICAL` nur beim Zugriff auf ESDS und RRDS möglich. Ist nichts anderes angegeben, gilt standardmäßig `IN PHYSICAL SEQUENCE`.
`READ BY ISN`	**Lesen nach ISN:** `READ BY ISN` bedeutet, dass die Datensätze in der Reihenfolge der Adabas-ISNs (Interne Satznummern) bzw. VSAM-RBAs (relative Byte-Adressen von ESDS) bzw. -RRNs (relative Datensatznummern von RRDS) gelesen werden. Anstelle des Schlüsselworts `BY` können Sie, mit gleichem Ergebnis, auch das Schlüsselwort `WITH` verwenden. Für `READ BY ISN` müssen die Operanden *operand6* und *operand7* als numerische Konstante oder Benutzervariable im Bereich von `0` bis `4294967295` zur Verfügung gestellt werden. Für Extended ESDS VSAM-Dateien kann dieser Wert bei einer Steuerintervallgröße von 4 KB bis zu 16 TB und bei einer Steuerintervallgröße von 32 KB bis zu 128 TB betragen. `READ BY ISN` ist nur bei Adabas- und VSAM-Datenbanken möglich, und zwar bei VSAM nur für ESDS und RRDS. **Anmerkung:** Bei XML-Datenbanken (z.B. Tamino) dient `READ BY ISN` zum Lesen von XML-Objekten entsprechend der Reihenfolge der Objekt-IDs.
`READ IN LOGICAL SEQUENCE`	**Lesen in logischer Reihenfolge:** `LOGICAL SEQUENCE` bedeutet, dass die Datensätze in der Reihenfolge der Werte eines bestimmten Deskriptorfeldes (Key) gelesen werden. 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. Wenn Sie kein Deskriptorfeld angeben, wird der im verwendeten DDM eingetragene Standarddeskriptor (Feld `Default Sequence`) genommen. **Anmerkung:** 1. Bei VSAM-Datenbanken: `LOGICAL` ist nur möglich für KSDS mit definierten Primär- und Alternativschlüsseln bzw. ESDS mit Alternativschlüsseln. Wird ein Deskriptorfeld verwendet, das mit Nullwertunterdrückung definiert ist (gilt nur für Adabas), werden Datensätze, bei denen das Deskriptorfeld einen Nullwert enthält, nicht gelesen. Wird als

Syntax-Element	Beschreibung
	Deskriptorfeld ein multiples Feld verwendet (gilt nur für Adabas), wird ein einzelner Datensatz mehrmals gelesen, wenn das Feld mehrere Werte enthält. Informationen zu `READ IN LOGICAL SEQUENCE` finden Sie auch im *Leitfaden zur Programmierung*, siehe *Statements für Datenbankzugriffe*, *READ-Statement*.
`ASCENDING | DESCENDING | VARIABLE | DYNAMIC SEQUENCE`	**Lesen in aufsteigender/absteigender Reihenfolge:** Diese Klausel gilt nur für Adabas-, XML, VSAM- und SQL-Datenbanken. Bei einem `READ PHYSICAL`-Statement gilt sie nur für VSAM- und Db2-Datenbanken. In einem `READ BY ISN`-Statement kann diese Klausel nur bei Adabas Version 8.6 (oder höher) angewendet werden Mit dieser Klausel können Sie bestimmen, ob die Datensätze in aufsteigender oder absteigender Reihenfolge gelesen werden sollen. ■ Standardmäßig werden die Datensätze in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort `ASCENDING` auch ausdrücklich angeben können, aber nicht müssen). ■ Wenn die Datensätze in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort `DESCENDING` an. ■ Wenn erst zur Laufzeit bestimmt werden soll, ob die Datensätze in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort `VARIABLE` oder `DYNAMIC` gefolgt von einer Variablen (*operand5*) an. Der Wert von *operand5* zu Beginn der `READ`-Verarbeitungsschleife bestimmt dann die Reihenfolge. *operand5* muss Format/Länge A1 haben und kann den Wert `A` (für Ascending/aufsteigend) oder `D` (für Descending/absteigend) enthalten. ■ Wenn das Schlüsselwort `VARIABLE` benutzt wird, wird die Leserichtung (Wert von *operand5*) am Anfang der `READ`-Verarbeitungsschleife ausgewertet, und sie bleibt bestehen, bis die Schleife beendet wird, ganz gleich ob das Feld von *operand5* in der `READ`-Schleife geändert wird oder nicht. ■ Wenn das Schlüsselwort `DYNAMIC` benutzt wird, wird die Leserichtung (Wert von *operand5*) vor jedem Aufruf eines Datensatzes in der `READ`-Verarbeitungsschleife ausgewertet und kann von Datensatz zu Datensatz geändert werden. Dies ermöglicht es überall in der `READ`-Schleife, die Reihenfolge beim Durchblättern von aufsteigend in absteigend (und umgekehrt) zu ändern. Die Verwendung dieser Option ist in einem `READ BY ISN`-Statement nicht gestattet.
`STARTING FROM ... ENDING AT/TO`	**STARTING FROM/ENDING AT-Klausel:** Mit den Klauseln `STARTING FROM` und `ENDING AT` können Sie angeben, ab welchem Wert und bis zu welchem Wert des Deskriptorfeldes gelesen werden soll. Die `STARTING FROM`-Klausel (= oder `EQ` oder `EQUAL TO` oder `[STARTING] FROM`) legt den Startwert für die `READ`-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 `READ`) Wert benutzt. Wenn

Syntax-Element	Beschreibung
	kein höherer (oder niedrigerer für absteigendes READ) Wert vorhanden ist, wird die Schleife nicht durchlaufen. Um die Datensätze auf einen Endwert zu begrenzen, können Sie eine ENDING AT-Klausel mit den Bedingungen THRU, ENDING AT oder TO angeben, wobei dann bis einschließlich der angegebenen Werte gelesen wird. Immer wenn das READ-Deskriptorfeld den angegebenen Endwert überschreitet, wird die Schleife automatisch beendet. Obwohl die Basis-Funktionalität der Schlüsselwörter TO, THRU und ENDING AT sich jeweils ähnelt, so unterscheiden sie sich doch im Detail.
THRU/ENDING AT	**THRU/ENDING AT-Option:** Wenn THRU oder ENDING AT 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 ASCENDING (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 READ-Schleife zurückgegebenen Wert (und den Datensatz) darstellt. Wenn Sie aber einen rückwärtsgerichteten (DESCENDING) Lesevorgang starten, muss der höhere Wert im Startwert und der niedrigere Wert im Endwert erscheinen. **Bei Verwendung in READ BY DESCRIPTOR** 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). **Bei Verwendung in READ BY ISN** Bei Vorhandensein von „ISN verfügbar als Endwert“ verlässt Natural die READ-Schleife, sobald diese ISN-Nummer erreicht wurde. In diesem Fall ist der letzte Datensatz in der Datensicht der Endwert-Datensatz. Bei Nichtvorhandensein von „ISN verfügbar als Endwert“ liest Natural einen Datensatz nach dem Endwert. Dies ist bei aufsteigender (ASCENDING) Lesefolge die nächst höhere ISN bzw. bei absteigender (DESCENDING) Lesefolge die nächst niedrigere ISN (außer wenn die Datei keinen weiteren Datensatz nach dem Datensatz des letzten Ergebnisses enthält). THRU / ENDING AT kann für alle Datenbanken benutzt werden, die die READ- oder HISTOGRAM-Statements unterstützen.
TO	**TO-Option:** 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. Im

Syntax-Element	Beschreibung
	Falle eines Endwertstopps ist der Datensatz in der Datensicht der letzte gefundene Datensatz im angegebenen Wertebereich. **Bei Verwendung in READ BY DESCRIPTOR** Unabhängig davon, ob in aufsteigender (ASCENDING) Lesefolge oder absteigender (DESCENDING) Lesefolge gelesen wird, ist der niedrigere Wert in den Startwert und der höhere Wert in den Endwert zu setzen. **Bei Verwendung von READ BY ISN** Wie bei THRU, setzen Sie die ISN des Datensatzes, den Sie zuerst erhalten möchten, in den Startwert und die ISN des Datensatzes, den Sie zuletzt erhalten möchten, in den Endwert ein. In einem READ BY ISN-Statement kann die TO-Option nur verwendet werden, wenn es gegen Adabas for Mainframe 8.6 (oder höher) ausgeführt wird.

Anmerkungen zu Funktionsunterschieden zwischen THRU/ENDING AT und TO bei Verwendung in READ BY DESCRIPTOR

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 die View (Datensicht) den ersten Datensatz, der außerhalb des Bereiches ist (Out-of-Range).	Wenn die READ-Schleife beendet wird, weil der Endwert erreicht worden ist, enthält die 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.
Um auf einen Bereichsüberlauf hin abzuprüfen, muss der Deskriptorwert in der zugrunde liegenden	Der Deskriptor ist für die zurückgegebenen Satzfelder nicht erforderlich.

THRU/ENDING AT	TO
Datenbank-View erscheinen, d.h. er muss im Satzpuffer zurückgegeben werden.	
Endwerte-Prüfungen auf Adabas-Mehrwertfelder (MU-Feld) oder Sub-/Super-/Hyperdeskritporen 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 7 (oder höher), Db2, oder VSAM benutzt werden.



Anmerkung: Das Ergebnis von READ/HISTOGRAM THRU/ENDING AT kann vom Ergebnis bei READ/HISTOGRAM TO abweichen, wenn Natural und die Datenbank, auf die zugegriffen wird, sich auf verschiedenen Plattformen mit unterschiedlichen Sortierfolgen befinden.

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:

Systemvariable	Erläuterung
*ISN	Die Systemvariable *ISN enthält die Adabas-ISN des gerade verarbeiteten Datensatzes. **Anmerkung:** 1. Bei VSAM-Datenbanken enthält *ISN entweder die RRN (für RRDS) oder die RBA (für ESDS) das aktuellen Datensatzes. 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 READ

- 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 8 - SHARED HOLD-Klausel
- Beispiel 9 - SKIP RECORDS-Klausel
- Beispiel 10 - READ DESCENDING BY ISN

Beispiel 1 - READ-Statement

```
** Example 'READEX1S': 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
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 `READEX1S`:

```
PERSONNEL         NAME            ISN         CNT
   ID
--------- -------------------- ----------- -----------

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: `READEX1R`.

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=O) '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=O) '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      FIRST-NAME         MAKE
      ID
--------------- --------------- ---------------

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=O) 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=O)
         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
    IF *PF-KEY = 'PF3'
      STOP
    END-IF
  END-READ
  /*
  IF *COUNTER(0290) = 0
    STOP
  END-IF
END-REPEAT
END
```

Beispiel 8 - SHARED HOLD-Klausel

```
READ EMPL-VIEW WITH NAME = ...
     IN SHARED HOLD MODE=Q                /* Record in shared hold until next ↩
record is read.
   ...
   GET EMPL-VIEW  *ISN                    /* The record remains unchanged!
   ...
END-READ
```

Beispiel 9 - SKIP RECORDS-Klausel

```
READ EMPL-VIEW WITH NAME = ...              /* Records found are put in hold while ↩
reading.
     SKIP RECORDS IN HOLD                   /* Records already held by other users ↩
are skipped
  ...                                        /* to prevent error NAT3145.
  UPDATE
  END TRANSACTION
END-READ
```

Beispiel 10 - READ DESCENDING BY ISN

```
** Example 'READVISN': READ with DESCENDING/VARIABLE BY ISN
**
** Note: The ASCENDING and DESCENDING order for READ BY ISN can only be
**       applied to ADABAS 8.6 (or higher) on mainframes.
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
*
1 #DIR-ASCENDING  (A1) CONST<'A'>
1 #DIR-DESCENDING (A1) CONST<'D'>
*
END-DEFINE
*
WRITE 'READ with ASCENDING BY ISN = 500'
READ (5) EMPLOY-VIEW ASCENDING BY ISN = 500
  DISPLAY NOTITLE *ISN NAME FIRST-NAME
END-READ
*
WRITE / 'READ with DESCENDING BY ISN = 500'
READ (5) EMPLOY-VIEW DESCENDING BY ISN = 500
  DISPLAY         *ISN NAME FIRST-NAME
END-READ
*
WRITE / 'READ with VARIABLE ascending BY ISN = 500'
READ (5) EMPLOY-VIEW VARIABLE #DIR-ASCENDING BY ISN = 500
  DISPLAY         *ISN NAME FIRST-NAME
END-READ
*
WRITE / 'READ with VARIABLE descending BY ISN = 500'
READ (5) EMPLOY-VIEW VARIABLE #DIR-DESCENDING BY ISN = 500
  DISPLAY         *ISN NAME FIRST-NAME
END-READ
END                                                                    *        ↩
```

Ausgabe des Programms `READVISN`:

```
   ISN              NAME              FIRST-NAME
----------- -------------------- --------------------

READ with ASCENDING BY ISN = 500
        500 JENSEN               HANS
        501 JENSEN               CLAUS
        502 SOERENSEN            KARL
        503 ERIKSSEN             JONAS
        504 ANDERSEN             ANITA

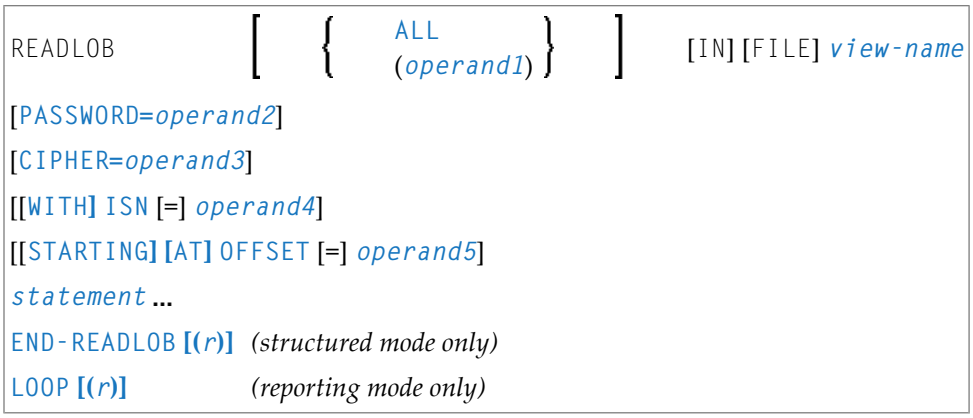
READ with DESCENDING BY ISN = 500
        500 JENSEN               HANS
        499 MARTINEZ             MANUEL
        498 OSEA                 ROBERTO
        497 FERNANDEZ            CARMEN
        496 DE LA IGLESIA        JORGE

READ with VARIABLE ascending BY ISN = 500
        500 JENSEN               HANS
        501 JENSEN               CLAUS
        502 SOERENSEN            KARL
        503 ERIKSSEN             JONAS
        504 ANDERSEN             ANITA

READ with VARIABLE descending BY ISN = 500
        500 JENSEN               HANS
        499 MARTINEZ             MANUEL
        498 OSEA                 ROBERTO
        497 FERNANDEZ            CARMEN
        496 DE LA IGLESIA        JORGE
```

104 READLOB

- Funktion READLOB 850
- Einschränkungen bei READLOB 850
- Syntax-Beschreibung READLOB 851
- Bei READLOB verfügbare Systemvariablen 853
- Funktionstechnische Überlegungen bei READLOB 854
- Beispiele READLOB 855



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

Verwandte Statements: READ | FIND | GET

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Funktion READLOB

Das Statement READLOB wird bei einem einzelnen Datensatz verwendet, bei dem das definierte LOB-Feld (LOB = Large OBject) während der Schleifenausführung in festgelegten Segmenten gelesen wird.

Zu Beginn der Schleife wird der Versatz (Offset) innerhalb des LOB-Feldes gesetzt, ab dem die ersten Daten gelesen werden sollen. Bei der nächsten Schleifen-Iteration wird das Segment zurückgegeben, das auf das zuletzt gelesene Segment folgt. Wenn das Ende der LOB-Daten erreicht ist, wird die Schleife beendet.

Dieses Statement leitet eine Verarbeitungsschleife ein. Siehe auch *Schleifenverarbeitung* im *Leitfaden zur Programmierung*.

Einschränkungen bei READLOB

Das Statement READLOB kann nur für den Zugriff auf Adabas-Datenbanken benutzt werden.

Syntax-Beschreibung READLOB

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I	B *							ja	nein
operand2	C	S				A												ja	nein
operand3	C	S						N										ja	nein
operand4	C	S						N	P	I	B *							ja	nein
operand5	C	S						N	P	I	B *							ja	nein

* Format B von *operand1*, *operand4* und *operand5* kann mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

operand1	**Anzahl der zu lesen LOB-Segmente:**
	
	Die Anzahl der durchzuführenden Schleifenverarbeitungen kann durch Angabe von *operand1* (in Klammern, unmittelbar nach dem Schlüsselwort READLOB eingeschränkt werden., und zwar entweder als numerische Konstante (0 - 4294967295) oder als Name einer numerischen Variable.
	
	Beispiel:
	
	`READLOB (5) IN FILE VIEW01 ...`
	
	`#CNT := 10`
	`READLOB (#CNT) IN FILE VIEW01 ...`
	
	Für dieses Statement hat das hier angegebene Limit Vorrang vor einem Limit, welches mit dem LIMIT-Statement gesetzt wurde. Falls mit dem Profil- oder Session-Parameter LT ein kleineres Limit gesetzt ist, gilt das LT-Limit.
	
	Anmerkung: Die Auswertung von *operand1* erfolgt beim Start des READLOB-Statements. Wird der Wert von *operand1* während der READLOB-Schleife geändert, hat dies keine Auswirkung auf die Anzahl der Schleifenausführungen.

ALL	**ALL-Option:** Sie können wahlweise das Schlüsselwort ALL angeben, um zu verdeutlichen, dass die Daten des LOB-Feldes bis zum Ende gelesen werden sollen.

Syntax-Element	Beschreibung
	Wenn Sie *operand1* und ALL weglassen, wird die ALL-Option standardmäßig verwendet.
view-name	**View-Name:** Als *view-name* geben Sie den Namen einer View (Datensicht) an, die entweder mit einem DEFINE DATA-Statement oder außerhalb des Programms in einer Global oder Local Data Area definiert worden sein muss. ■ Die View darf nur ein LOB-Feld mit einem einzigen Wert enthalten, zusätzliche Felder sind nicht zulässig. ■ Falls das LOB-Feld ein MU- oder PE-Feld ist, muss eine einzige Ausprägung angegeben sein; es ist keine Bereichsnotation zulässig. ■ Das LOB-Feld muss in dem DDM mit einer festen (nicht dynamischen) Länge definiert sein, welche die Segmentlänge des LOB-Feldes darstellt..
PASSWORD=*operand2* CIPHER=*operand3*	**PASSWORD- und CIPHER-Klauseln:** Die PASSWORD-Klausel dient zur Angabe eines Passwortes, wenn Daten aus einer passwortgeschützten Datei gelesen werden sollen. Die CIPHER-Klausel dient zur Angabe eines Chiffrierschlüssels, wenn Daten aus einer verschlüsselten Datei gelesen werden sollen. Weitere Informationen siehe Statements FIND und PASSW.
WITH ISN=*operand4*	**WITH ISN-Option:** Diese Option dient zur Angabe der internen Folgenummer (ISN) des Datensatzes, auf den das READLOB-Statement zugreifen soll. Während der gesamten Schleifenausführung wird nur dieser Datensatz gelesen. *operand4* muss entweder in Form einer numerischen Konstante oder als eine benutzerdefinierte Variable oder über die Natural-Systemvariable *ISN angegeben werden. Das Feld wird durch die Ausführung des READLOB-Statements nicht verändert. **Anmerkung:** Die Auswertung von *operand4* erfolgt beim Start des READLOB-Statements. Wird der Wert von *operand4* innerhalb der READLOB-Schleife geändert, hat das keine Auswirkung auf den zurzeit gelesenen Datensatz. Wird diese Option nicht angegeben, dann wird standardmäßig das *ISN-Feld des zuletzt aktiven Datenbank-Statements verwendet.
STARTING AT OFFSET=*operand5*	**STARTING AT OFFSET-Klausel:** Dient zur Angabe des Start-Offset innerhalb des LOB-Feldes, d.h., ab wo das Lesen des ersten Segments beginnen soll. Das erste Byte des LOB-Feldes hat den Versatz Null (0).

Syntax-Element	Beschreibung
	operand5 muss entweder in Form einer numerischen Konstante oder als benutzerdefinierte Variable ohne Dezimalstellen angegeben werden. Das Feld wird durch die Ausführung des READLOB-Statements nicht verändert. Wird die Klausel nicht angegeben, dann wird Null (0) als Start-Versatz angenommen, was zur Folge hat, dass das LOB-Feld ab dem Feldanfang gelesen wird. Siehe auch *NUMBER während der Schleifenverarbeitung, die im Abschnitt ***Bei READLOB vorhandene Systemvariablen*** weiter unten beschrieben wird.
END-READLOB (*r*)	**Ende des READLOB-Statements:** Im Structured Mode muss das für Natural reservierte Schlüsselwort END-READLOB zum Beenden des READLOB-Statements benutzt werden. Im Structured Mode können Sie bei END-READLOB Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des READLOB-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.
LOOP (*r*)	

Bei READLOB verfügbare Systemvariablen

Beim READLOB-Statement stehen die Natural-Systemvariablen *ISN, *COUNTER und *NUMBER zur Verfügung.

Format/Länge dieser Systemvariablen ist P10. Die Format/Länge-Werte können nicht geändert werden.

In der folgenden Tabelle wird die Verwendung dieser Natural-Systemvariablen in Verbindung mit dem READLOB-Statement erläutert:

Systemvariable	Erläuterung
*ISN	Enthält die Adabas-interne Folgenummer (ISN) des Datensatzes, der zurzeit verarbeitet wird. Das mit einem READLOB-Statement immer auf denselben Datensatz zugreift, bleibt der von der Systemvariablen über alle Schleifen-Iterationen zurückgegebene Wert immer derselbe.
*COUNTER	Enthält die Anzahl der durchlaufenen Verarbeitungsschleifen.
*NUMBER	Diese Systemvariable wird zweifach verwendet:

Systemvariable	Erläuterung	
	Vor dem Aufruf:	Die Systemvariable gibt den Byte-Offset im LOB-Feld an, ab dem das Segment gelesen werden soll. Der Wert Null (0) stellt das Byte am linken Rand des LOB-Feldes dar. Dies gilt nicht bei der ersten Schleifen-Iteration. In diesem Fall wird der Lese-Offset durch die STARTING AT OFFSET-Klausel bestimmt.
	Nach dem Aufruf:	Wenn die Daten gefunden wurden (d.h., der Offset war kleiner als die Länge des LOB-Feldes), erhält die Systemvariable den Offset-Wert plus die Segmentlänge. Das kann zu einem *NUMBER-Wert führen, der größer ist als die Länge des gesamten LOB-Feldes. Falls keine Daten gefunden wurden (d.h., der Offset war größer als die oder gleich der Länge des LOB-Feldes), bleibt der Wert der Systemvariablen *NUMBER unverändert.
	Falls ein konsekutives Lesen über ein LOB-Feld angefordert wird, darf der *NUMBER-Wert innerhalb des READLOB-Statements nicht geändert werden, da *NUMBER den Offset für die exakte Fortsetzung mit dem nächsten Segment in der darauffolgenden Schleifen-Iteration enthält. Falls jedoch eine Fortsetzung an einer anderen Stelle innerhalb des LOB-Feldes (Neupositionierung) gewünscht wird, können Sie den *NUMBER-Wert auf diesen Offset ändern. Wird *NUMBER zurückgesetzt, hat dies zur Folge, dass das nächste Segment vom LOB-Anfang kommt. Wird *NUMBER um (*n*) erhöht, wird die entsprechende Anzahl Bytes bei der LOB-Feldverarbeitung übersprungen.	

Funktionstechnische Überlegungen bei READLOB

- Das READLOB-Statement liest den Datensatz immer ohne Hold (Sperre). Um die Stabilität der LOB-Daten während der Durchsuchung des Feldes durch das READLOB-Statement sicherzustellen (d.h. zur Vermeidung einer Aktualisierung durch andere Benutzer), kann der Datensatz mit dem Datenbank-Statement, welches den ISN-Wert liefert, in den Hold-Status gesetzt werden, und zwar
  - in einen *Exclusive Hold* bedingt durch ein UPDATE- oder DELETE-Statement, welches sich auf ein äußeres READ- oder FIND-Statement bezieht, oder
  - in einen *Shared Hold* mit einer IN SHARED HOLD-Option, die in einem READ- oder FIND-Statement angewendet wird.
- Da das READLOB-Statement den Datensatz immer ohne Hold (Sperre) liest, darf sich ein UPDATE-, DELETE- oder GET SAME-Statement nicht auf ein READLOB-Statement beziehen.

Beispiele READLOB

- Beispiel 1 - Datensatznummer aus der READ-Schleife holen
- Beispiel 2 - Datensatznummer über benutzerdefinierte Variable holen
- Beispiel 3 - Datensatznummer aus der READ-Schleife holen/ mit Exclusive Hold)

Beispiel 1 - Datensatznummer aus der READ-Schleife holen

```
DEFINE DATA LOCAL
1 VIEW01 VIEW OF ..
  2 NAME
  2 L@LOBFIELD
1 VIEW02 VIEW OF ..
  2 LOBFIELD_SEGMENT                /* LOB field defined in DDM with (A1000).
END-DEFINE
*
READ VIEW01 BY NAME = 'SMITH'       /* Outer statement reads all demanded record
                                    /* fields, except the LOB field.
     IN SHARED HOLD MODE=Q          /* Set record into shared hold to enforce LOB
                                    /* data stability during READLOB.
  DISPLAY NAME 'Total-length LOB-field' L@LOBFIELD
  READLOB VIEW02                    /* Record number used from active record of
                                    /* READ statement.
                                    /* LOB is read in segments with length 1000.
      STARTING AT OFFSET = 2000     /* Start to read the LOB field at byte 2000.
    WRITE 'Loop counter:' *COUNTER  10X ' Next offset:' *NUMBER
    PRINT VIEW02.LOBFIELD_SEGMENT
  END-READLOB
END-READ
END
```

Beispiel 2 - Datensatznummer über benutzerdefinierte Variable holen

```
DEFINE DATA LOCAL
1 #ISN  (I4)
1 #CNT  (I4)
1 #OFF  (I4)
1 VIEW02 VIEW OF ..
  2 LOBFIELD_SEGMENT            /* LOB field defined in DDM with (A1000).
END-DEFINE
*
INPUT (AD=T)
  / ' Read record (ISN):' #ISN
  / 'Number of segments:' #CNT
  / '   Start at offset:' #OFF
*
READLOB (#CNT) VIEW02           /* Read max. (#CNT) segments with length 1000.
```

```
    WITH ISN = #ISN             /* Record number provided by user.
                                /* Record is not in hold.
    STARTING AT OFFSET = #OFF   /* Start to read the LOB field at byte (#OFF).
  WRITE 'Loop counter:' *COUNTER  10X ' Next offset:' *NUMBER
  PRINT VIEW02.LOBFIELD_SEGMENT
END-READLOB
END
```

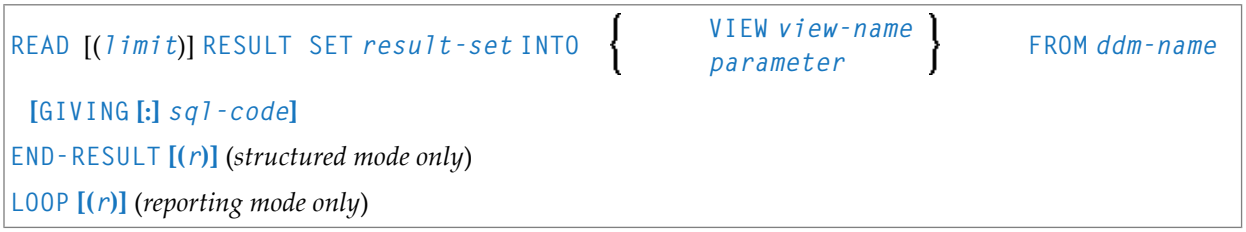
Beispiel 3 - Datensatznummer aus der READ-Schleife holen/ mit Exclusive Hold)

```
DEFINE DATA LOCAL
1 VIEW01 VIEW OF ..
  2 NAME
1 VIEW02 VIEW OF ..
  2 LOBFIELD_SEGMENT                 /* LOB field defined in DDM with (A1000).
END-DEFINE
*
R1. READ VIEW01 BY NAME = 'SMITH'    /* Outer statement reads all demanded record ↩
fields,
                                     /* except the LOB field.
  DISPLAY NAME
  READLOB VIEW02                     /* Record number used from active record of ↩
READ stmt.
                                     /* LOB is read in segments with length 1000.
      STARTING AT OFFSET = 2000      /* Start to read the LOB field at byte 2000.
    WRITE 'Loop counter:' *COUNTER  10X ' Next offset:' *NUMBER
    PRINT VIEW02.LOBFIELD_SEGMENT
  END-READLOB
  ...
  UPDATE (R1.)                     /* Set record into exclusive hold that enforces ↩
LOB
                                     /* data stability during READLOB.
  END OF TRANSACTION
END-READ
END
```

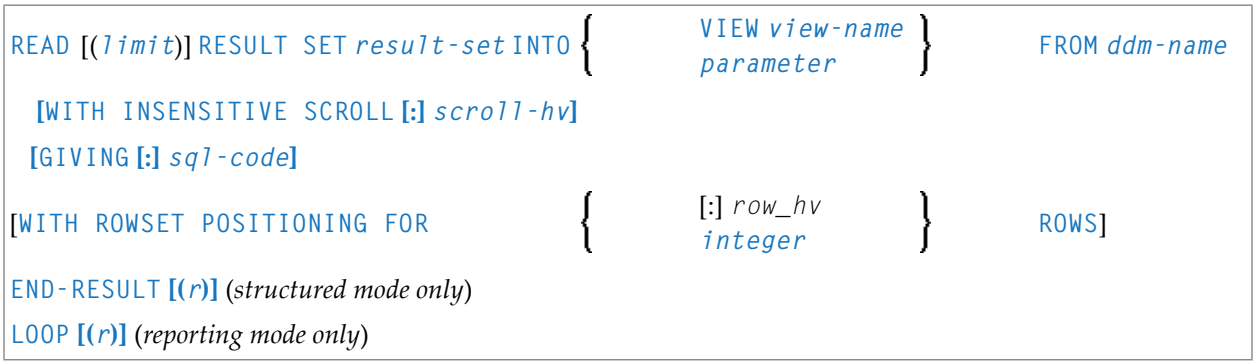
105 READ RESULT SET (SQL)

- Funktion READ RESULT SET (SQL) 858
- Einschränkung bei READ RESULT SET (SQL) 859
- Syntax-Beschreibung READ RESULT SET (SQL) 859
- Beispiel für READ RESULT SET (SQL) 861

Common Set-Syntax:



Extended Set-Syntax:



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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *READ RESULT SET - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion READ RESULT SET (SQL)

Das SQL-Statement `READ RESULT SET` kann nur in Verbindung mit einem `CALLDBPROC`-Statement verwendet werden. Es dient dazu, eine Ergebnismenge (Result Set) zu lesen, die von einer mit einem vorhergehenden `CALLDBPROC`-Statement aufgerufenen Stored Procedure erzeugt wurde.

Einschränkung bei READ RESULT SET (SQL)

Dieses Statement steht nur bei Natural for Db2 zur Verfügung.

Syntax-Beschreibung READ RESULT SET (SQL)

Syntax-Element	Beschreibung
limit	**Limit-Option:** Sie können die Anzahl der zu lesenden Zeilen begrenzen. Sie können das Limit entweder als numerische Konstante (0 - 4294967295) oder als Variable mit Format N, P oder I angeben.
result-set	**Resultset (Ergebnismenge):** Als *result-set* geben Sie eine Resultset-Locator-Variable an, die mit einem vorangehenden CALLDBPROC-Statement gefüllt wurde. *result-set* muss eine Variable von Format/Länge I4 sein. **Anmerkung:** Wenn zwischen dem CALLDBPROC-Statement und dem READ RESULT SET-Statement eine Syncpoint-Operation stattfand, kann das READ RESULT SET-Statement nicht mehr auf die Result Sets zugreifen.
INTO	**INTO-Klausel:** Mit der INTO-Klausel geben Sie in dem Programm die Zielfelder an, die mit dem Resultset (Ergebnismenge) gefüllt werden sollen. Sie können mit der INTO-Klausel entweder einzelne Parameter oder eine oder mehrere Views angeben, und zwar so wie sie im DEFINE DATA-Statement definiert wurden.
VIEW *view-name*	**VIEW-Klausel:** Als *view-name* geben Sie eine View (Datensicht) an, deren Felder die Spalten des Resultset (Ergebnismenge) aufnehmen, der durch die mit dem CALLDBPROC-Statement aufgerufene Stored Procedure erstellt wurden. Die Anzahl der Spalten des Resultset (Ergebnismenge) muss der Anzahl der Felder entsprechen, die in der View angegeben sind (dazu zählen keine Gruppenfelder, Redefinierungsfelder und Indikator-Felder).
parameter	**Parameter:** Als Parameter geben Sie den Namen eines Feldes an, das eine Spalte des Resultset (Ergebnismenge) aufnimmt, der durch die mit dem CALLDBPROC-Statement aufgerufene Stored Procedure erstellt wurde.
FROM *ddm-name*	**DDM-Name:**

Syntax-Element	Beschreibung
	Als *ddm-name* geben Sie den Namen des DDMs an, das benutzt wird, um auf die Datenbank zuzugreifen, die die Stored Procedure ausführt. Weitere Informationen siehe *ddm-name*.
WITH INSENSITIVE SCROLL [:] *scroll_hv*	**WITH INSENSITIVE SCROLL-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Mit dieser Klausel wird die Anwendung veranlasst, einen beliebig positionierbaren Insensitive Cursor zu verwenden, um auf das von dem zuvor aufgerufenen Stored Procedure erstellten Resultset (Ergebnismenge) zuzugreifen. Damit diese Klausel verwendet werden kann, muss die Stored Procedure das Resultset (Ergebnismenge) mit einem positionierbaren Cursor erstellt haben. *scroll_hv* muss eine alphanumerische Natural-Variable sein, die die Scroll-Richtung enthält. *scroll_hv* wird bei jeder Ausführung der READ RESULT SET-Verarbeitungsschleife ausgewertet. Wenn außerdem die Option GIVING *sqlcode* angegeben wird, bleibt die Verarbeitungsschleife auch dann geöffnet, wenn der SQLCODE +100 (Zeile nicht gefunden) von der RDBMS zurückgegeben wird. Die Verarbeitung wird beendet, wenn die Anwendung ein ESCAPE-Statement absetzt oder wenn der SQLCODE +100 fünf Mal nacheinander auftritt, ohne dass eine Terminal-Ein-/Ausgabe erfolgt ist. Wird die Option GIVING *sqlcode* nicht angegeben, so wird die Verarbeitungsschleife beendet, wenn ein SQLCODE von der RDBMS zurückgeliefert wird, der ungleich 0 (erfolgreich) ist.
GIVING *sqlcode*	**GIVING *sqlcode*-Klausel:** Mit dieser optionalen Klausel erhalten Sie den SQLCODE der SQL-FETCH-Operation, mit der das Resultset (Ergebnismenge) verarbeitet wurde. Wenn Sie diese Option angeben und der SQLCODE der SQL-Operation ungleich 0 ist, wird keine Natural-Fehlermeldung ausgegeben. In diesem Fall muss die als Reaktion auf den SQLCODE-Wert auszuführende Aktion im aufrufenden Natural-Objekt programmiert werden. Das Feld *sqlcode* muss eine Variable mit Format/Länge I4 sein. Wenn Sie die Option GIVING *sqlcode* nicht verwenden, gibt Natural eine Fehlermeldung aus, falls der SQLCODE ungleich 0 ist.
WITH ROWSET POSITIONING FOR ... ROWS	**WITH ROWSET POSITIONING FOR ... ROWS-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Mit dieser Klausel wird die Anwendung veranlasst, mehrere Zeilen mit Daten aus dem Resultset (Ergebnismenge) zu lesen, der von der zuvor aufgerufenen Stored Procedure erstellt worden ist. Die Variable *integer* oder *:row_hv* bestimmt die Anzahl der abgerufenen Zeilen.

Syntax-Element	Beschreibung
	Wenn außerdem die Option `GIVING sqlcode` angegeben wird, bleibt die Verarbeitungsschleife auch dann geöffnet, wenn der SQLCODE `+100` (Zeile nicht gefunden) von der RDBMS zurückgegeben wird. Die Verarbeitung wird beendet, wenn die Anwendung ein `ESCAPE`-Statement absetzt oder wenn der SQLCODE `+100` fünf Mal nacheinander auftritt, ohne dass eine Terminal-Ein-/Ausgabe erfolgt ist. Wird die Option `GIVING sqlcode` nicht angegeben, so wird die Verarbeitungsschleife beendet, wenn ein SQLCODE von der RDBMS zurückgeliefert wird, der ungleich `0` (erfolgreich) ist.
`END-RESULT` (*r*)	**Ende des READ RESULT SET-Statement:** Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-RESULT` zum Beenden des `READ RESULT SET`-Statements verwendet werden. Im Structured Mode können Sie bei `END-RESULT` Labels oder Zeilennummern angeben. Im Reporting Mode muss das Natural-Statement `LOOP` um Beenden des `READ RESULT SET`-Statements verwendet werden. Im Reporting Mode können Sie bei `LOOP` Labels oder Zeilennummern angeben.
`LOOP` (*r*)	

Beispiel für READ RESULT SET (SQL)

Siehe *Example of CALLDBPROC/READ RESULT SET* im Abschnitt *CALLDBPROC - SQL* in der *Natural for Db2*-Dokumentation.

106 READ WORK FILE

- Funktion READ WORK FILE 864
- Syntax 1 - READ WORK FILE mit Verarbeitungsschleife 864
- Syntax 2 - READ WORK FILE ohne Verarbeitungsschleife 865
- Syntax-Beschreibung READ WORK FILE 866
- Feldlängen bei READ WORK FILE 869
- Variabler Index-Bereich bei READ WORK FILE 869
- Verarbeitung dynamischer Variablen bei READ WORK FILE 870
- Verarbeitung von X-Arrays bei READ WORK FILE 870
- Beispiele READ WORK FILE 870

Verwandte Statements: CLOSE WORK FILE | DEFINE WORK FILE | WRITE WORK FILE

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

Funktion READ WORK FILE

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.

Auf Großrechnern kann dieses Statement nur in einem Programm verwendet werden, das unter Com-plete, CICS, TSO oder im Batch-Betrieb ausgeführt wird. Die entsprechende JCL muss mit der Ausführungs-JCL bereitgestellt werden, wenn eine Arbeitsdatei gelesen werden soll. Näheres hierzu siehe *Operations*-Dokumentation.

Informationen bezüglich Arbeitsdateien finden Sie auch in der Beschreibung des Profilparameters WORK in der *Parameter-Referenz*.

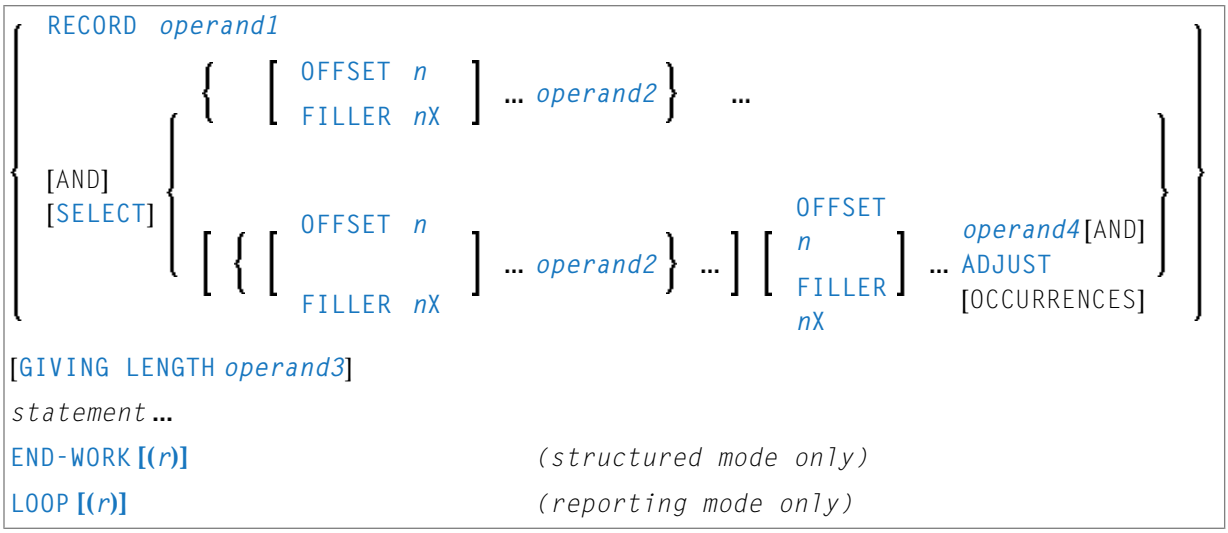
Informationen zur Unicode- und Codepage-Unterstützung siehe *Arbeitsdateien und Druckdateien* in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

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.

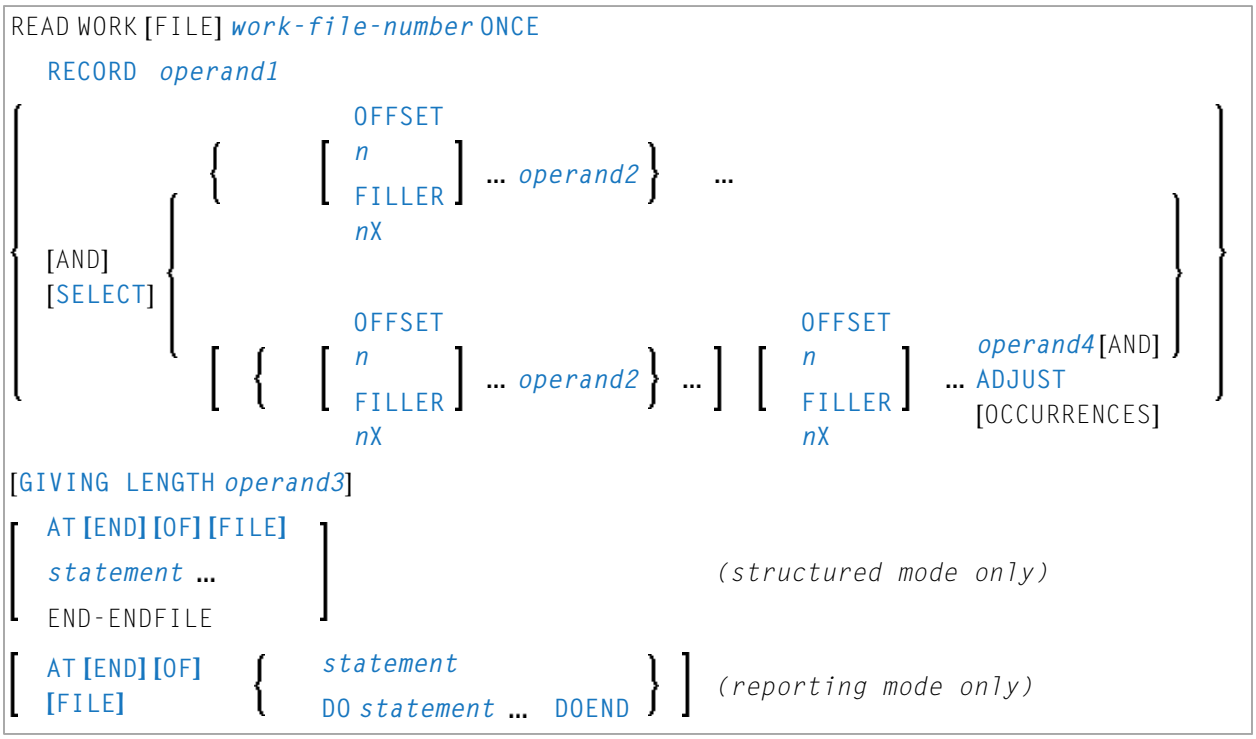
Syntax 1 - READ WORK FILE mit Verarbeitungsschleife

```
READ WORK [FILE] work-file-number
```



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

Syntax 2 - READ WORK FILE ohne Verarbeitungsschleife



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

Syntax-Beschreibung READ WORK FILE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L	C			ja	ja
operand2		S	A	G		A	U	N	P	I	F	B	D	T	L	C			ja	ja
operand3		S								I									ja	ja
operand4			A			A	U	N	P	I	F	B	D	T	L	C			nein	nein

Format C ist bei Natural Connection nicht gültig.

Siehe auch ***Feldlängen***.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Arbeitsdateinummer:** Gibt die für Natural definierte Nummer der Arbeitsdatei an, die gelesen werden soll. Die Nummer der Arbeitsdatei ist entweder ■ eine numerische Konstante im Wertebereich 1:32 oder ■ eine mit der Klausel CONSTANT definierte numerische Variable im Format B/N/P/I, die einen Wert im Bereich 1:32 zuweist. Dezimalstellen bei den Formaten (N/P) sind nicht erlaubt.
ONCE	**ONCE-Option:** 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. 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 zurückpositioniert, welcher dann bei der nächsten Ausführung des READ WORK FILE ONCE-Statements gelesen wird.
RECORD *operand1* FILLER *n*X	**RECORD-Option:**

Syntax-Element	Beschreibung
	Im Reporting Mode muss eine Operandenliste (*operand1*) entsprechend dem Layout des Datensatzes verfügbar gemacht werden. `FILLER nX` muss angegeben werden, wenn die Gesamtlänge der Operanden kleiner als die Länge des Arbeitsdateidatensatzes ist. Im Structured Mode oder wenn der zu verwendende Datensatz mittels einem `DEFINE DATA`-Statement definiert wird, kann nur ein Feld (oder eine Gruppe) angegeben werden, und `FILLER` darf nicht verwendet werden. Das Feld (oder die Gruppe) muss lang genug sein, um den gesamten Arbeitsdateidatensatz aufnehmen zu können. 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 (*operand1*) mit Leerzeichen; demzufolge wird bei einer End-of-File-Bedingung ein leerer Bereich zurückgegeben. Kurze Datensätze werden mit Leerzeichen aufgefüllt. **Die `RECORD`-Option kann nicht benutzt werden:** ■ Wenn eine Entire Connection-Arbeitsdatei gelesen wird. ■ Wenn dynamische Variablen benutzt werden.
`SELECT`	**SELECT-Option (Voreinstellung):** Wird die `SELECT`-Option verwendet, so werden nur die in der Operandenliste (*operand2*) angegebenen Felder zur Verfügung gestellt. Die Position der Felder im Eingabedatensatz kann durch eine `OFFSET`- und/oder `FILLER`-Angabe spezifiziert werden.
	OFFSET *n* — `OFFSET 0` spezifiziert das erste Byte des Datensatzes. `OFFSET` kann nicht für Arbeitsdateien des Typs `UNFORMATTED` angegeben werden.
	FILLER *n*X — Bedeutet, dass *n* Bytes des Eingabedatensatzes übersprungen werden.
	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. Wenn ein Datensatz nicht alle in der `SELECT`-Option angegebenen Felder füllt, gilt folgendes: ■ Ein Feld, das nur teilweise gefüllt wurde, wird mit Leerzeichen bzw. Nullen aufgefüllt.

Syntax-Element	Beschreibung
	■ Ein Feld, das gar nicht gefüllt wurde, behält denselben Inhalt wie zuvor. Beim Lesen des Datei-Typs CSV werden die Optionen OFFSET und FILLER ignoriert.
operand4 AND ADJUST OCCURRENCES	**ADJUST-Klausel:** Geben Sie ein eindimensionales X-Array mit vollständigem Bereich (*) an. Das X-Array wird auf bzw. um die Anzahl der Ausprägungen erweitert bzw. reduziert, die zur Aufnahme aller gelesenen Daten benötigt werden. Siehe ***Verarbeitung von X-Arrays*** weiter unten. **Anmerkung:** Diese Klausel wird von Entire Connection nicht unterstützt.
GIVING LENGTH *operand3*	**GIVING LENGTH-Klausel** Mit dieser Klausel können Sie die tatsächliche Länge des gelesenen Datensatzes feststellen. Die Länge (Anzahl der Bytes) erhalten Sie in *operand3*. *operand3* muss mit Format/Länge I4 definiert sein. Wenn die Arbeitsdatei als TYPE UNFORMATTED definiert ist, verweist die zurückgegebene Länge auf die Anzahl der aus dem Byte-Strom gelesenen Bytes, einschließlich der beim FILLER-Operanden übersprungenen Bytes.
AT END OF FILE	**AT END OF FILE-Klausel** Diese Klausel kann nur zusammen mit der **ONCE-Option** verwendet werden. Wenn Sie die ONCE-Option verwenden, dann sollten Sie auch eine AT END OF FILE-Klausel angeben, um festzulegen, was beim Auftreten einer End-of-File-Bedingung geschehen soll. Wenn Sie die ONCE-Option nicht verwenden, wird eine End-of-File-Bedingung wie das normale Ende einer Verarbeitungsschleife behandelt.
END-WORK (*r*)	**Ende des READ WORK FILE-Statements:** Im Structured Mode ohne Verarbeitungsschleife muss dass für Natural reservierte Schlüsselwort END-WORK zum Beenden des READ WORK FILE-Statements verwendet werden. Im Structured Mode können Sie bei END-WORK Labels oder Zeilennummern angeben.
LOOP (*r*)	**Ende des READ WORK FILE-Statements:** Im Reporting Mode ohne Verarbeitungsschleife muss dass Natural-Statement LOOP zum Beenden des READ WORK FILE-Statements verwendet werden. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

Feldlängen bei READ WORK FILE

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*.

Variabler Index-Bereich bei READ WORK FILE

Zum Auslesen eines Arrays aus einer Arbeitsdatei können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel:

```
READ WORK FILE work-file-number #ARRAY (I:J)
```

Verarbeitung dynamischer Variablen bei READ WORK FILE

Arbeitsdateityp	Verarbeitung
UNFORMATTED	Das Lesen einer dynamischen Variable von einer UNFORMATTED-Arbeitsdatei bewirkt, dass der komplette Rest der Datei in die Variable (von der aktuellen Position) gestellt wird. Wenn die Datei mehr als 1073741824 Bytes umfasst, dann werden mindestens 1073741824 Bytes in die Variable gestellt.
FORMATTED	Das Lesen einer dynamischen Variable von einer Arbeitsdatei vom Typ FORMATTED bewirkt, dass der Rest des Arbeitsdateidatensatzes in die Variable gestellt wird. So wird die Länge der dynamischen Variable bei Arbeitsdateien mit variabler Datensatzlänge bei jeder Ausführung des READ WORK FILE-Statements so angepasst, dass sie exakt der Länge des zurzeit gelesenen Datensatzes entspricht.

Verarbeitung von X-Arrays bei READ WORK FILE

Wird die ADJUST-Klausel *nicht* angegeben, werden X-Arrays genauso wie reguläre Arrays behandelt, d.h. ihre vorhandenen Ausprägungen werden gefüllt.

Wird dieADJUST-Klausel angegeben,wird ein mit vollständigem Bereich (*) angegebenes eindimensionales X-Array so angepasst, dass es alle Daten aus dem restlichen Datensatz für eine Arbeitsdatei vom Typ FORMATTED bzw. alle Daten aus dem restlichen Datensatz für eine Arbeitsdatei vom Typ UNFORMATTED aufnehmen kann.

Beispiele READ WORK FILE

- Beispiel 1 - READ WORK FILE
- Beispiel 2 - READ WORK FILE Formatiert mit dynamischer Variablen
- Beispiel 3 - READ WORK FILE Unformatiert mit dynamischer Variablen
- Beispiel 4 - READ WORK FILE Formatiert mit X-array und ADJUST bei Ausprägungen
- Beispiel 5 - READ WORK FILE Unformatiert mit X-array und ADJUST bei Ausprägungen

- Beispiel 6 - READ WORK FILE mit einer mit CONST definierten numerischen Variablen als Arbeitsdateinummer

Beispiel 1 - READ WORK FILE

```
** 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 Programs 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             ↩
```

Beispiel 2 - READ WORK FILE Formatiert mit dynamischer Variablen

```
** Example 'RWFEX2': READ WORK FILE - Formatted with dynamic variable
************************************************************************
DEFINE DATA LOCAL
  1 #DYNA (A) DYNAMIC
END-DEFINE
*
DEFINE WORK FILE 1 TYPE 'FORMATTED'
*
WRITE WORK FILE 1 VARIABLE 'text1 text2 text3 '
WRITE WORK FILE 1 VARIABLE 'text4 text5'
*
READ WORK FILE 1 AND SELECT #DYNA
  DISPLAY *LENGTH(#DYNA) #DYNA (AL=40)
  /*
  /* Length: 18  Dyn.Var: 'text1 text2 text3'
  /* Length: 11  Dyn.Var: 'text4 text5'
END-WORK
*
END
```

Ausgabe des Programs RWFEX2:

```
Page      1                                                  11-07-15  09:21:09

  LENGTH                      #DYNA
----------- ----------------------------------------

         18 text1 text2 text3
         11 text4 text5                                                      ↩
```

Beispiel 3 - READ WORK FILE Unformatiert mit dynamischer Variablen

```
** Example 'RWFEX3': READ WORK FILE - Unformatted with dynamic variable
************************************************************************
DEFINE DATA LOCAL
  1 #DYNA (A) DYNAMIC
END-DEFINE
*
DEFINE WORK FILE 1 TYPE 'FORMATTED'
*
WRITE WORK FILE 1 VARIABLE 'text1 text2 text3 '
WRITE WORK FILE 1 VARIABLE 'text4 text5'
*
DEFINE WORK FILE 1 TYPE 'UNFORMATTED'
*
READ WORK FILE 1 AND SELECT #DYNA
  DISPLAY *LENGTH(#DYNA) #DYNA (AL=40)
  /*
```

```
  /* Length: 29  Dyn.Var: 'text1 text2 text3 text4 text5'
END-WORK
*
END
```

Ausgabe des Programs RWFEX3:

```
Page      1                                                  11-07-15  09:31:04

  LENGTH                     #DYNA
----------- ----------------------------------------

         29 text1 text2 text3 text4 text5  ↩
```

Beispiel 4 - READ WORK FILE Formatiert mit X-array und ADJUST bei Ausprägungen

```
** Example 'RWFEX4': READ WORK FILE - Formatted with X-array
**                                    and ADJUST its occurrences
************************************************************************
DEFINE DATA LOCAL
  1 #ARR (A6/1:*)
  1 #OCC (I4)
END-DEFINE
*
DEFINE WORK FILE 1 TYPE 'FORMATTED'
*
WRITE WORK FILE 1 VARIABLE 'text1 text2 text3 '
WRITE WORK FILE 1 VARIABLE 'text4 text5'
*
READ WORK FILE 1 AND SELECT #ARR(*) AND ADJUST OCCURRENCES
  #OCC := *OCCURRENCE(#ARR)
  DISPLAY #OCC #ARR(1:#OCC)
  /*
  /* Occurrences: 3 Array(*): 'text1', 'text2', 'text3'
  /* Occurrences: 2 Array(*): 'text4', 'text5'
END-WORK
*
END ↩
```

Ausgabe des Programs RWFEX4:

```
Page      1                                                  11-07-15  09:36:13

   #OCC      #ARR
----------- ------

          3 text1
            text2
            text3
```

```
          2 text4
            text5                                                                ↩
```

Beispiel 5 - READ WORK FILE Unformatiert mit X-array und ADJUST bei Ausprägungen

```
** Example 'RWFEX5': READ WORK FILE - Unformatted with X-array
**                                   and ADJUST its occurrences
************************************************************************
DEFINE DATA LOCAL
  1 #ARR (A6/1:*)
  1 #OCC (I4)
END-DEFINE
*
DEFINE WORK FILE 1 TYPE 'FORMATTED'
*
WRITE WORK FILE 1 VARIABLE 'text1 text2 text3 '
WRITE WORK FILE 1 VARIABLE 'text4 text5'
*
DEFINE WORK FILE 1 TYPE 'UNFORMATTED'
*
READ WORK FILE 1 AND SELECT #ARR(*) AND ADJUST OCCURRENCES
  #OCC := *OCCURRENCE(#ARR)
  DISPLAY #OCC #ARR(1:#OCC)
  /*
  /*Occurrences: 5 Array(*): 'text1', 'text2', 'text3', 'text4', 'text5'
END-WORK
*
END ↩
```

Ausgabe des Programs RWFEX5:

```
Page      1                                                  11-07-15  09:41:25

   #OCC      #ARR
----------- ------

          5 text1
            text2
            text3
            text4
            text5                                                                ↩
```

Beispiel 6 - READ WORK FILE mit einer mit CONST definierten numerischen Variablen als Arbeitsdateinummer

```
** Example 'RWFEX6': READ WORK FILE - with numeric CONST variable as
**                                    work file number
** see similar example RWFEX5 with numeric constant
************************************************************************
DEFINE DATA LOCAL
  1 #ARR    (A6/1:*)
  1 #OCC    (I4)
  1 #WF-1   (N4) CONST<1>
END-DEFINE
*
DEFINE WORK FILE #WF-1 TYPE 'FORMATTED'
*
WRITE WORK FILE  #WF-1#WF-1 VARIABLE 'text1 text2 text3 '
WRITE WORK FILE  #WF-1 VARIABLE 'text4 text5'
*
DEFINE WORK FILE #WF-1 TYPE 'UNFORMATTED'
*
READ WORK FILE #WF-1 AND SELECT #ARR(*) AND ADJUST OCCURRENCES
  #OCC := *OCCURRENCE(#ARR)
  DISPLAY #OCC #ARR(1:#OCC)
  /*
  /*Occurrences: 5 Array(*): 'text1', 'text2', 'text3', 'text4', 'text5'
END-WORK
*
END
```

Ausgabe des Programs RWFEX6:

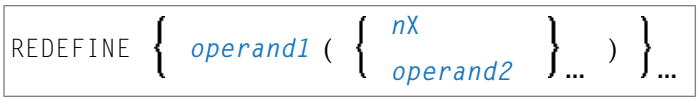
```
Page      1                                                  21-12-20  17:42:43

   #OCC      #ARR
----------- ------

          5 text1
            text2
            text3
            text4
            text5
```

107 REDEFINE

- Funktion REDEFINE 878
- Einschränkung bei REDEFINE 878
- Syntax-Beschreibung REDEFINE 878
- Beispiele REDEFINE 879



Dieser Abschnitt behandelt folgende Themen:

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

Gehört zur Funktionsgruppe: ***Reporting Mode-Statements***

Funktion REDEFINE

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 bei REDEFINE

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 REDEFINE

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`REDEFINE` *operand1* *operand2*	**Methode der Redefinition:** Die Byte-Positionen von *operand1* werden unabhängig vom Format von links nach rechts redefiniert. Das Format von *operand2* muss nicht mit dem von *operand1* identisch sein. Jedoch sollten die Daten an den Byte-Positionen von *operand2* mit den Formatangaben für *operand2* übereinstimmen, um unerwünschte Ergebnisse im Ausgabe-Report zu vermeiden. Wenn beispielsweise ein alphanumerisches Feld in ein numerisches Feld redefiniert wird und keine numerischen Daten gemäß der Formatangabe enthält, kann die Verwendung des Feldes einen Programmabbruch zur Folge haben. Eine empfohlene Vorgehensweise zur Überprüfung numerischer Werte in einem alphanumerischen Feld finden Sie unter ***Beispiel 5: Überprüfung numerischer Werte in einem alphanumerischen Feld***. **Weitere Redefinition:** Ein mit einem `REDEFINE`-Statement neudefiniertes Feld kann mit einem weiteren `REDEFINE`-Statement nochmals redefiniert werden.
n `X`	**Füllbyte-Notation:** Mit der Notation *n*`X` können Sie in der redefinierten Variable oder dem nachgestellten Feld *n* Füllbytes definieren. Nachgestellte Füllbytes müssen nicht unbedingt angegeben werden.

Beispiele REDEFINE

- Beispiel 1 - REDEFINE-Statement
- Beispiel 2 - REDEFINE-Statement
- Beispiel 3 - REDEFINE-Statement
- Beispiel 4 - REDEFINE-Statement
- Beispiel 5: Überprüfung numerischer Werte in einem alphanumerischen Feld

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 Hexadezimalformat).

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
--------- ---- ------ -----

 20140326  26    3     2014
```

Beispiel 5: Überprüfung numerischer Werte in einem alphanumerischen Feld

Wenn ein alphanumerisches Feld, das sowohl Buchstaben als auch Zahlen enthält, als numerisches Feld neu definiert wird, kann der Zugriff auf einen Wert, der für das numerische Format ungültig ist, zu einem Abbruch führen. Wenn Sie in einem alphanumerischen Feld auf einen numerischen Wert prüfen müssen, wird empfohlen, die Option IS zu verwenden. Der numerische Wert kann dann mit der Funktion VAL.

Beispiel:

```
DEFINE DATA LOCAL
1 #A (A4)
1 REDEFINE #A
 2 #N(N4)
END-DEFINE
#A := '-123'
IF #A IS (N4) THEN /* check if #A can be converted to numeric format
 #N := VAL(#A)     /* get the numeric value
 WRITE "TRUE"
ELSE
 WRITE "FALSE"     / * #A contains non-numeric characters and cannot be converted.
END-IF
END
```

Im obigen Beispiel enthält #A den Zeichenwert '-123'. Obwohl #A als alphanumerisches Feld definiert ist, prüft die Bedingung IS (N4), ob sein Inhalt als numerischer Wert interpretiert werden kann. Da '-123' ein gültiger numerischer Wert ist, ist die Bedingung wahr, und mit VAL(#A) kann der numerische Wert in #N ermittelt werden.

```
IF #N = -123
...
END-IF
```

Das obige Beispiel zeigt, dass der Zugriff auf ein als numerisch definiertes Feld, das nicht-numerische Zeichen enthält, zu einem nicht normalen Verhalten führen kann. Daher werden die Funktionen `IS` und `VAL` beim Zugriff auf ein neu definiertes numerisches Feld empfohlen.

108 REDUCE

- Funktion REDUCE 884
- Syntax-Beschreibung REDUCE 884

```
REDUCE { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

Dieser Abschnitt 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 REDUCE

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 REDUCE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S	A			A	U					B						nein	nein
operand2	C	S								I								nein	nein
operand3			A	G		A		N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand4	C	S					U	N	P	I									nein	nein
operand5		S								I4									nein	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
dynamic-clause	Das Statement REDUCE DYNAMIC VARIABLE dient dazu, die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variablen (*operand1*) auf die angegebene Länge (*operand2*) zu verringern. Weitere Informationen, siehe ***DYNAMIC-Klausel*** weiter unten.
operand1	*operand1* ist die dynamische Variable, für die die zugewiesene Länge verringert werden soll.
operand2	*operand2* 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.
array-clause	Mit dieser Klausel wird die Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit *(dim[,dim[,dim]])* angegebene Ober- und Untergrenze verringert. Weitere Informationen siehe ***Array-Klausel*** weiter unten.
operand3	*operand3* 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.
dim operand4	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 verwendet werden sollte, kann ein Stern (*) anstatt *operand4* angegeben werden. Weitere Informationen siehe ***Dimension*** weiter unten.
GIVING *operand5*	Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die GIVING-Klausel angegeben wird, enthält *operand5* die Natural-Fehlernummer, wenn vorher ein Fehler aufgetreten ist, oder Null (0) bei Erfolg.

DYNAMIC-Klausel

```
[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-Klausel

```
[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 *

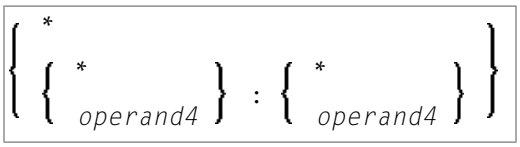
REDUCE ARRAY #ga TO (1:10,1:20) /* (1:10) for the 1st dimension is rejected
                                /* because the dimension is dependent and
                                /* must be specified with (*:*).
```

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:



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.

109 REINPUT

- Funktion REINPUT 890
- Syntax-Beschreibung REINPUT 891
- Beispiele REINPUT 898

```
REINPUT   [FULL] [(statement-parameters)] { USING HELP       }
                                          { WITH-TEXT-option }
          [MARK-option]
          [ALARM-option]
```

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 REINPUT

Das REINPUT-Statement 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 Subprogrammen, 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 Subprogrammen aufgerufen und das REINPUT-Statement steht in einem dieser Subprogramme, 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 Subprogramms 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.

Siehe auch *Dialog-Gestaltung*, *Statements REINPUT/REINPUT FULL* im *Leitfaden zur Programmierung*.

Anmerkung:

Wird ein Eingabe-/Ausgabefeld (Option `(AD=M)`) durch ein `INPUT`-Statement angezeigt, werden die am Schirm sichtbaren Daten nur dann in die Variable zurück übertragen, wenn das Feld als „verändert“ (`MODIFIED`) angesehen wird. Ein Feld erhält den Status `MODIFIED`, wenn eine der folgenden Operationen erfolgt ist:

- Der Inhalt des Feldes wurde verändert (d.h., es wurden *andere* Daten in das Feld eingegeben).
- Die Taste EEOF (Erase to End of Field/Löschen bis Feldende) wird bei einem leeren Feld gedrückt.
- Leerzeichen werden in ein leeres Feld oder nach dem letzten Nicht-Leerzeichen im Feld eingegeben.
- Der Profilparameter `CVMIN` wurde auf `ON` gesetzt und die Daten im Feld werden durch Editiermaßnahmen verändert, die letzten Endes zur Wiederherstellung des Feldinhaltes führen (z.B. durch Überschreiben des ersten Zeichens mit dem vorhandenen Zeichen).

Der Inhalt eines Feldes, der letztendlich unverändert bleibt, wird nicht vom Bildschirmfeld in die Variable übertragen.

Die Ausführung eines `REINPUT`-Statements (ohne `FULL`-Option) hat keinen Einfluss auf den `MODIFIED`-Status eines Eingabe-/Ausgabefeldes. Ein Feld gilt weiterhin als *nicht verändert*, wenn es nicht über das `INPUT`-Statement mittels einer der oben aufgelisteten Operationen verändert wurde. Anders ausgedrückt wird ein Feld als verändert behandelt, wenn mindestens eine der erwähnten Operationen durchgeführt wurde, und zwar unabhängig davon, wie oft das `INPUT`-Statement durch `REINPUT`-Statements (ohne `FULL`-Option) neu gesendet wurde.

Mit anderen Worten wird ein mittels `INPUT`-Statement angezeigter Feldwert, der von einem `REINPUT`-Statement (ohne `FULL`-Option) getriggert wurde, nur dann in die Variable übernommen, wenn der Feldinhalt durch eine der genannten Operationen verändert wurde.

Der `MODIFIED`-Status kann geprüft werden, wenn im Programmcode eine Attributkontrollvariable (Option `CV`) zu dem Feld zugeordnet wurde, das mit der `MODIFIED`-Option z.B. des `IF`-Statements nach dem `INPUT`-Statement geprüft wird.

Syntax-Beschreibung REINPUT

Syntax-Element-Beschreibung:

<table>
<tr><th>Syntax-Element</th><th>Beschreibung</th></tr>
<tr><td>REINPUT FULL</td><td>FULL-Option:

Wenn Sie die Option FULL in einem REINPUT-Statement angeben, wird das entsprechende INPUT-Statement vollständig neu ausgeführt:

■ Bei einem normalen REINPUT-Statement (ohne FULL-Option) werden Inhalte von Variablen, die zwischen INPUT- und REINPUT-Statement geändert wurden, nicht angezeigt; d.h. alle Variablen auf dem Schirm zeigen den Inhalt, den sie hatten, als das INPUT-Statement ursprünglich ausgeführt wurde.

■ Bei einem REINPUT FULL-Statement werden alle nach der ersten Ausführung des INPUT-Statements gemachten Änderungen sichtbar, wenn das INPUT-Statement erneut ausgeführt wird; d.h. alle Variablen auf dem Schirm haben den Inhalt, den sie zum Zeitpunkt der Ausführung des REINPUT-Statements hatten.

Anmerkung: Der Inhalt reiner Eingabefelder (AD=A) wird durch REINPUT FULL wieder gelöscht.

Eine andere Eigenschaft des REINPUT FULL-Statements besteht darin, dass der Status der Kontrollvariable auf NOT MODIFIED (nicht geändert) zurückgesetzt wird. Dies erfolgt nicht mittels des normalen REINPUT-Statements. Um zu überprüfen, ob einer Attribut-Kontrollvariablen der Status MODIFIED (geändert) zugewiesen wurde, benutzen Sie die MODIFIED-Option.

Siehe auch Beispiel 3 - REINPUT FULL mit MARK POSITION.</td></tr>
<tr><td>statement-parameters</td><td>Parameter:

Bei einem REINPUT-Statement angegebene Parameter gelten für alle Felder, die im Statement angegeben sind.

Auf Element-Ebene (Feldebene) gesetzte Parameter (siehe MARK-Option) haben für das betreffende Feld Gültigkeit vor den auf Statement-Ebene gesetzten.

<table><tr><td>Parameter, die mit dem REINPUT-Statement angegeben werden können:</td><td>Angabe (S = auf Statement-Ebene, E = auf Element-Ebene)</td></tr><tr><td>AD - Attributdefinition *</td><td>SE</td></tr><tr><td>CD - Farbdefinition</td><td>S</td></tr></table>* Wenn Sie AD=P auf Statement-Ebene setzen, sind alle Felder geschützt, jedoch nicht die in der MARK-Option angegebenen.

Informationen zu den o.g. Session-Parametern finden Sie in der Parameter-Referenz-Dokumentation.</td></tr>
<tr><td>USING HELP</td><td>USING HELP-Option:

Diese Option bewirkt, dass die für die Eingabemaske (INPUT-Map) definierte Helproutine aufgerufen wird.</td></tr>
</table>

Syntax-Element	Beschreibung
	`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 Maske aufgerufen. Beispiel: `REINPUT USING HELP MARK 3` In diesem Beispiel wird die für das dritte Feld der Eingabemaske definierte Helproutine aufgerufen.
WITH-TEXT-option	**WITH TEXT-Option:** Mit dieser Option können Sie einen Text angeben, der in der Meldungszeile angezeigt werden soll. Siehe ***WITH TEXT Option*** weiter unten.
MARK-option	**MARK-Option** Mit dieser 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.
ALARM-option	**ALARM-Option:** 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 dieser Option 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 } [(attributes)] [,operand3]...7
              { operand2   }
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate													Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I		B *							ja	nein
operand2	C	S				A	U												ja	nein
operand3	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:

Syntax-Element	Beschreibung
operand1	**Meldungstext aus der Natural-Fehlermeldungsdatei:** Als *operand1* geben Sie eine Natural-Fehlernummer an. Natural ruft dann die entsprechende Fehlermeldung aus der Natural- Fehlermeldungsdatei ab. Es können entweder benutzerdefinierte Meldungen oder Natural- Systemmeldungen abgerufen werden. ■ 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 abgerufen Siehe auch ***Beispiel 4 - mit TEXT-Option*** . Natural-Meldungsdateien werden mit der `SYSERR`-Utility erstellt und gepflegt.
operand2	**Meldungstext:** Als *operand2* geben Sie den Text an, der in der Meldungszeile ausgegeben werden soll. Siehe auch ***Beispiel 4 - mit TEXT-Option*** .
attributes	**Ausgabe-Attribute:** Als Attribute können Sie *operand1* oder *operand2* bestimmte Anzeige- und Farbattribute zuordnen. Diese Attribute und die Syntax, die benutzt werden kann, sind im Abschnitt ***Ausgabe-Attribute*** weiter unten beschrieben.
operand3	**Dynamische Ergänzung im Meldungstext:** *operand3* kann in Form einer numerischen Konstanten oder Textkonstanten oder als Name einer Variablen angegeben werden. Die angegebenen Werte dienen dazu, einen Teil des Textes einer mit *operand1* oder *operand2* angegebenen Meldung dynamisch zu generieren. Innerhalb des Textes der Fehlermeldung dient die Notation :*n*: zur Referenzierung von *operand3*, wobei *n* die Ausprägung (1 - 7) von *operand3* darstellt. Siehe auch ***Beispiel 4 - mit TEXT-Option*** .

Syntax-Element	Beschreibung
	Anmerkung: Wird der *operand3* 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 *operand3* um numerische Konstanten handelt, setzen Sie Leerzeichen vor und nach dem Komma, damit es nicht als Dezimalkomma missinterpretiert wird. Alternativ können mehrere Operanden *operand3* 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). Nicht signifikante Nullen oder Leerzeichen werden aus dem Feldwert entfernt, bevor er in einer Meldung angezeigt wird.

Ausgabeattribute

attributes sind zur Text-Anzeige verwendete Ausgabeattribute. Sie können folgende Attribute angeben:

```
{ AD=AD-value... }
{ CD=CD-value... }...
```

Die möglichen Parameterwerte sind in der *Parameter-Referenz*-Dokumentation aufgeführt:

- *AD - Attribut-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 dieser Option können Sie ein bestimmtes Feld markieren, so dass bei der Ausführung des REINPUT-Statements der Cursor in dieses Feld platziert 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
operand4	C	S					N	P	I								ja	nein
operand5	C	S	A				N	P	I								ja	nein

Syntax-Element-Beschreibung:

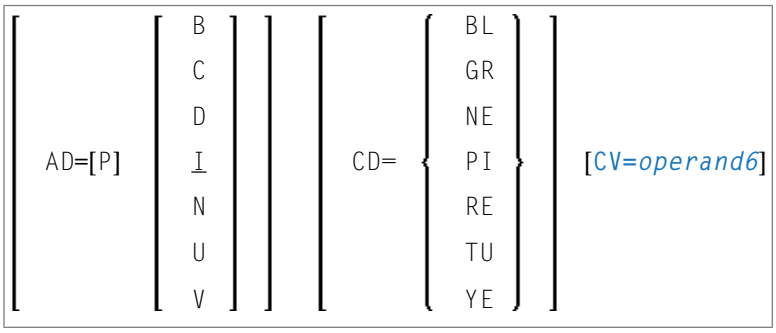
Syntax-Element	Beschreibung

operand5
**fieldname*
	Zu markierendes Feld:
	
	Jedes mit einem INPUT-Statement angegebene Eingabefeld (AD=A oder AD=M) wird durchnummeriert (beginnend mit 1). Sie können als *operand5* die Nummer des Feldes angeben, in das der Cursor platziert werden soll.
	
	Die Notation **fieldname* wird verwendet, um den Cursor in ein Feld zu positionieren, und zwar mittels des (im INPUT-Statement verwendeten) Namens des Feldes.
	
	Ist das betreffende INPUT-Feld ein Array, kann zur Markierung einer oder mehrerer Ausprägungen des Arrays ein eindeutiger Index oder ein Indexbereich angegeben werden.
	
	`INPUT #ARRAY (A1/1:5)`
	`...`
	`REINPUT (AD=P) 'TEXT' MARK *#ARRAY (2:3)`
	
	Ist *operand5* ebenfalls ein Array, so werden die Werte von operand5 als Feldnummern für das INPUT-Array benutzt.
	
	`RESET #X(N2/1:2)`
	`INPUT #ARRAY ...`
	`...`
	`REINPUT (AD=P) 'TEXT' MARK #X (1:2)`

MARK POSITION	**MARK POSITION-Option:** Mit dieser Option können Sie den Cursor an eine bestimmte Stelle, die Sie mit *operand4* angeben, innerhalb eines Feldes platzieren. Siehe auch ***Beispiel 3 - REINPUT FULL mit MARK POSITION*** .
operand4	**Cursor-Position:** *operand4* gibt die Cursor-Position an. *operand4* darf keine Dezimalziffern enthalten.
attributes	**Attribut-Zuweisungen:** Siehe ***Attribut-Zuweisungen*** weiter unten.

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	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
operand6	S	C	nein	nein

Mit dem Attribut `AD=P` können Sie ein Eingabefeld (`AD=A` oder `AD=M`) gegen Eingaben schützen.

Anmerkung: Reine Ausgabefelder (`AD=O`) können nicht durch ein entsprechendes Attribut zu Eingabefeldern gemacht werden.

Informationen zu den Attributen `AD`, `CD` und `CV` finden Sie in der *Parameter-Referenz*-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=operand6)`-Klausel referenziert wird. See ***Example 5 - REINPUT with Attribute Assignment Using a Control Variable***.

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 Farbangabe 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. Siehe auch ***Beispiel 2 - REINPUT mit Attribut-Zuweisung*** .

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 REINPUT

- Beispiel 1 - REINPUT-Statement
- Beispiel 2 - REINPUT mit Attribut-Zuweisung
- Beispiel 3 - REINPUT FULL mit MARK POSITION
- Beispiel 4 - mit TEXT-Optionen
- Beispiel 5 - REINPUT mit Attribut-Zuweisung mittels Kontrollvariable

Beispiel 1 - REINPUT-Statement

```
** Example 'REIEX1': REINPUT
************************************************************************
DEFINE DATA LOCAL
1 #FUNCTION (A1)
1 #PARM     (A1)
END-DEFINE
*
INPUT #FUNCTION #PARM
*
DECIDE FOR FIRST CONDITION
  WHEN #FUNCTION = 'A' AND #PARM = 'X'
    REINPUT 'Function A with parameter X selected.'
            MARK *#PARM
  WHEN #FUNCTION = 'C' THRU 'D'
    REINPUT 'Function 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:

```
#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-Optionen

```
** 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
  MOVE (AD=U CD=RE) TO #CV
  REINPUT FULL WITH TEXT #VAR (CD=YE)
```

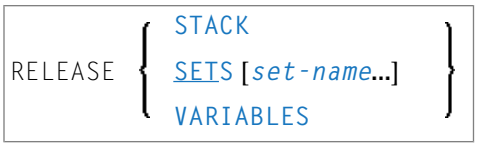
```
                MARK *#HELLO   (CV=#CV)
END-IF
END
```

110 REJECT

Siehe ACCEPT/REJECT.

111 RELEASE

- Funktion RELEASE 906
- Syntax-Beschreibung RELEASE 906
- Beispiel für RELEASE 907



Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements: `STACK` | `FIND` **mit** `RETAIN`**-Option** | `DEFINE DATA GLOBAL`

Funktion RELEASE

Das Statement `RELEASE` dient dazu

- den kompletten Inhalt des Natural-Stack 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 RELEASE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
set-name	C	S				A											nein	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`RELEASE STACK`	**RELEASE STACK-Option:** Mit `RELEASE STACK` löschen Sie alle zurzeit im Natural-Stack abgelegten Kommandos und Daten.
`RELEASE SETS`	**RELEASE SETS-Option:** `RELEASE SETS` gilt nur für Adabas-Datenbanken.

Syntax-Element	Beschreibung
	Wenn Sie nur `RELEASE SETS` ohne Angabe eines *`set-name`* angeben, werden alle ISNs freigegeben, die mit einem `FIND`-Statement, das eine `RETAIN`-Klausel enthält, gehalten wurden.

`RELEASE SETS` *`set-name`*	Mit `RELEASE SET` *`set-name`* geben Sie eine bestimmte ISN frei, zum Beispiel:
	
	`RELEASE SET 'CITY-SET'`
	
	`MOVE 'CITY-SET' TO #SET(A32)`
	`RELEASE SET #SET`

`RELEASE VARIABLES`	**RELEASE VARIABLES-Option:** 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 für RELEASE

```
** 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`:

```
        NAME                 CITY            DATE
                                               OF
                                             BIRTH
-------------------- -------------------- ----------

RUBIN                NEW YORK             1945-10-27
WALLACE              NEW YORK             1945-08-04
```

112 REPEAT

- Funktion REPEAT 910
- Syntax-Beschreibung REPEAT 910
- Beispiele REPEAT 911

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: FOR | ESCAPE

Gehört zur Funktionsgruppe: ***Schleifenverarbeitung***

Funktion REPEAT

Mit dem Statement REPEAT können Sie eine Verarbeitungsschleife initiieren.

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

Syntax-Beschreibung REPEAT

Zwei unterschiedliche Strukturen sind bei diesem Statement möglich:

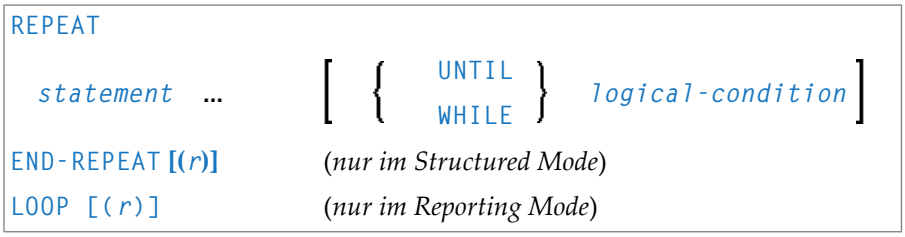
- **Syntax 1** - Statements werden ein- oder mehrmals ausgeführt
- **Syntax 2** - Statements werden überhaupt nicht oder mehrmals ausgeführt

Wann die Bedingung ausgewertet wird, ist abhängig davon, ob Sie sie an den Anfang oder das Ende der logischen Bedingung stellen.

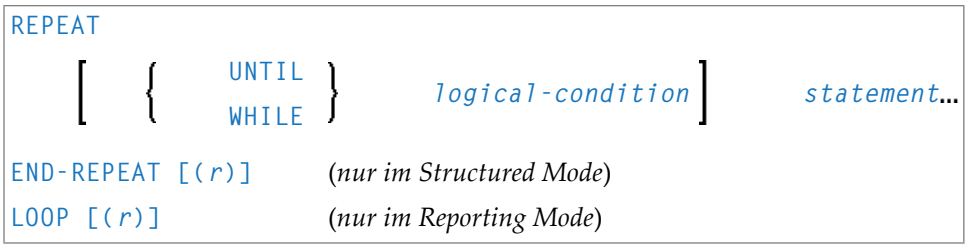
Weitere Informationen zu logischen Bedingungen, siehe den Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung*).

Eine Erläuterung der in den Syntax-Diagrammen benutzten Symbole siehe ***Syntax-Symbole***.

Syntax 1:



Syntax 2:



Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`UNTIL`	**UNTIL-Option:** Die Schleife wird so oft ausgeführt, bis die logische Bedingung erfüllt ist.
`WHILE`	**WHILE-Option:** Die Schleife wird solange ausgeführt, wie die logische Bedingung erfüllt ist.
`logical-condition`	**Logische Bedingung:** 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` (*r*) `LOOP` (*r*)	**Ende des REPEAT-Statements:** Im Structured Mode muss das für Natural reservierte Wort `END-REPEAT` zum Beenden des `REPEAT`-Statements benutzt werden. Im Structured Mode können Sie bei `END-REPEAT` Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement `LOOP` zum Beenden des `REPEAT`-Statements benutzt. Im Reporting Mode können Sie bei `LOOP` Labels oder Zeilennummern angeben.

Beispiele REPEAT

- Beispiel 1 - REPEAT-Statement

- Beispiel 2 - REPEAT-Statement mit Optionen WHILE und UNTIL

Beispiel 1 - REPEAT-Statement

```
** Example 'RPTEX1S': 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
  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 `RPTEX1S`:

```
ENTER A PERSONNEL NUMBER: 11500304
```

Nach Eingabe und Bestätigung der Personalnummer:

```
        NAME
--------------------

KLUGE
```

Äquivalentes Reporting-Mode-Beispiel: `RPTEX1R`.

Beispiel 2 - REPEAT-Statement mit Optionen WHILE und UNTIL

```
** Example 'RPTEX2S': 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 `RPTEX2S`:

```
#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: `RPTEX2R`.

113 REQUEST DOCUMENT

- Funktion REQUEST DOCUMENT 916
- Syntax-Beschreibung REQUEST DOCUMENT 917
- Automatisch erzeugte Header 923
- URL-Kodierung bei Sonderzeichen 924
- HTTP/HTTPS Status Codes: weitergeleitet und verweigert 926
- Beispiele REQUEST DOCUMENT 926

```
REQUEST DOCUMENT FROM url
[       WITH [with-clause]     ]
[       RETURN [return-clause] ]
RESPONSE http-response-code
[GIVING natural-error-number]
```

Dieser Abschnitt behandelt folgende Themen:

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

Verwandtes Statement: `PARSE JSON` und `PARSE XML`

Gehört zur Funktionsgruppe: ***Internet und XML***

Funktion REQUEST DOCUMENT

Mit dem Statement `REQUEST DOCUMENT` haben Sie die Möglichkeit, im Internet Dokumente abzurufen und hoch zu laden. Siehe auch *Statements für Internet-Zugang und Parsing* im *Leitfaden zur Programmierung*.

Informationen zur Unicode-Unterstützung siehe *Unicode- und Codepage-Unterstützung in der Natural-Programmiersprache*, Abschnitt *Natural-Statements*, Unterabschnitt *REQUEST DOCUMENT* in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Einschränkungen für Protokollarten

Aus technischen Gründen wird das HTTPS-Protokoll nur unter z/OS unterstützt.

Keine Unterstützung von Cookies

Cookies werden nicht unterstützt. Sie werden ignoriert.

Syntax-Beschreibung REQUEST DOCUMENT

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
url	C	S				A										ja	nein
http-response-code		S							I4							ja	ja
natural-error-number		S							I4							ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
url	**Adresse des Dokuments:** *url* ist der URL, der benutzt wird, um auf das Dokument zuzugreifen. Falls *url* eine Internet Protocol Version 6 (IPv6)-Adresse in hexadezimaler Notation ist, müssen Sie die Adresse in eckigen Klammern ([]) einschließen. Siehe auch ***Beispiel 7 - Get-Anforderung einer IPv6-Adresse an Port-Nummer 8080***. Es ist nicht möglich, eine Internet Protocol Version 4 (IPv6)-Adresse in eine IPv6-Adresse einzubetten.
with-clause	**WITH-Klausel:** Siehe *with-clause*.
return-clause	**RETURN-Klausel:** Siehe *return-clause*.
http-response-code	**RESPONSE:** *http-response-code* ist die HTTP/HTTPS Response Status Code-Nummer, die für den Request zurückgegeben wird, zum Beispiel: 200 (Request Completed/Anfrage ausgeführt). Siehe auch ***HTTP/HTTPS Status Codes weitergeleitet und verweigert***. Eine Liste der möglichen HTTP/HTTPS Status Codes finden Sie unter dem RFC 2616 Memorandum, das vom World Wide Web Consortium (W3C) publiziert wird.
natural-error-number	**GIVING-Option:** *natural-error-number* enthält die vierstellige Natural-Fehlernummer, wenn die Anfrage nicht ausgeführt werden konnte.

with-clause

```
[USER user-id]
[PASSWORD user-password]
[HEADER {[NAME] header-name-out [VALUE] header-value-out} ...]
[DATA {ALL outbound-document [ENCODED [[IN] CODEPAGE code-page-out]]
|{[NAME] variable-name-out [VALUE] variable-value-out} ...}]
```

Sie können die *with-clause* benutzen, um optional Benutzer/Passwort, Header und Einzelheiten zu den Daten für die Anfrage anzugeben.

Eine leere *with-clause* wird ignoriert (d.h., wenn nach WITH kein Wert angegeben wird).

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
user-id	C	S				A											ja	nein
user-password	C	S				A											ja	nein
header-name-out	C	S				A											ja	nein
header-value-out	C	S				A		N	P	I	F		D	T	L		ja	nein
outbound-document	C	S				A	U	N	P	I	F	B	D	T	L		ja	nein
code-page-out	C	S				A											ja	nein
variable-name-out	C	S				A											ja	nein
variable-value-out	C	S				A		N	P	I	F		D	T	L		ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
user-id	**Benutzerkennung:** *user-id* ist die Kennung des Benutzers, die für die Anfrage (Request) benutzt wird.
user-password	**Benutzer-Passwort:** *user-password* ist das Passwort des Benutzers, das für die Anfrage benutzt wird.
header-name-out *header-value-out*	**HEADER NAME/VALUE-Option:** *header-name-out* und *header-value-out* können nur gemeinsam verwendet werden: ■ *header-name-out* ist der Name einer mit dieser Anfrage versandten Header-Variablen.

Syntax-Element	Beschreibung
	■ *header-value-out* ist der Wert einer mit dieser Anfrage versandten Header-Variablen. *header-name-out*: Header-Namen dürfen keinen CR (Carriage Return), LF (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-value-out*: 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. Siehe auch ***Automatisch erzeugte Headers***.
outbound-document	**DATA ALL-Option:** *outbound-document* ist ein vollständiges Dokument, das versendet werden soll. Dieser Wert ist normalerweise erforderlich für die HTTP REQUEST-METHOD PUT (siehe ***Automatisch erzeugte Headers***).
code-page-out	**DATA ALL ENCODED-Option:** Bei der Datenübertragung mit dem REQUEST DOCUMENT-Statement erfolgt normalerweise keine Codepage-Konvertierung. Wenn Sie ausgehende Daten in einer bestimmten Codepage kodieren möchten, müssen Sie die CODEPAGE-Option benutzen: *outbound-document* wird von der Standard-Codepage (Wert der Systemvariablen *CODEPAGE) in die Codepage konvertiert, die in *code-page-out* angegeben wird. **Kodierung und Charset-Attribute** Falls das *outbound-document* eine Kodierung (XML) oder ein „Charset“ (HTML) enthält, wird empfohlen, dass der Wert der ENCODED-Option den Attribut-Wert des Dokuments abbildet. Beispiel: Wenn ein ausgehendes XML-Dokument die Attribut-Kodierung ="UTF-8" enthält, kodieren Sie das REQUEST DOCUMENT-Statement mit der Option DATA ALL #DOCUMENT ENCODED CODEPAGE 'UTF-8'.
variable-name-out *variable-value-out*	**DATA NAME/VALUE-Option:** *variable-name-out* und *variable-value-out* fordern nur spezifische DATA-Variablen-Informationen anstelle des vollständigen Dokuments an. Sie dürfen nur zusammen benutzt werden:

Syntax-Element	Beschreibung
	■ *data-variable-name* ist der Name der DATA-Variablen, die mit dieser Anfrage gesendet werden soll. ■ *data-variable-value* ist der Wert der DATA-Variablen, die mit dieser Anfrage gesendet werden soll. Dieser Wert wird für die HTTP REQUEST-METHOD POST benötigt (URL-Kodierung nötig, insbesondere das Et-Zeichen, auch kaufmännisches Und-Zeichen genannt, (&), Gleichheitszeichen (=), Prozentzeichen (%). **Einschränkung:** Wenn *data-variable-name* und *data-variable-value* gegeben sind und die Kommunikation standardmäßig http:// oder https:// ist, dann wird die REQUEST-METHOD POST (siehe ***Automatisch erzeugte Headers***) mit Content-Typ application/x-www-form-urlencoded verwendet. Während der Anfrage werden *data-variable-name* und *data-variable-value* durch Gleichheitszeichen (=) und Et-Zeichen (&) getrennt. Deshalb dürfen diese Operanden keine Gleichheitszeichen (=), Et-Zeichen (&) und, wegen der URL-Kodierung, keine Prozentzeichen (%) enthalten. Dieser werden als reserviert betrachtet und brauchen nicht, wie unter ***Reservierte Zeichen*** angegeben, kodiert zu werden.

return-clause

```
[HEADER [ALL header-all-in] [[NAME] header-name-in [VALUE] header-value-in ...]]
[PAGE inbound-document [ENCODED [[FOR TYPES mime-type ...] [IN] CODEPAGE code-page-in]]]
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
header-all-in		S				A											ja	ja
header-name-in	C	S				A											ja	nein
header-value-in		S	A *			A		N	P	I	F	B	D	T	L		ja	ja
inbound-document		S				A	U				B						ja	ja
mime-type	C	S				A											ja	nein
code-page-in	C	S				A											ja	nein

Mit der *return-clause* können Sie die Return-Informationen für die Headers und/oder das Dokument angeben.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
header-all-in	**HEADER ALL-Option:** *header-all-in* enthält alle mit der HTTP-Response angegebenen Header-Werte. 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.
header-name-in *header-value-in*	**HEADER NAME/VALUE-Option:** *header-name-in* und *header-value-in* geben nur header-spezifische Informationen zurück. Sie können nur gemeinsam verwendet werden: ■ *header-name-in* ist der Name eines mit dieser Anfrage erhaltenen Header. Aus Gründen der Kompatibilität mit dem Web Interface können Header-Namen mit Unterstrich (_) anstatt Bindestrich (-) geschrieben werden. Intern wird der Unterstrich durch einen Bindestrich ersetzt. Wenn *header-name-in* eine Leerzeichenkette ist, werden die Status-Informationen zurückgegeben: `HTTP/1.0 200 OK` ■ *header-value-in* ist der Skalar- oder Array-Wert, der erforderlich ist, um den mit dieser HTTP-Anfrage zurückgegebenen Header zu erhalten. Eine Array-Definition ist erforderlich, wenn mehr als eine Ausprägung des selben Header erwartet wird, zum Beispiel, bei mehreren Set-Cookie Headers. Nur eine Dimension eines mehrdimensionalen Array darf einen Indexbereich enthalten (siehe ***Beispiel 9***). Ein X-Array muss verwirklicht sein, bevor es benutzt werden kann. Wenn die Anzahl der Ausprägungen die Anzahl der Header übersteigt, werden die nicht benutzten Ausprägungen zurückgesetzt. Wenn die Anzahl der Header die Anzahl der Ausprägungen übersteigt, werden die übrig bleibenden Header ignoriert. Beispiel einer Array-Definition siehe ***Beispiel 9 - RETURN HEADER NAME VALUE mit Array-Definition***.
inbound-document	**PAGE ALL-Option:** *inbound-document* ist das für diese Anfrage zurückgegebene Dokument. Es erfolgt keinerlei Kodierung der zurückgegebenen Seite, d.h., die Seite bleibt so kodiert, wie sie vom HTTP-Server geliefert wird.
code-page-in	**ENCODED-Option:**

Syntax-Element	Beschreibung
	Bei der Datenübertragung mit dem `REQUEST DOCUMENT`-Statement erfolgt normalerweise keine Codepage-Konvertierung. Wenn Sie herein kommenden Daten in einer bestimmten Codepage kodieren möchten, können Sie die `ENCODED`-Option benutzen: Falls nötig, wird das *`inbound-document`* in der Standard-Codepage (Wert der Systemvariablen `*CODEPAGE`) der Natural-Session kodiert. Wenn der Wert von *`code-page-in`* leer ist, dann wird *`inbound-document`* von der Codepage, die mit dem Schlüsselwort-Subparameter `RDCP` definiert ist, in die Standard-Codepage (A/B) oder (U) kodiert. Der Schlüsselwort-Subparameter `RDCP` des Profilparameters `XML` wird benutzt, um den Namen der Standard-HTML/XML-Codepage anzugeben. **Anmerkung:** „Returned MIME type contains an encoding“ bedeutet, dass der HTTP-Server einen Content Type Header mit einer Klausel `charset=` zurückgibt, zum Beispiel: `charset=ISO-8859-1`. Beispiele für die Verwendung der Klausel `RETURN PAGE ENCODED` siehe unter ***Beispiel 8 - RETURN PAGE ENCODED-Klausel***.
`mime-type`	**FOR TYPES-Option:** Die als Antwort auf eine HTTP/HTTPS-Anfrage eintreffenden Daten können Binärdaten enthalten (z.B. Bilddaten/gif) oder Zeichendaten (z.B. Text/html). Zusammen mit der Antwort erhält das `REQUEST DOCUMENT`-Statement einen Parameter, der den Typ des Inhalts des angeforderten Dokuments (MIME-Typ, auch bekannt als Internet-Media-Typ) angibt. Dieser Parameter kann Informationen über die Codepage enthalten, in der das Dokument kodiert ist. *`mime-type`* ist die Liste der MIME-Typen, für die eine Kodierung des zurück gegebenen Dokuments *`inbound-document`* ausgeführt wird. Falls der zurückgegebene MIME-Typ eine Kodierung enthält, wird das *`inbound-document`* von dieser Codepage in die Standard-Codepage (A/B) oder (U) kodiert. Falls der zurückgegebene MIME-Typ *keine* Kodierung enthält, dann wird das *`inbound-document`* von der mit *`code-page-in`* definierten Codepage in die Standard-Codepage (Wert der Systemvariablen `*CODEPAGE`) (A/B) oder (U) kodiert. Falls der zurückgegebene MIME-Typ *keine* Kodierung enthält, dann erfolgt eine zusätzliche Prüfung, ob der zurückgegebene MIME-Typ mit einem der in *`mime-type`* gegebenen Typen übereinstimmt. Wird eine Übereinstimmung festgestellt, dann wird *`inbound-document`* von der mit *`code-page-in`* definierten Codepage in die Standard-Codepage (A/B) oder (U) kodiert.

Automatisch erzeugte Header

Für eine HTTP-Anfrage werden einige Headers benötigt, z.B. REQUEST-METHOD oder Content Type. Diese Headers werden in Abhängigkeit von den Parametern, die mit dem REQUEST DOCUMENT-Statement mitgegeben werden, automatisch erzeugt.

Anmerkung: Es ist möglich, die automatisch erzeugten Headers zu überschreiben. Natural prüft diese jedoch nicht auf Fehler. Es können unerwartete Fehler auftreten.

- HTTP REQUEST-METHOD
- Content-Typ

HTTP REQUEST-METHOD

Das REQUEST DOCUMENT-Statement unterstützt folgende REQUEST-METHODs: HEAD, POST, GET und PUT.

Die folgende Tabelle zeigt die HTTP REQUEST-METHOD, die in Abhängigkeit von den gegebenen Operanden erzeugt wird:

	WITH HEADER	WITH DATA	RETURN HEADER	RETURN PAGE
HEAD	o	-	x	-
POST	o	x	o	x
GET	o	-	o	x
PUT	o	DATA ALL*	o	o

Zusätzlich zu den oben aufgeführten Standard-REQUEST-METHODs können in einem REQUEST-METHOD Header die Methoden DELETE, PATCH, OPTIONS und TRACE angegeben werden.

Erläuterung:

o Optional. Operand kann optional angegeben werden.

- Operand kann nicht angegeben werden.

x Operand wird immer angegeben.

* Betrifft nur DATA ALL und nicht DATA NAME VALUE.

Content-Typ

Die `REQUEST-METHOD POST` erfordert einen Content Type Header für die HTTP-Anfrage. Wenn kein Content Type Header explizit angegeben wird, fügt Natural den folgenden Standard Content Type Header in die Anfrage ein:

```
application/x-www-form-urlencoded
```

URL-Kodierung bei Sonderzeichen

Wenn `POST`-Daten mit dem Content-Typ `application/x-www-form-urlencoded` gesendet werden, müssen bestimmte Zeichen mittels URL-Kodierung dargestellt werden, was bedeutet, dass das Zeichen ersetzt wird durch *`%hexadecimal-character-code`*. Nachfolgend sind einige grundsätzliche Informationen aufgeführt.

- Nicht-ASCII-Zeichen
- Nicht eindeutige Zeichen
- Reservierte Zeichen

Ausführliche Informationen, wann und warum eine URL-Kodierung notwendig ist, finden Sie in den Memoranden RFC 1630, RFC 1738 und RFC 1808, die vom World Wide Web Consortium (W3C) veröffentlicht werden.

Nicht-ASCII-Zeichen

Alle Nicht-ASCII-Zeichen (d.h., gültige ISO 8859/1-Zeichen, die nicht ebenfalls ASCII-Zeichen sind) müssen URL-kodiert werden. Beispielsweise erscheint die Datei `köln.html` in einer URL als `k%F6ln.html`.

Nicht eindeutige Zeichen

Um Server-Fehler zu vermeiden, sollten Sie nicht eindeutige Zeichen URL-kodiert angeben:

Nicht eindeutiges Zeichen	URL-Kodierung
Tabulatorzeichen	%09
Leerzeichen	%20
[	%5B
\	%5C
]	%5D
^	%5E
`	%60

Nicht eindeutiges Zeichen	URL-Kodierung
{	%7B
|	%7C
}	%7D
~	%7E

Reservierte Zeichen

Einige Zeichen haben spezielle Bedeutungen in URLs, 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 vereinfachte Erklärung, vollständige Informationen finden Sie in den RFCs).

Reservierte Zeichen sind:

Reserviertes Zeichen	URL-Kodierung
"	%22
#	%23
%	%25
&	%26
+	%2B
,	%2C
/	%2F
:	%3A
<	%3C
=	%3D
>	%3E
?	%3F
@	%40

HTTP/HTTPS Status Codes: weitergeleitet und verweigert

Eine Liste der HTTP/HTTPS Status Codes finden Sie im Memorandum RFC 2616, das vom World Wide Web Consortium (W3C) publiziert wird.

Bei HTTP/HTTPS-Status Codes für umgeleitete oder zurückgewiesene Anfragen gelten folgende Besonderheiten.

- Statuscodes 301 - 303 (redirected/weitergeleitet)
- Statuscode 401 (denied/verweigert, unauthorized/nicht authentifiziert)

Statuscodes 301 - 303 (redirected/weitergeleitet)

Die HTTP Status Codes 301, 302 oder 303 bedeuten, dass sich der URL, unter dem sich das Dokument befindet, geändert hat und dass die Anfrage deshalb an einen anderen URL per Umleitung weitergeleitet wurde. Als Statuscode wird der Rückgabe-Header mit dem Namen `LOCATION` (Adresse) angezeigt. Dieser Header enthält den URL, wohin die angeforderte Seite umgezogen ist. Es kann eine neue `REQUEST DOCUMENT`-Anfrage benutzt werden, um die umgezogene Seite zu suchen.

HTTP-Browser leiten automatisch zum neuen URL weiter, aber das Statement `REQUEST DOCUMENT` nimmt die Weiterleitung nicht automatisch vor.

Statuscode 401 (denied/verweigert, unauthorized/nicht authentifiziert)

Der HTTP Status Code 401 bedeutet, dass die angeforderte Seite nur aufgerufen werden kann, wenn mit der Anfrage eine gültige Benutzerkennung und ein gültiges Passwort angegeben werden. Als Rückmeldung wird der Rückgabe-Header mit dem Namen `WWW-AUTHENTICATE` mit dem für diese Anfrage erforderlichen REALM (Bereich) übermittelt.

HTTP-Browser zeigen üblicherweise einen Dialog mit Benutzerkennung und Passwort an, aber beim Statement `REQUEST DOCUMENT` wird kein Dialog angezeigt.

Beispiele REQUEST DOCUMENT

- Beispiel 1 - Allgemeiner Request
- Beispiel 2 - Einfacher GET-Request (keine Daten)
- Beispiel 3 - Einfacher HEAD-Request (keine zurückgelieferte Seite)
- Beispiel 4 - Einfacher POST-Request (Standard-REQUEST-METHOD)
- Beispiel 5 - Einfacher PUT-Request (mit DATA ALL)
- Beispiel 6 - REQUEST DOCUMENT mit Proxy-Autorisation
- Beispiel 7 - GET-Request einer IPv6-Adresse an Port-Nummer 8080

- Beispiel 8 - RETURN PAGE ENCODED-Klausel
- Beispiel 9 - RETURN HEADER NAME VALUE mit Array-Definition



Anmerkung: Es gibt einen Beispiel-Dialog `V5-RDOC` für dieses Statement in der Beispiel-Library `SYSEXV`.

Beispiel 1 - Allgemeiner Request

```
REQUEST DOCUMENT FROM "http://bolsap1:5555/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 - Einfacher GET-Request (keine Daten)

```
REQUEST DOCUMENT FROM "http://pcnatweb:8080"
  RETURN
    PAGE #Resultxml
RESPONSE #rc
```

Beispiel 3 - Einfacher HEAD-Request (keine zurückgelieferte Seite)

```
REQUEST DOCUMENT FROM "http://pcnatweb"
RESPONSE #rc
```

Beispiel 4 - Einfacher POST-Request (Standard-REQUEST-METHOD)

```
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 - Einfacher PUT-Request (mit DATA ALL)

```
REQUEST DOCUMENT FROM "http://pcnatweb/test.txt"
  WITH
    DATA ALL     #document
  RETURN
    PAGE #Resultxml
RESPONSE #rc
```

Beispiel 6 - REQUEST DOCUMENT mit Proxy-Autorisation

```
DEFINE DATA
LOCAL
1 #URL              (A) DYNAMIC
1 #PAGE             (A) DYNAMIC
1 #HTTPSTAT         (I4)
1 #PATH             (A) DYNAMIC
1 #NAME             (A) DYNAMIC
1 #VALUE            (A) DYNAMIC
1 #USERID           (A8)
1 #PASSWORD         (A8)
1 #AUTHHDR          (A) DYNAMIC
1 #CREDENTIALS      (A) DYNAMIC
1 #PADDCHAR         (A2/0:2) INIT  <' ','==','='>
1 #LENGTH           (I4)
1 #FACTOR           (I4)
1 #REMAINDER        (I4)
/*
/* #USR4210
1 PARM-FUNCTION  (A2) INIT <'BA'>
1 PARM-RC        (I4)
1 PARM-ERRTXT    (A72)
1 PARM-A         (A) DYNAMIC
1 PARM-B         (B) DYNAMIC
1 PARM-RFC       (B1)
END-DEFINE
**
ASSIGN #URL = 'http://www.softwareag.com/corporate/default.asp'
REQUEST DOCUMENT FROM #URL
  RETURN PAGE #PAGE ENCODED CODEPAGE ' '
  RESPONSE #HTTPSTAT
************************************************************
** HTTP response 407 means, that the proxy server requires **
** the client to identify itself!                          **
************************************************************
IF #HTTPSTAT = 407
  INPUT           'Please enter userid and password' (AD=I) /
  'Userid  :' #USERID / 'Password:' #PASSWORD
  ASSIGN #AUTHHDR = 'Proxy-Authorization'
  COMPRESS #USERID ':' #PASSWORD INTO PARM-B  LEAVING NO
```

```
************************************************************
** Convert credentials to ASCII                           **
************************************************************
  MOVE ENCODED PARM-B TO PARM-B
   CODEPAGE 'ASCII'
************************************************************
** ENCODE CREDENTIALS WITH BASE64                         **
************************************************************
** BASE64 demands that the length of the encoded string is **
** a multiple of 3. If encoding length does not match this **
** the encoded string has to be padded with '=' characters.**
** Since USR4210 does not support padding of the base64    **
** string with trailing '=' characters, we have to do this **
** our selfs.                                             **
************************************************************
  CALLNAT 'USR4210N' PARM-FUNCTION PARM-RC PARM-ERRTXT PARM-B PARM-A
    PARM-RFC
*
  #LENGTH := *LENGTH(PARM-A)
  WRITE PARM-A (AL=40) #LENGTH EJECT
  DIVIDE 3 INTO #LENGTH GIVING #FACTOR REMAINDER #REMAINDER
  COMPRESS  PARM-A #PADDCHAR(#REMAINDER)
    INTO #CREDENTIALS LEAVING NO
************************************************************
** Build Proxy-Authorization HTTP header                  **
************************************************************
  COMPRESS 'Basic '  #CREDENTIALS
    INTO #CREDENTIALS
  ASSIGN #AUTHHDR = 'Proxy-Authorization'
************************************************************
** Repeat HTTP request with valid client identification    **
************************************************************
  REQUEST DOCUMENT FROM #URL
   WITH HEADER NAME #AUTHHDR VALUE #CREDENTIALS
    RETURN PAGE #PAGE ENCODED CODEPAGE ' '
    RESPONSE #HTTPSTAT
END-IF
END
```

Beispiel 7 - GET-Request einer IPv6-Adresse an Port-Nummer 8080

```
REQUEST DOCUMENT FROM "http://[2BFC:5022:4081:0000::C:41]:8080"
  RETURN
    PAGE #Resultxml
RESPONSE #rc
```

Beispiel 8 - RETURN PAGE ENCODED-Klausel

1. Server gibt einen Header zurück: `'Content-type: text/html;charset=UTF-8'`

   Programmcode-Beispiel 1:

   ```
   ...
   RETURN PAGE inbound-document
   ```

   Resultierende Verarbeitung:

   inbound-document remains UTF-8 encoded.

   Programmcode-Beispiel 2:

   ```
   ...
   RETURN PAGE inbound-document ENCODED [..]
   ```

   Resultierende Verarbeitung:

   inbound-document wird unabhängig davon, ob *code-page-in* und *mime-type* angegeben werden, von UTF-8 in die Standard-Codepage umgesetzt. Wenn im Content Type Header eine gültige Kodierung zurückgeliefert wird, werden *mime-type* und *code-page-in* ignoriert.
2. Server liefert einen Header zurück: `'Content-type: text/xml'`

   Programmcode-Beispiel 1:

   ```
   ...
   RETURN PAGE inbound-document ENCODED
   ```

   Resultierende Verarbeitung:

   inbound-document wird nicht umgesetzt, weil der Content Type-Header keine gültige Kodierung enthält.

   Programmcode-Beispiel 2:

   ```
   ...
   RETURN PAGE inbound-document ENCODED FOR TYPES 'text/xml' IN CODEPAGE 'USASCII'
   ```

   Resultierende Verarbeitung:

   inbound-document wird von USASCII-Codepage in die Standard-Codepage umgesetzt. In diesem Fall erfolgt die Umsetzung gemäß der Annahme des Programmierers über die Kodierung der empfangenen Seite.

   Programmcode-Beispiel 3:

   ```
   ...
   RETURN PAGE inbound-document ENCODED FOR TYPES 'text/html'
       IN CODEPAGE '  '
   ```

Resultierende Verarbeitung:

inbound-document wird nicht umgesetzt, weil der in *mime-type* angegebene MIME-Typ `'text/html'` nicht mit dem im Content Type Header angegebenen Mime-Typ `'text/xml'` übereinstimmt.

Programmcode-Beispiel 4:

```
...
RETURN PAGE inbound-document ENCODED IN CODEPAGE ' '
```

Resultierende Verarbeitung:

inbound-document wird von der mit dem Subparameter `RDCP` des Profilparameters `XML` angegebenen Standard-Codepage in die Standard-Codepage umgesetzt.



Anmerkung: Der Standardwert für den `RDCP`-Subparameter, der gültig ist, wenn nichts anderes explizit angegeben wird, ist `ISO-8859-1`. Siehe auch *Statements PARSE XML und REQUEST DOCUMENT aktivieren/deaktivieren* in der *Parameter-Referenz*-Dokumentation.

Beispiel 9 - RETURN HEADER NAME VALUE mit Array-Definition

```
DEFINE DATA
LOCAL
1 #FROM      (A) DYNAMIC
1 #HEADER    (A) DYNAMIC
1 #PAGE      (A) DYNAMIC
1 #COOKIES (A20/1:3,1:4,2:5)
1 #RC      (I4)
END-DEFINE
ASSIGN #FROM = 'http://www.myserver.com'
REQUEST DOCUMENT FROM #FROM
   RETURN
        HEADER NAME 'Set-Cookie' VALUE #COOKIES(1,2:3,3)
        PAGE   #PAGE
        RESPONSE #RC
PRINT #COOKIES(*,*,*)
END
```

Im obigen Beispiel-Programm wären die folgenden Array-Definitionen (mit mehreren Dimensionen) *ungültig*:

```
RETURN HEADER NAME 'Set-Cookie' VALUE #COOKIES(1:3,2:3,3)
RETURN HEADER NAME 'Set-Cookie' VALUE #COOKIES(*,2,*)
```

114 RESET

- Funktion RESET 934
- Syntax-Beschreibung RESET 934
- Beispiel für RESET-Statement 936

```
RESET [INITIAL] operand1 ...
```

Dieser Abschnitt 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 RESET

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 RESET

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`RESET` *operand1*	**Zurücksetzen auf Nullwert:** `RESET` (ohne `INITIAL`) setzt jedes angegebene Feld (*operand1*) auf seinen **Standard-Ausgangswert**. Wenn *operand1* 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. Allgemeine Informationen über dynamische Variablen finden Sie in dem Abschnitt *Dynamische und große Variablen benutzen*.
`RESET INITIAL` *operand1*	**Zurücksetzen auf Ausgangswert:** Mit `RESET INITIAL` werden die angegebenen Felder (*operand1*) auf die für sie im `DEFINE DATA`-Statement definierten Ausgangswerte zurückgesetzt. ■ 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. ■ 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 für RESET-Statement

```
** 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
```

115 RESIZE

- Funktion RESIZE 940
- Syntax-Beschreibung RESIZE 940

```
RESIZE { dynamic-clause } [GIVING operand5]
       { array-clause   }
```

Dieser Abschnitt 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 RESIZE

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 RESIZE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1		S	A			A	U					B						nein	nein
operand2	C	S								I								nein	nein
operand3			A	G		A		N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand4	C	S						N	P	I									nein	nein
operand5		S								I4									nein	ja

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
dynamic-clause	**DYNAMIC-Klausel:** Mit dem Statement RESIZE DYNAMIC können Sie die Länge des aktuell zugewiesenen Speicherplatzes einer dynamischen Variable (*operand1*) an den mit *operand2* angegebenen Wert anpassen. Weitere Informationen, siehe ***DYNAMIC-Klausel*** weiter unten.
operand1	**Zu verändernde dynamische Variable:** *operand1* ist die dynamische Variable, für die die zugewiesene Länge angepasst werden soll.
operand2	**Neue Längenangabe:** *operand2* 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.
array-clause	**ARRAY-Klausel:** Mit dem Statement RESIZE ARRAY wird die Anzahl der Ausprägungen des X-Arrays (*operand3*) auf die mit (*dim*[,*dim*[,*dim*]]) angegebene Ober- und Untergrenze angepasst. Weitere Informationen siehe ***Array-Klausel*** weiter unten.
operand3	**X-Array-Name:** *operand3* 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 *operand4*	**X-Array-Ober- und Untergrenze:** 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 verwendet werden soll, kann ein Stern (*) anstatt *operand4* angegeben werden. Weitere Informationen, siehe ***Dimension*** weiter unten.
GIVING *operand5*	**GIVING-Klausel:** Wenn die GIVING-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die GIVING-Klausel angegeben wird, enthält *operand5* 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 der in der ***Array-Klausel*** angegebenen Dimensionen (*dim*) wird mittels der folgenden Syntax definiert:

```
{ operand4 } : { operand4 }
{ *        }   { *        }
```

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.

116 RETRY

- Funktion RETRY 946
- Einschränkung bei RETRY 946
- Beispiel für RETRY-Statement 946

RETRY

Dieser Abschnitt 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 RETRY

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 bei RETRY

Dieses Statement kann nur für den Zugriff auf Adabas-Datenbanken verwendet werden.

Beispiel für RETRY-Statement

```
** 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
```

117 ROLLBACK (SQL)

- Funktion ROLLBACK (SQL) 950
- Hinweis für Nicht-Natural-Programme 950
- Beispiel für ROLLBACK (SQL) 951

```
ROLLBACK
```

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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *ROLLBACK - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

Funktion ROLLBACK (SQL)

Das SQL-Statement `ROLLBACK` entspricht dem Natural-DML-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 `SYNCPOINT`-, `COMMIT`-, `END TRANSACTION`- oder `BACKOUT TRANSACTION`-Statement. Außerdem bewirkt `ROLLBACK`, dass alle während der Transaktion vorgehaltenen Datensätze freigegeben werden.

Wenn ein Programm versucht, Datenbankänderungen, die bereits durch eine Terminal-Ein-/Ausgabe 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 in einer anderen Standard-Programmiersprache geschriebenes externes 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 externe Programm absetzen.

Beispiel für ROLLBACK (SQL)

```
...
DELETE FROM SQL-PERSONNEL WHERE NAME = 'SMITH'
ROLLBACK
...
```

118 RUN

- Funktion RUN 954
- Syntax-Beschreibung RUN 954
- Dynamische Quellcode-Generierung und -Ausführung 955
- Beispiel für RUN-Statement 956

```
RUN [REPEAT] operand1 [ operand2 [(parameter)]] ... 40
```

Dieser Abschnitt 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 Subprogrammen***

Funktion RUN

Das RUN-Statement dient dazu, ein Natural-Source-Programm aus der Natural-Systemdatei zu lesen und es dann auszuführen.

Für Natural RPC: Siehe *Notes on Natural Statements on the Server* (in der *Natural RPC (Remote Procedure Call)*-Dokumentation).

Syntax-Beschreibung RUN

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
REPEAT	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. 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.
operand1	**Programmname:** Der Name des auszuführenden Programms (*operand1*) kann als alphanumerische Konstante oder als Inhalt einer alphanumerischen Variablen angegeben werden. Wird eine Variable verwendet, so muss sie 8 Stellen lang sein.

Syntax-Element	Beschreibung
	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. Das ausgeführte Programm wird in den Arbeitsbereich des *Programm-Editors* gelesen und überschreibt dabei den vorherigen Inhalt des Arbeitsbereichs.
operand2	**Parameter:** Mit dem RUN-Statement können Parameter an das Programm, das ausgeführt werden soll, übergeben werden. 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. 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. **Anmerkung:** Wenn *operand2* eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts übergeben, aber nicht die Datumskomponente.
parameter	Wenn *operand2* eine Datumsvariable ist, können Sie den Session- Parameter DF als *parameter* für diese Variable angeben.

Dynamische Quellcode-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 Quellcode-Generierung erfolgt, indem man Sourcetext in globalen Variablen unterbringt, und diese Variablen dadurch referenziert, dass man im Quellcode 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 Quellcode 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 für RUN-Statement

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                            ↩
```

XII

- 119 SELECT (SQL) 961
- 120 SEND METHOD 983
- 121 SEPARATE 995
- 122 SET CONTROL 1009
- 123 SET GLOBALS 1013
- 124 SET KEY 1017
- 125 SET TIME 1029
- 126 SET WINDOW 1033
- 127 SKIP 1037
- 128 SORT 1041
- 129 STACK 1053
- 130 STOP 1059

119 SELECT (SQL)

- Funktion SELECT (SQL) 962
- Syntax 1 - Cursor-orientierte Auswahl 962
- Syntax 2 - Nicht cursor-orientierte Auswahl 964
- Syntax-Element-Beschreibung SELECT (SQL) 964
- Verknüpfungsabfragen (Join Query) 981

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

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch `SELECT` (SQL) im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

Funktion SELECT (SQL)

Das Natural SQL-Statement `SELECT` unterstützt sowohl das **cursor-orientierte** `SELECT`, mit dem eine beliebige Anzahl von Zeilen gelesen werden kann, als auch das **nicht cursor-orientierte Singleton** `SELECT`, das maximal eine Zeile liest. Mit dem Konstrukt `SELECT ... END-SELECT` verwendet Natural die gleiche Datenbankschleifenverarbeitung wie beim Natural-DML-Statement `FIND`.

Zwei verschiedene Strukturen sind möglich:

- ***Syntax 1 - Cursor-orientierte Auswahl***
- ***Syntax 2 - Nicht cursor-orientierte Auswahl***

Syntax 1 - Cursor-orientierte Auswahl

Ebenso wie das Natural-DML-Statement `FIND` dient das cursor-orientierte `SELECT`-Statement dazu, auf der Basis eines Suchkriteriums einen Satz Zeilen (Datensätze) aus einer oder mehreren Db2-Tabellen auszuwählen. Da eine Datenbankverarbeitungschleife initiiert wird, muss die Schleife mit einem `LOOP`-Statement (im Reporting Mode) oder einem `END-SELECT`-Statement (im Structured Mode) abgeschlossen werden. Mit diesem Konstrukt verwendet Natural die gleiche Schleifenverarbeitung wie beim Natural-DML-Statement `FIND`.

Darüber hinaus braucht das Anwendungsprogramm keine Cursor-Verwaltung durchzuführen. Dies wird automatisch durch Natural erledigt.

- Syntax 1 - Common Set

- Syntax 1 - Extended Set

Syntax 1 - Common Set

```
SELECT selection into-clause table-expression
[ { UNION } [ DISTINCT ] [ (SELECT selection                    ] ]
  { EXCEPT }   [ ALL      ]   table-expression)                 ]
  { INTERSECT }               SELECT selection                  ]
                              table-expression                  ] ...
                    [ORDER BY criteria]
                    statement ...
{ END-SELECT [(r)] }
{ LOOP [(r)]       }
```

Syntax 1 - Extended Set

```
[WITH_CTE common-table-expression, ...]
SELECT selection into-clause table-expression
[ { UNION } [ DISTINCT ] [ (SELECT selection                    ] ]
  { EXCEPT }   [ ALL      ]   table-expression)                 ]
  { INTERSECT }               SELECT selection                  ]
                              table-expression                  ] ...
[ORDER BY criteria]
[ OPTIMIZE FOR integer { ROW  } ]
                       { ROWS }
[WITH isolation-level]
[SKIP LOCKED DATA]
[QUERYNO integer]
[OFFSET row-count]
[FETCH FIRST row-limit]
[WITH HOLD]
[WITH RETURN]
[WITH scroll-mode]
[WITH ROWSET POSITIONING FOR max-rowsets]
[IF NO RECORDS FOUND instruction]
statement ...
{ END-SELECT [(r)] }
{ LOOP [(r)]       }
```

Syntax 2 - Nicht cursor-orientierte Auswahl

Das SELECT SINGLE-Statement unterstützt die Funktionalität eines keine Cursor verwendenden Singleton Select, d.h., ein *select-expression*, der ohne Verwendung eines Cursors maximal eine Zeile liefert. Es kann nicht von einem Positioned UPDATE- oder einem DELETE-Statement referenziert werden.

- Syntax 2 - Common Set
- Syntax 2 - Extended Set

Syntax 2 - Common Set

```
SELECT SINGLE
selection into-clause table-expression
[IF NO RECORDS FOUND instruction]
statement ...
{ END-SELECT [(r)] }
{ LOOP [(r)]       }
```

Syntax 2 - Extended Set

```
SELECT SINGLE
selection into-clause table-expression
[WITH isolation-level]
[FETCH FIRST row-limit]
[IF NO RECORDS FOUND instruction]
statement ...
{ END-SELECT [(r)] }
{ LOOP [(r)]       }
```

Syntax-Element-Beschreibung SELECT (SQL)

Dieser Abschnitt beschreibt in alphabetischer Reihenfolge die Syntax-Elemente, die in den Syntax-Diagrammen unter ***Syntax 1 - Cursor-orientierte Auswahl*** und ***Syntax 2 - Nicht cursor-orientierte Auswahl*** dargestellt sind.

- END-SELECT | LOOP (r)
- OFFSET row-count

- FETCH FIRST row-limit
- IF NO RECORDS FOUND instruction
- INTO-Klausel (into-clause)
- OPTIMIZE FOR integer ROWS
- ORDER BY criteria
- QUERYNO-Klausel
- selection
- SKIP LOCKED DATA
- statement
- table-expression
- UNION | EXCEPT | INTERSECT-Klausel
- WITH_CTE common-table-expression
- WITH HOLD-Klausel
- WITH isolation-level
- WITH RETURN-Klausel
- WITH scroll-mode
- WITH ROWSET POSITIONING FOR max-rowsets

END-SELECT | LOOP (r)

Im Structured Mode muss das für Natural reservierte Schlüsselwort `END-SELECT` zum Beenden des `SELECT`-Statements benutzt werden.

Im Structured Mode können Sie bei `END-SELECT` Labels oder Zeilennummern angeben.

Im Reporting Mode muss das `LOOP`-Statement zum Beenden des `SELECT`-Statements benutzt werden.

Im Reporting Mode können Sie bei `LOOP` Labels oder Zeilennummern angeben.

OFFSET row-count

```
OFFSET [offset-row-count] { ROW  }
                          { ROWS }
```

Die `OFFSET`-Klausel gibt die Anzahl der Zeilen an, die in der Ergebnistabelle übersprungen werden sollen, bevor Zeilen von dort abgerufen werden. Eine begrenzte Anzahl von Zeilen am Ende einer Ergebnismenge (Result Set) kann bei Abfragen mit potenziell großen Ergebnismengen die Leistung verbessern.

offset-row-count ist eine numerische Variable oder Konstante, die die Anzahl der zu überspringenden Zeilen bestimmt. Die Zahl muss Null (`0`) oder eine positive Ganzzahl sein.

FETCH FIRST row-limit

```
FETCH FIRST [ 1 | integer ] { ROW | ROWS } ONLY
```

Die FETCH FIRST-Klausel begrenzt die Anzahl der mittels FETCH abzurufenden Zeilen. Es verbessert die Verarbeitungszeit von Abfragen mit möglicherweise großen Ergebnismengen, wenn nur eine beschränkte Anzahl von Zeilen erforderlich ist.

IF NO RECORDS FOUND instruction

Anmerkung: Diese Klausel ist tatsächlich 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 }
```

Mit der IF NO RECORDS FOUND-Klausel können Sie eine Schleifen-Verarbeitung einleiten, die ausgeführt werden soll, falls 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 unmittelbar ausgeführt, bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine Statements ausgeführt werden, muss die IF NO RECORDS FOUND-Klausel das Schlüsselwort ENTER enthalten

Anmerkung: Falls die Ergebnismenge des SELECT-Statements aus einer einzelnen Zeile mit NULL-Werten besteht, wird die IF NO RECORDS FOUND-Klausel nicht ausgeführt. Dies kann

der Fall sein, wenn die Auswahlliste nur aus einer der *aggregate-functions* SUM, AVG, MIN oder MAX bei Spalten besteht, und die Menge, mit dem diese *aggregate-functions* operieren, leer ist. Bei oben genannter 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 wird.

INTO-Klausel (into-clause)

```
INTO { parameter,...                                  }
     { VIEW {view-name [correlation-name ]},...       }
```

In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Auswahlergebnis gefüllt werden sollen.

Sie können in der INTO-Klausel entweder einzelne Parameter (*parameters*) oder eine oder mehrere im DEFINE DATA-Statement definierte Views (Datensichten) angeben.

Alle Zielfelder können entweder aus einer einzigen Tabelle kommen oder, bei einer Verknüpfungsoperation (vgl. Abschnitt *Verknüpfungsabfragen (Join Query)*), auch aus mehreren Tabellen.

Anmerkung: In der Standard-SQL-Syntax wird die INTO-Klausel nur bei Operationen ohne Cursor-Auswahl (Singleton SELECT) benutzt und kann nur dann angegeben werden, wenn eine einzelne Zeile ausgewählt werden soll. Dagegen wird in Natural die INTO-Klausel sowohl bei cursor-orientierten und nicht-cursor-orientierten Auswahloperationen verwendet.

Die Auswahl (*selection*) kann auch aus nur einem Stern (*) bestehen. In einem standardmäßigen *select-expression* steht dieser Stern 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 bei der Auswahl verwendet. Ihre Namen müssen vorhandenen Datenbank-Spaltennamen entsprechen.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
parameter	Wenn Sie einzelne Parameter als Zielfelder angeben, müssen diese in Anzahl und Format mit den in der entsprechenden Auswahl angegebenen Spalten (*columns*) bzw. Skalar-Ausdrücken (*scalar-expressions*) übereinstimmen, wie oben beschrieben. Siehe ***scalar-expressions*** und ***Beispiel 5***.
view-name	Der Name einer Natural-View (Datensicht), so wie im DEFINE DATA-Statement definiert. Wenn in der INTO-Klausel eine oder mehrere Views (Datensichten) referenziert werden, muss die Anzahl der in der Auswahl gemachten Angaben der Anzahl der in der/den Views definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt). **Anmerkung:** Die Natural-Zielfelder wie auch die Tabellenspalten müssen in einem Natural-DDM definiert sein (wobei ihre Namen allerdings unterschiedlich sein dürfen, da die Zuweisung entsprechend ihrer Reihenfolge erfolgt). Siehe ***Beispiel 5***.
correlation-name	Wenn die VIEW-Klausel in einem SELECT * verwendet wird, in dem mehrere Tabellen mit einem JOIN verknüpft werden, sind *correlation-names* erforderlich, falls die angegebene View (Datensicht) 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 *correlation-name* qualifiziert. Der einer View zugewiesene *correlation-name* muss einem der *correlation-names* entsprechen, mit denen die verknüpften Tabellen qualifiziert werden. Siehe auch ***Verknüpfungsabfragen (Join Query)*** und ***Beispiel 6***.

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
```

Diese Beispiele sind gleichbedeutend mit den folgenden Beispielen:

Beispiel 3:

```
...
SELECT NAME, AGE
  INTO NAME, AGE
```

Beispiel 4:

```
...
SELECT NAME, AGE
  INTO VIEW PERS
```

Beispiel 5:

```
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 einer Natural-View (Datensicht) sind, erhalten den Inhalt der Datenbankspalten `FIRSTNAME` und `AGE`.

Beispiel 6:

```
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
...
```

OPTIMIZE FOR integer ROWS

```
OPTIMIZE FOR integer ROWS
```

Die `OPTIMIZE FOR integer ROWS`-Klausel wird verwendet, um Db2 im Voraus über die Anzahl an Ganzzahl-Zeilen zu informieren, die von der Ergebnistabelle eingelesen werden sollen. Ohne diese Klausel geht Db2 davon aus, dass alle Zeilen der Ergebnistabelle eingelesen werden sollen und führt dementsprechend eine Optimierung durch.

Diese optionale Klausel ist nützlich, wenn Sie wissen, wie viele Zeilen wahrscheinlich ausgewählt werden, weil eine Optimierung von Ganzzahl-Zeilen die Verarbeitungszeit verbessern kann, wenn die Anzahl der tatsächlich ausgewählten Zeilen 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
```

ORDER BY criteria

```
ORDER BY { { sort-key [ ASC | DESC ] } , ... | INPUT SEQUENCE | ORDER OF table-designator }
```

Die `ORDER BY`-Klausel sortiert die Ergebnismenge eines `SELECT`-Statement in einer bestimmten Reihenfolge.

Die Ergebnismenge kann nach Sortierschlüssel (*sort-key*), nach Eingabe-Reihenfolge (`INPUT SEQUENCE`) oder entsprechend der mit dem Tabellenbezeichner (*table-designator*) angegebenen Tabelle sortiert werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
sort-key	Zur Angabe eines Sortierschlüssels haben Sie folgende Möglichkeiten: ■ Geben Sie eine Ganzzahl *n* an. ■ Geben Sie an, dass die Sortierung durch Sortieren der Werte der *n*ten Zeile der Ergebnismenge erfolgt, oder, durch Angabe eines Spaltennamens, dass sie durch Sortieren der Werte der gegeben Spalte erfolgt. ■ Geben Sie einen Skalar-Ausdruck an, wobei die Sortierung durch Sortieren der Werte des Ausdrucks erfolgt.

Syntax-Element	Beschreibung
	Der Ausdruck kann aus Spalten der *host-variables* und Konstanten der Ergebnismenge bestehen. Falls mehrere Sortierschlüssel existieren, werden die Zeilen nach dem ersten Sortierschlüssel sortiert; doppelte, erste Sortierschlüssel werden nach dem zweiten Sortierschlüssel sortiert usw. Wird im Sortierschlüssel eines FULLSELECT, der einen SET-Operator (UNION, EXCEPT, INTERSECTION) enthält, ein Spaltenname angegeben, muss er nicht-qualifiziert sein.
ASC|DESC	Die in der ORDER BY-Klausel angegebene Reihenfolge kann entweder aufsteigend (ASC) oder absteigend (DESC) sein. Der Standardwert ist ASC. Siehe auch ***Beispiel 2*** weiter unten.
INPUT-SEQUENCE	Gibt an, dass die Ergebnistabelle die Eingabe-Reihenfolge der Zeilen abbildet, die in der VALUES-Klausel eines INSERT-Statements angegeben sind.
ORDER OF *table-designator*	Gibt an, dass die Zeilen der Ergebnistabelle genauso wie bei der mit dem Tabellenbezeichner (*table-designator*) bezeichneten Tabelle sortiert werden sollen. Der *table-designator* muss außerdem in der FROM-Klausel angegeben werden.

Beispiele

Beispiel 1:

```
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
  ...
```

Beispiel 2:

```
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
  ...
```

QUERYNO-Klausel

```
QUERYNO integer
```

Die QUERYNO-Klausel gibt die für dieses SQL-Statement zu benutzende Zahl bei der EXPLAIN-Ausgabe und der Ablaufverfolgung von Datensätzen an. Die Zahl wird als QUERYNO-Spalte in der PLAN_TABLE für die Zeilen benutzt, die Informationen zu diesem Statement enthalten.

selection

Siehe ***Selection*** in *Select Expressions*.

SKIP LOCKED DATA

```
SKIP LOCKED DATA
```

Die SKIP LOCKED DATA-Klausel gibt an, dass Zeilen übersprungen werden, wenn auf der betreffenden Zeile inkompatible Sperren durch andere Transaktionen vorliegen.

statement

Das (oder die) Natural-Statement(s), die während der Verarbeitungsschleife ausgeführt werden sollen.

table-expression

Siehe *table-expression* in *Select Expressions*.

UNION | EXCEPT | INTERSECT-Klausel

```
{ UNION     } [ DISTINCT ] [ (SELECT selection table-expression) ]
{ EXCEPT    } [ ALL      ] [ SELECT selection table-expression   ] ...
{ INTERSECT }
```

UNION, EXCEPT und INTERSECT leiten eine Abfrage ein, bei der SET-Operationen beteiligt sind.

SET-Operationen vereinigen die Ergebnisse von zwei oder mehr *select-expressions* miteinander. Die in den einzelnen *select-expressions* angegebenen Spalten müssen SET-operationskompatibel sein, d.h. in Anzahl, Typ und Format zueinander passen.

Nur der erste *select-expression* darf eine INTO-Klausel enthalten.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
UNION	Vereinigt die Ergebnisse von zwei oder mehr *select-expressions* miteinander.
EXCEPT	Gibt die Differenzmenge der Ergebnismengen zweier *select-expressions* an.
INTERSECT	Gibt die Schnittmenge zweier Ergebnismengen an.
DISTINCT	Gibt an, dass die Ergebnismenge keine redundanten (doppelten) Zeilen enthält. Dies ist die Standardeinstellung.
ALL	Gibt an, dass die Ergebnismenge redundante (doppelte) Zeilen enthält. Wenn die SET-Operation nicht ausdrücklich ALL enthält, werden redundante (doppelte) Zeilen aus dem Ergebnis entfernt.

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
...
```

WITH_CTE common-table-expression

```
WITH_CTE common-table-expression-name [(column-name,...)] AS (fullselect)
```

Mit dieser Klausel können Sie eine Ergebnistabelle definieren, die in einer beliebigen FROM-Klausel eines nachfolgenden SELECT-Statements referenziert werden kann.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
WITH_CTE	Das Natural-spezifische Schlüsselwort WITH_CTE entspricht dem SQL-Schlüsselwort WITH. WITH_CTE wird durch den Natural Compiler in das SQL-Schlüsselwort WITH umgesetzt.
common-table-expression-name	Muss eine nicht qualifizierte SQL-Kennung sein und muss sich von anderen, im selben Statement angegebenen *common-table-expression* unterscheiden. Nach dem Schlüsselwort WITH_CTE können mehrere *common-table-expressions* angegeben werden. Jeder dieser Ausdrücke kann in der FROM-Klausel eines *common-table-expression-name* des nachfolgenden SELECT-Statements referenziert werden.
column-name	Muss eine nicht qualifizierte SQL-Kennung sein und muss innerhalb eines *common-table-expression-name* eindeutig sein.
AS (*fullselect*)	Die Anzahl der *column-names* muss gleich der Anzahl der Spalten des *fullselect* sein.

Ein *common-table-expression* kann in folgenden Fällen benutzt werden

- anstelle einer View (Datensicht), um das Erstellen einer View zu vermeiden,
- wenn dieselbe Ergebnistabelle gemeinsam in einem *fullselect* verwendet werden muss,
- wenn die Ergebnistabelle 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
```

WITH HOLD-Klausel

```
WITH HOLD
```

Die `WITH HOLD`-Klausel wird benutzt, um zu verhindern, dass der Cursor durch eine `COMMIT`-Operation innerhalb von Datenbankschleifen geschlossen wird. Wenn `WITH HOLD` angegeben wird, schreibt eine `COMMIT`-Operation alle Änderungen der aktuellen logischen Arbeitseinheit weg, gibt aber nur die Sperren frei, die nicht zur Verwaltung des Cursors erforderlich sind. Diese optionale Klausel ist hauptsächlich nützlich bei der Stapelverarbeitung, sie wird ignoriert im pseudo-konversationalen CICS-Modus und in nachrichtengesteuerten IMS-Programmen.

Beispiel

```
SELECT name INTO #name FROM table
WHERE AGE = 2 WITH HOLD
```

WITH isolation-level

```
WITH { CS
       RR
       RR KEEP UPDATE LOCK
       RS
       RS KEEP UPDATE LOCKS
       UR }
```

Mit dieser optionalen Klausel können Sie eine explizite Isolationsstufe angeben, mit der das Statement ausgeführt werden soll.

Folgende Optionen stehen zur Verfügung:

Option	Bedeutung
`CS`	Cursorstabilität
`RR`	Wiederholbares Lesen
`RR KEEP UPDATE LOCKS`	Wiederholbarer Lesevorgang unter Beibehaltung von Aktualisierungssperren. Gilt nur bei ***Syntax 1 - Extended Set*** und nur, wenn ein **positioniertes** `UPDATE`-Statement oder ein **positioniertes** `DELETE`-Statement mit dem `SELECT`-Statement verarbeitet wird.
`RS`	Lesestabilität
`RS KEEP UPDATE LOCKS`	Lesestabilität unter Beibehaltung von Aktualisierungssperren. Gilt nur bei ***Syntax 1 - Extended Set*** und nur, wenn ein **positioniertes** `UPDATE`-Statement oder ein **positioniertes** `DELETE`-Statement mit dem `SELECT`-Statement verarbeitet wird.

Option	Bedeutung
UR	Nicht festgeschriebener Lesevorgang UR kann nur bei einem SELECT-Statement angegeben werden, und wenn es für die Tabelle nur eine Lesezugriffsberechtigung gibt. Die standardmäßige Isolationsstufe wird durch Isolation des Pakets oder Plans festgelegt, worindas Statement eingebunden ist. Die standardmäßige Isolationsstufe ist außerdem abhängig davon, ob für die Ergebnistabelle nur eine Lesezugriffsberechtigung besteht oder nicht. Um die standardmäßige Isolationsstufe zu ermitteln, greifen Sie auf die IBM-Literatur zurück.

WITH RETURN-Klausel

```
WITH RETURN
```

Die WITH RETURN-Klausel dient zum Erstellen von Ergebnismengen. Deshalb gilt diese Klausel nur für Programme, die als eine Natural Stored Procedure eingesetzt werden. Wenn die WITH RETURN-Klausel in einem SELECT-Statement angegeben wird, bleibt der zugrunde liegende Cursor offen, wenn die zugehörige Verarbeitungsschleife verlassen wird, außer wenn die Verarbeitungsschleife alle Zeilen der Ergebnismenge selbst gelesen hatte. Während der ersten Ausführung der Verarbeitungsschleife wird nur der Cursor geöffnet. Die erste Zeile wird noch nicht abgerufen. Dadurch wird es dem Natural-Programm ermöglicht, eine vollständige Ergebnismenge an den Aufrufer der Stored Procedure zurückzugeben. Es bleibt dem Natural-Programmierer überlassen, wie viele Zeilen der Natural Stored Procedure abgearbeitet werden und wie viele nicht abgearbeitete Zeilen der Ergebnismenge an den Aufrufer der Stored Procedure zurückgegeben werden. Wenn Sie Zeilen der WITH RETURN-Operation in der Natural Stored Procedure verarbeiten möchten, müssen Sie folgende Codezeilen eingeben, um zu verhindern, dass die „erste leere“ Zeile in der Verarbeitungsschleife verarbeitet wird.

```
IF *counter =1 ESCAPE TOP END-IF
```

Wenn Sie sich entschließen, die Verarbeitung der Zeilen zu beenden, müssen Sie folgende Codezeilen in der Verarbeitungsschleife eingeben:

```
IF condition ESCAPE BOTTOM END-IF
```

Wenn das Programm alle Zeilen der Ergebnismenge liest, wird der Cursor geschlossen, und es wird keine Ergebnismenge für dieses SELECT WITH RETURN an den Aufrufer der Stored Procedure zurückgegeben.

Die folgenden Programme sind Beispiele zum Einlesen vollständiger Ergebnismengen (*Beispiel 1*) und unvollständiger Ergebnismengen (*Beispiel 2*).

Beispiele

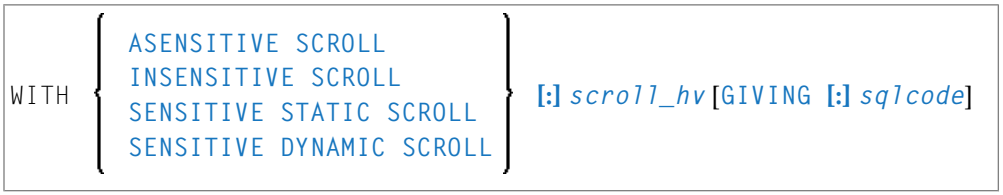
Beispiel 1:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Return all rows of the result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH RETURN
ESCAPE BOTTOM
END-SELECT
END
```

Beispiel 2:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Read the first two rows and return the rest as result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH   RETURN
WRITE PROCEDURE *COUNTER
IF *COUNTER = 1 ESCAPE TOP END-IF
IF *COUNTER = 3 ESCAPE BOTTOM END-IF
END-SELECT
END
```

WITH scroll-mode



Natural for Db2 unterstützt verschiebbare Cursor von Db2 über die Klauseln `WITH ASENSITIVE SCROLL`, `WITH SENSITIVE STATIC SCROLL` und `SENSITIVE DYNAMIC SCROLL`.

Verschiebbare Cursor ermöglichen es Natural for Db2-Anwendungen, eine Zeile in einer Ergebnismenge beliebig zu positionieren. Mit nicht verschiebbaren Cursor können die Daten nur sequenziell vom oben nach unten gelesen werden.

Verschiebbare Cursor ermöglichen es der Anwendungen, eine beliebige Zeile in einem Cursor zu positionieren, solange der Cursor offen ist.

Die Positionierung erfolgt in Abhängigkeit vom *scroll_hv*-Inhalt. Der Inhalt wird jedes Mal ausgewertet, wenn eine FETCH-Operation gegen Db2 ausgeführt wird.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ASENSITIVE SCROLL	Gibt an, dass der Cursor entweder INSENSITIVE oder SENSITIVE DYNAMIC ist. Dies wird von Db2 beim Öffnen des Cursor in Abhängigkeit von READ-ONLY-Eigenschaft des Cursor bestimmt: Wenn über den Cursor nur gelesen werden kann (d.h. READ-ONLY), wird er INSENSITIVE, wenn der Cursor nicht READ-ONLY ist, wird er SENSITIVE DYNAMIC.
INSENSITIVE SCROLL	Gibt an, dass der Cursor INSENSITIVE ist bei Aktualisierungen, Löschungen und Einfügungen, die gegen die Basistabelle ausgeführt werden. INSENSITIVE SCROLL bezieht sich auf einen Cursor, der nicht in **Positioned** UPDATE- oder **Positioned** DELETE-Operationen benutzt werden kann. Außerdem gibt ein INSENSITIVE SCROLL-Cursor keine UPDATE-, DELETE- oder INSERT-Operationen gegen die Basistabelle wieder, nachdem der Cursor geöffnet wurde. Siehe auch **Anmerkung**.
SENSITIVE STATIC SCROLL	Gibt an, dass der Cursor SENSITIVE ist bei Aktualisierungen und Löschungen, die gegen die Basistabelle ausgeführt werden, jedoch gegen Einfügungen, nachdem der Cursor geöffnet worden ist. SENSITIVE STATIC SCROLL bezieht sich auf einen Cursor, der für **Positioned** UPDATE- oder **Positioned** DELETE-Operationen benutzt werden kann. Außerdem gibt ein SENSITIVE STATIC SCROLL-Cursor UPDATE und DELETE-Operationen an Zeilen der Basistabelle wieder. Der Cursor gibt keine INSERT-Operationen wieder. Siehe auch **Anmerkung**.
SENSITIVE DYNAMIC SCROLL	SENSITIVE DYNAMIC gibr an, dass der Cursor SENSITIVE ist bei Aktualisierungen, Löschungen und Einfügungen, die gegen die Basistabelle ausgeführt werden, nachdem der Cursor geöffnet worden ist. SENSITIVE DYNAMIC verschiebbare Cursor geben UPDATE-, DELETE- und INSERT-Operationen gegen die Basistabelle wieder, während der Cursor geöffnet ist.

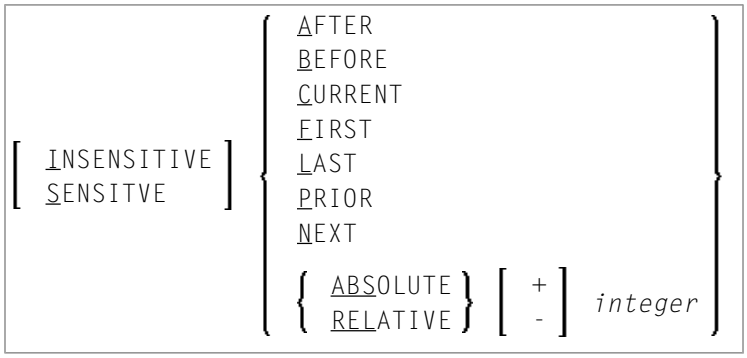
Anmerkung: INSENSITIVE und SENSITIVE STATIC verschiebbare Cursor verwenden temporäre Ergebnistabellen und benötigen eine TEMP-Datenbank in Db2 (siehe entsprechende IBM Db2 Literatur).

scroll_hv

Die Variable *scroll_hv* muss alphanumerisch sein.

Die Variable *scroll_hv* gibt an, welche Zeile der Ergebnistabelle während einer Ausführung der Datenbank-Verarbeitungsschleife abgerufen wird. Außerdem gibt sie während einer FETCH-Operation die Sensitivität von UPDATE- oder DELETE-Operationen gegen die Zeile der

Basistabelle an. Der Inhalt der Variablen *scroll_hv* wird jedesmal ausgewertet, wenn der Datenbank-Verarbeitungsschleifenzyklus ausgeführt wird.



***scroll_hv*-Sensitivitätsangabe**

Die Sensitivitätsangabe `INSENSITIVE` oder `SENSITIVE` ist optional.

- Wenn sie bei einem `FETCH` gegen einen `INSENSITIVE SCROLL`-Cursor weggelassen wird, gilt der Standardwert `INSENSITIVE`.
- Wenn sie bei einem `FETCH` gegen einen `SENSITIVE STATIC/DYNAMIC SCROLL`-Cursor weggelassen wird, gilt der Standardwert `SENSITIVE`.

Die Sensitivität gibt an, ob die Zeilen in der Basistabelle überprüft werden oder nicht, wenn eine `FETCH`-Operation für einen verschiebbaren Cursor ausgeführt wird.

- Wenn die entsprechende Basistabellen-Spalte für die `WHERE`-Klausel qualifiziert ist und nicht gelöscht wurde, gibt ein `SENSITIVE FETCH` die Zeile der Basistabelle zurück.
- Wenn die entsprechende Basistabellen-Spalte nicht für die `WHERE`-Klausel qualifiziert ist oder nicht gelöscht wurde, gibt `SENSITIVE FETCH` einen Status „freier Datenbereich durch Aktualisieren“ (UPDATE-Hole) oder „freier Datenbereich durch Löschen“ (DELETE-Hole), d.h. `SQLCODE +222`, zurück.

Ein `INSENSITIVE FETCH` überprüft nicht die betreffende Basistabellen-Spalte.

***scroll_hv*-Optionen**

Im Folgenden werden die verfügbaren Optionen erläutert, um die abzurufende(n) Zeile(n), die Position, von der aus der `FETCH` gestartet werden soll und/oder die Richtung, in der geblättert werden soll, festzulegen:

Option	Erläuterung
`AFTER`	Positioniert nach der letzten Zeile. Es wird keine Zeile abgerufen.
`BEFORE`	Positioniert vor der ersten Zeile. Es wird keine Zeile abgerufen.
`CURRENT`	Ruft die aktuelle Zeile (erneut) ab.
`FIRST`	Ruft die erste Zeile ab.
`LAST`	Ruft die letzte Zeile ab.
`NEXT`	Ruft die Zeile nach der aktuellen Zeile ab. Dies ist der Standardwert.

Option	Erläuterung
PRIOR	Ruft die Zeile vor der aktuellen Zeile ab.
+|-*integer*	Gilt nur in Verbindung mit ABSOLUTE oder RELATIVE. Gibt die Position der abzurufenden Zeile absolut (ABSOLUTE) oder relativ (RELATIVE) an. Geben Sie ein Plus- (+) oder Minus-Zeichen (-) und danach eine Ganzzahl ein. Der Standardwert ist ein Plus-Zeichen (+).
ABSOLUTE	Gilt nur in Verbindung mit +|-*integer*. Benutzt *integer* als die absolute Position innerhalb des Resultset (Ergebnismenge), von dem aus die Zeile abgerufen wird. Weitere Einzelheiten hinsichtlich positiver und negativer Positionszahlen entnehmen Sie der *Db2 SQL Reference*-Dokumentation von IBM.
RELATIVE	Gilt nur in Verbindung mit +|-*integer*. Benutzt *integer* als Position relativ zur aktuellen Position innerhalb des Resultset (Ergebnismenge), von dem aus die Zeile abgerufen wird. Weitere Einzelheiten hinsichtlich positiver und negativer Positionszahlen entnehmen Sie der *Db2 SQL Reference*-Dokumentation von IBM.

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 vom Db2 SQLCODE der zugrunde liegenden FETCH-Operation zurückgegeben. Dadurch wird es der Anwendung ermöglicht, auf verschiedene, bei geöffnetem verschiebbarem Cursor vorgefundene Status-Codes zu reagieren. Die wichtigsten, von SQLCODE angezeigten Status-Codes sind in der folgenden Tabelle aufgeführt:

SQLCODE	Erläuterung
0	FETCH-Operation erfolgreich, Daten zurückgegeben, außer bei einem FETCH mit der Option BEFORE oder AFTER.
+100	Zeile nicht gefunden, Cursor noch geöffnet, keine Daten zurückgegeben.
+222	Freier Datenbereich durch Aktualisieren (UPDATE-Hole) oder freier Datenbereich durch Löschen (DELETE-Hole), Cursor noch geöffnet, keine Daten zurückgegeben. Die entsprechende Zeile der Basistabelle wurde aktualisiert oder gelöscht, so dass die Zeile nicht mehr für die WHERE-Klausel qualifiziert ist.
+231	FETCH-Operation mit der Option CURRENT, wobei der Cursor aber nicht auf einer Zeile positioniert ist, keine Daten zurückgegeben. Diese Situation entsteht, wenn das vorangegangene FETCH den SQLCODE +100 zurückgegeben hat.

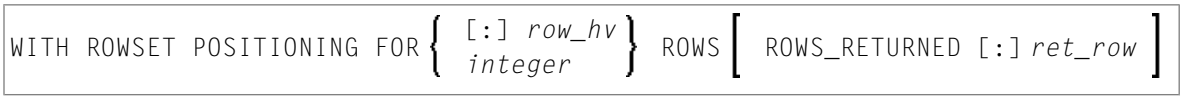
Wenn Sie GIVING [:] *sqlcode* angeben, muss die Anwendung auf die verschiedenen Status-Codes reagieren. Wenn ein SQLCODE +100 fünfmal hintereinander ohne Terminal-Ein-/Ausgabe

auftritt, gibt die Natural für Db2-Laufzeitumgebung 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, erzeugt, außer bei `SQLCODE 0` und `SQLCODE +100`, jeder `SQLCODE` den Natural-Fehler NAT3700, und die Verarbeitungsschleife wird beendet. Ein `SQLCODE +100` (Zeile nicht gefunden) beendet die Verarbeitungsschleife.

Siehe auch Beispielprogramm `DEM2SCRL` in der Natural-Systembibliothek `SYSDB2`.

WITH ROWSET POSITIONING FOR max-rowsets



Diese Klausel ermöglich eine Verarbeitung von Db2-Zeilengruppen, die der Multifetch-Verarbeitung in der Natural-DML entspricht. `[:] row_hv (I4)` oder *integer* bestimmt die maximale Anzahl Zeilen, die von Db2 an Natural zurückgegeben werden. Die Zahl bestimmt entweder die Größe des Natural Multi-Fetch-Puffers, der für die Standardverarbeitung von mehreren Zeilen benutzt wird, oder die maximale Anzahl Zeilen, die von Db2 an das Natural-Programm für die erweiterte Verarbeitung mehrerer Zeilen zurückgegeben wird.

ROWS_RETURNED [:] *ret_row*-Klausel:

Mit dieser Klausel wir eine Variable im Format I4 angegeben, die die Anzahl der Zeilen empfängt, die von Db2 aufgrund der zuletzt ausgeführten Db2-Abrufoperation für die erweiterte Verarbeitung mehrerer Zeilen zurückgegeben wird.

Verknüpfungsabfragen (Join Query)

Eine Verknüpfungsabfrage (Join Query) ist eine Abfrage, bei der Daten aus mehr als einer Tabelle gewonnen werden. Alle beteiligten Tabellen müssen in der `FROM`-Klausel angegeben werden.

Eine Verknüpfungsabfrage bildet immer das kartesische Produkt der in der `FROM`-Klausel aufgelisteten Tabellen und entfernt später aus dieser kartesischen Produkttabelle alle Zeilen, die nicht die Verknüpfungsbedingung erfüllen, die in der `WHERE`-Klausel angegeben ist.

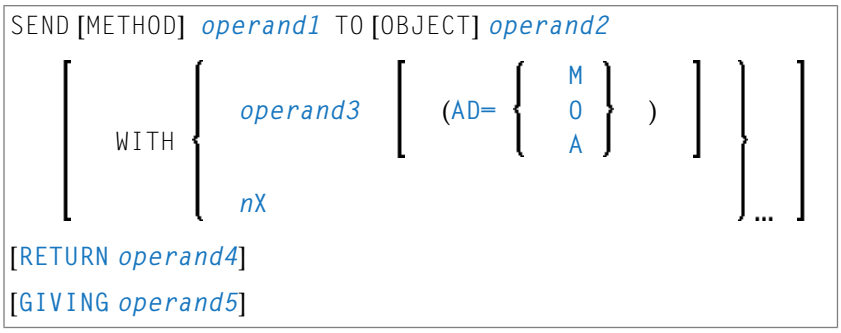
Falls die Tabellennamen ziemlich lang sind, können Sie Korrelationsnamen (*correlation-name*) verwenden, um sich Schreibarbeit zu ersparen. Korrelationsnamen müssen benutzt werden, wenn eine in der Auswahlliste angegebene Spalte in mehr als einer der zu verknüpfenden Tabellen existiert, damit klar ist, welche der namensgleichen Spalten ausgewählt werden sollen.

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
    ...
```

120 SEND METHOD

- Funktion SEND METHOD 984
- Syntax-Beschreibung SEND METHOD 984
- Beispiel für SEND METHOD-Statement 987



Dieser Abschnitt 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 SEND METHOD

Das `SEND METHOD`-Statement dient dazu, eine bestimmte Method eines Objekts aufzurufen. Informationen zur komponentenbasierten Programmierung, siehe *NaturalX* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung SEND METHOD

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A												ja	nein
operand2		S															O	nein	nein
operand3	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand4		S	A			A	U	N	P	I	F	B	D	T	L	C	G	O	ja	nein
operand5		S			N					I									ja	nein

Format C und G kann nur an Methods lokaler Klassen übergeben werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung

operand1	**Method-Name:**
	
	operand1 ist der Name einer Method, die vom in *operand2* angegebenen Objekt unterstützt wird.
	
	Da die Methoden-Namen in unterschiedlichen Interfaces einer Klasse identisch sein können, kann der Methoden-Name in *operand1* auch mit dem Interface-Namen versehen werden, um Mehrdeutigkeiten zu vermeiden.
	
	Im folgenden Beispiel hat das Objekt #O3 ein Interface Iterate mit der Method Start. Es gelten die folgenden Statements:
	
	`* Specifying only the method name.`
	`SEND 'Start' TO #O3`
	`* Qualifying the method name with the interface name.`
	`SEND 'Iterate.Start' TO #O3`
	
	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.

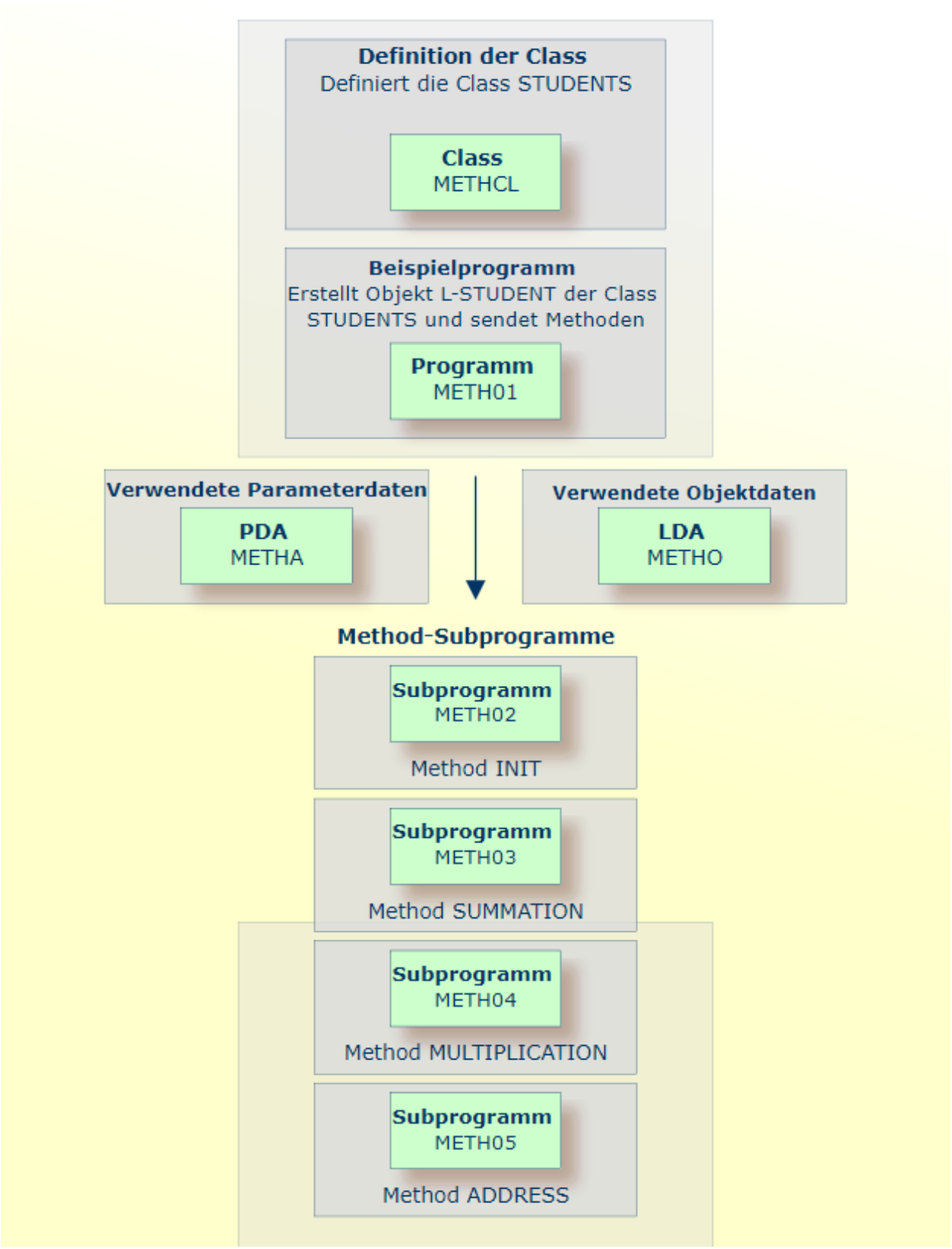
operand2	**Object-Handle:** Die Handle des Objekts, an das der Aufruf der Method gesendet werden soll. *operand2* muss als Objekt-Handle (HANDLE OF OBJECT) definiert werden. Das Objekt muss bereits vorhanden sein. Um eine Method des aktuellen Objekts innerhalb einer Method aufzurufen, verwenden Sie die Systemvariable *THIS-OBJECT.
operand3	**Methodenspezifische Parameter:** Als *operand3* können Sie Parameter angeben, die methodenspezifisch sind. Im folgenden Beispiel hat das Objekt #O3 die Method PositionTo mit dem Parameter Pos. Die Method wird wie folgt aufgerufen: `SEND 'PositionTo' TO #O3 WITH Pos` Methods können optionale Parameter haben. Optionale Parameter brauchen nicht angegeben zu werden, wenn die Method aufgerufen wird. Um einen optionalen Parameter wegzulassen, verwenden Sie den Platzhalter 1X. Um *n* optionale Parameter wegzulassen, verwenden Sie den Platzhalter *n*X. Im folgenden Beispiel hat die Method SetAddress des Objekts #O4 die Parameter FirstName (Vorname), MiddleInitial (Mittlere Initiale), LastName (Nachname), Street (Straße) und City (Stadt), wobei MiddleInitial, Street und City optional sind. Es gelten die folgenden Statements:

Syntax-Element	Beschreibung
	``` * Specifying all parameters. SEND 'SetAddress' TO #O4 WITH FirstName MiddleInitial LastName Street ↩ City * Omitting one optional parameter. SEND 'SetAddress' TO #O4 WITH FirstName 1X LastName Street City * Omitting all optional parameters. SEND 'SetAddress' TO #O4 WITH FirstName 1X LastName 2X ``` Wenn ein Pflichtparameter weggelassen wird, führt dies zu einem Laufzeitfehler.
AD=	**Attribut-Definition:** Wenn *operand3* eine Variable ist, können Sie sie wie folgt kennzeichnen: 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. Wenn *operand3* eine Konstante ist, kann AD nicht explizit angegeben werden. Für Konstanten gilt immer AD=O.
*n*X	**Zu überspringende Parameter:** Mit der Notation *n*X 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 drei Parameter zu überspringen). Dies bedeutet, dass für die nächsten *n* Parameter an die Method keine Werte übergeben werden. Bei einer in Natural implementierten Method muss ein zu überspringender Parameter mit dem Schlüsselwort OPTIONAL im DEFINE DATA PARAMETER-Statement des Subprogramms der Method definiert sein. OPTIONAL bedeutet, dass ein Wert vom aufrufenden Objekt an einen solchen Parameter übergeben werden kann, aber nicht unbedingt muss.
RETURN *operand4*	**RETURN-Klausel:** Wenn die RETURN-Klausel weggelassen wird und die Method einen Rückgabewert hat, wird der Rückgabewert nicht berücksichtigt. Wenn die RETURN-Klausel angegeben wird, enthält *operand4* den Rückgabewert der Method. Wenn die Ausführung der Method ohne Erfolg abgebrochen wird, wird *operand4* auf seinen ursprünglichen Wert zurückgesetzt. **Anmerkung:** Bei in Natural geschriebenen Klassen wird der Rückgabewert einer Method durch Eingabe eines zusätzlichen Parameters in der Parameter Data Area der Method und durch Kennzeichnung mit BY VALUE RESULT definiert. Weitere Informationen siehe Abschnitt ***PARAMETER-Klausel***. Deshalb enthält die Parameter Data Area einer Method, die in Natural geschrieben ist, und die einen Rückgabewert hat, neben den Methoden-Parametern immer ein zusätzliches Feld. Dies ist zu berücksichtigen, wenn Sie

Syntax-Element	Beschreibung
	eine Method einer in Natural geschriebenen Klasse aufrufen und die Parameter Data Area der Method im `SEND`-Statement verwenden möchten.
`GIVING` *operand5*	**GIVING-Klausel:** Wenn die `GIVING`-Klausel nicht angegeben wird, wird die Natural-Laufzeitfehlerverarbeitung angestoßen, wenn ein Fehler auftritt. Wenn die `GIVING`-Klausel angegeben wird, enthält *operand5* die Natural-Meldungsnummer, wenn ein Fehler aufgetreten ist, oder Null (0), wenn kein Fehler aufgetreten ist.

Beispiel für SEND METHOD-Statement

Das folgende Diagramm gibt eine Übersicht über die Programmierobjekte, die in diesem Beispiel benutzt werden. Der entsprechende Quellcode und die Programm-Ausgabe sind im Folgenden veranschaulicht


Definition der Class
Definiert die Class STUDENTS
Class
METHCL
Beispielprogramm
Erstellt Objekt L-STUDENT der Class STUDENTS und sendet Methoden
Programm
METH01
Verwendete Parameterdaten
PDA
METHA
Verwendete Objektdaten
LDA
METHO
Method-Subprogramme
Subprogramm
METH02
Method INIT
Subprogramm
METH03
Method SUMMATION
Subprogramm
METH04
Method MULTIPLICATION
Subprogramm
METH05
Method ADDRESS

Programm METH01: CTREATE OBJECT und SEND METHOD mit einer Klasse und mehreren Methods:

```
** 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.FULL-NAME := '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.FULL-NAME
#SUM   := L-STUDENT.SUM   /* property calls method SUMMATION
#MULTI := L-STUDENT.MULTI /* property calls method MULTIPLICATION
*
SEND METHOD 'ADDRESS' TO L-STUDENT          /* see METHCL
*
#STREET := L-STUDENT.STREET
#CITY   := L-STUDENT.CITY
*
*
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     METHO     Library SYSEXSYN                       DBID    10 FNR    32
Command                                                                  > +
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 METHA     Library SYSEXSYN                       DBID    10 FNR    32
Command                                                                  > +
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 Method 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 Method 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 Method 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 Method 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:

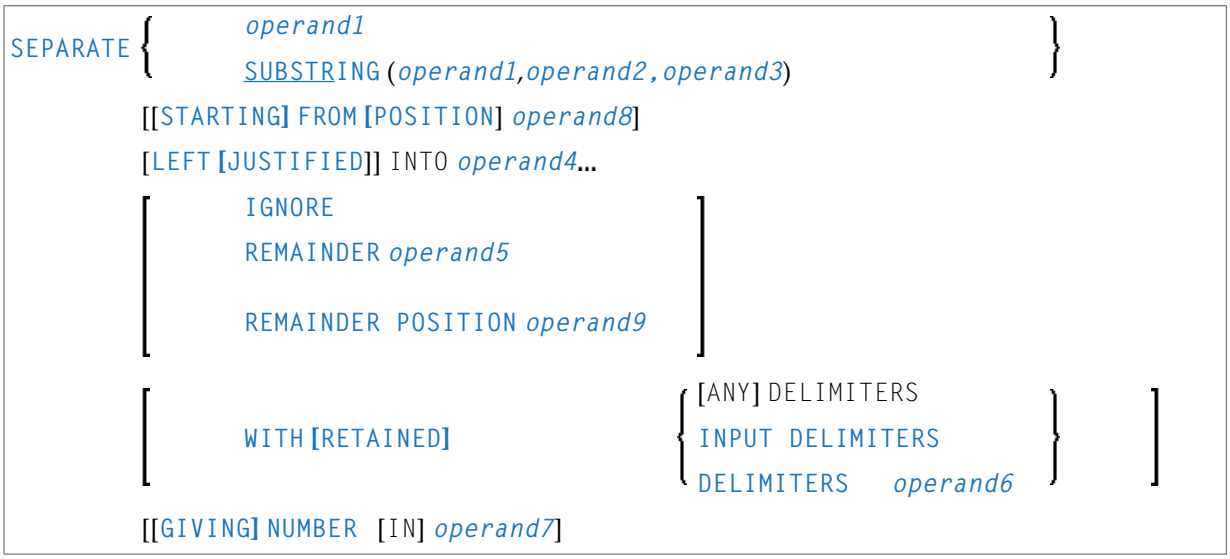
```
Page      1                                          05-01-17  15:59:04

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
```

121 SEPARATE

- Funktion SEPARATE 996
- Syntax-Beschreibung SEPARATE 996
- Regeln und operative Aspekte 1000
- Beispiele SEPARATE 1002



Dieser Abschnitt 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 SEPARATE

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 SEPARATE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A			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

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand5		S				A	U				B							ja	ja
operand6	C	S				A	U				B							ja	nein
operand7		S						N	P	I								ja	ja
operand8	C	S						N	P	I								ja	nein
operand9		S						N	P	I								ja	yes

* Format B von *operand2* und *operand3* können nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Ausgangsoperand:** *operand1* ist die alphanumerische/binäre Konstante oder Variable, deren Inhalt aufgeteilt werden soll. Nachgestellte Leerzeichen in *operand1* werden entfernt, bevor der Wert verarbeitet wird (auch wenn das Leerzeichen als Begrenzungszeichen verwendet wird; vgl. DELIMITERS-Option).
SUBSTRING	**SUBSTRING-Option:** Normalerweise wird der ganze Inhalt des Feldes aufgeteilt, und zwar vom Anfang des Feldes an. Die SUBSTRING-Option ermöglicht es Ihnen, nur einen bestimmten Teil des Feldes aufzuteilen. In der SUBSTRING-Klausel geben Sie nach dem Feldnamen (*operand1*) zunächst die erste Stelle (*operand2*) und dann die Länge (*operand3*) 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. Normalerweise wird der ganze Inhalt des Feldes aufgeteilt, und zwar vom Anfang des Feldes an. **Anmerkung:** Wenn Sie *operand2* weglassen, wird ab Anfang des Feldes (Position 1) aufgeteilt. Wenn Sie *operand3* weglassen, wird ab der angegebenen Stelle (*operand2*) bis zum Ende des Feldes aufgeteilt.
STARTING FROM POSITION *operand8*	**STARTING FROM POSITION-Option:** Diese Option bestimmt die Startposition des aufzuteilenden Ausgangsoperanden (*operand1*). Weitere Informationen siehe *Bereiche für STARTING FROM POSITION festlegen*.
LEFT JUSTIFIED	**LEFT JUSTIFIED-Option:**

Syntax-Element	Beschreibung
	Diese Option bewirkt, dass den aufgeteilten Feldwertteilen vorangestellte Leerzeichen aus den Zieloperanden entfernt werden.
operand4	**Zieloperand:** *operand4* 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. Die Anzahl der Zieloperanden entspricht der Anzahl der Begrenzungszeichen (einschließlich nachgestellter Begrenzungszeichen) in *operand1*, plus 1. Ist *operand4* 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. Allgemeine Informationen zu dynamischen Variablen finden Sie im Abschnitt *Dynamische und große Variablen benutzen* im *Leitfaden zur Programmierung*.
IGNORE / REMAINDER *operand5*	**IGNORE/REMAINDER-Optionen:** Wenn Sie nicht genug Zieloperanden angeben, um alle Feldwertteile aufzunehmen, erhalten Sie eine entsprechende Fehlermeldung. Um dies zu vermeiden, haben Sie zwei Möglichkeiten: ■ **IGNORE-Option:** Wenn Sie IGNORE angeben, ignoriert Natural es, falls nicht genügend Zieloperanden zur Aufnahme des Ausgangswertes vorhanden sind. ■ **REMAINDER-Option:** Wenn Sie REMAINDER *operand5* angeben, wird der Teil des Ausgangswertes, für den keine Zieloperanden mehr zur Verfügung stehen, in *operand5* gestellt. Den Inhalt von *operand5* können Sie dann weiter verarbeiten, zum Beispiel in einem anschließenden SEPARATE-Statement. REMAINDER darf nur bei Feldern mit einem einzelnen Wert verwendet werden. Bei Array-Ausgangsfeldern müssen Sie REMAINDER POSITION verwenden. Siehe auch ***Regeln und operative Aspekte*** und ***Beispiel 3***.
REMAINDER POSITION *operand9*	**REMAINDER POSITION-Option:** Der von der REMAINDER POSITION-Klausel zurückgegebene Wert entspricht der Position, ab der ein REMAINDER-Datenfeld gefüllt wird. Weitere Informationen siehe ***Regeln und operative Aspekte***.
DELIMITERS	**DELIMITERS-Option:** Siehe ***Delimiters-Option*** weiter unten.
RETAINED	**RETAINED-Option:**

Syntax-Element	Beschreibung

	Normalerweise werden die Begrenzungszeichen selbst nicht mit in die Zieloperanden übertragen.
	
	Wenn Sie allerdings `RETAINED` angeben, wird jedes Begrenzungszeichen (d.h. entweder das mit dem Session-Parameter `ID` festgelegte Standard-Begrenzungszeichen und Leerzeichen oder die mit *operand6* angegebenen Zeichen) ebenfalls in einen Zieloperanden übertragen.
	
	Beispiel:
	
	Das folgende `SEPARATE`-Statement überträgt `150` nach `#B`, das Plus-Zeichen (`+`) nach `#C` und `30` nach `#D`:
	
	`...`
	`MOVE '150+30' TO #A`
	`SEPARATE #A INTO #B #C #D WITH RETAINED DELIMITER '+'`
	`...`
	
	Siehe auch ***Beispiel 3***.

`GIVING NUMBER` *operand7*	**GIVING NUMBER-Option** Diese Option bewirkt, dass die Anzahl der Zieloperanden, die mit einem Wert gefüllt wurden (einschließlich der mit Leerzeichen gefüllten), in *operand7* ausgegeben wird. Die Anzahl, die Sie erhalten, errechnet sich aus der Anzahl der Delimiterzeichen plus 1. Wenn Sie die `IGNORE`-Option verwenden, enthält *operand7* maximal die Anzahl der Zieloperanden (*operand4*). Wenn Sie die REMAINDER-Option verwenden, enthält *operand7* maximal die Anzahl der Zieloperanden (*operand4*) plus *operand5*.

DELIMITERS-Option:

Begrenzungszeichen innerhalb von *operand1* bestimmen die Stellen, an denen der Wert geteilt werden soll.

```
[ WITH [RETAINED] { [ANY] DELIMITERS      } ]
                  { INPUT DELIMITERS      }
                  { DELIMITERS operand6   }
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`WITH [ANY] DELIMITERS`	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. Die Definition des Begrenzungszeichens kann mit dem Profilparameter `SCTAB` geändert werden.
`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.

Regeln und operative Aspekte

- Verarbeitung von Ausgangs- und Zielfeldern
- Bereiche für STARTING FROM POSITION festlegen
- Rückgabewert der REMAINDER POSITION-Klausel
- Feldüberlappungen: REMAINDER und REMAINDER POSITION
- Begrenzungszeichen beim SEPARATE-Statement

Verarbeitung von Ausgangs- und Zielfeldern

Nachgezogene Leerstellen in Ausgangsfeldern (und auch in einzelnen Werten und Array-Ausprägungen) werden ignoriert, wenn der Aufteilungsvorgang beginnt. Nachgezogene Leerstellen zählen nur dann, wenn der `REMAINDER POSITION`-Wert berechnet wird. Siehe auch ***Rückgabewert der REMAINDER POSITION-Klausel***.

Ist das Ausgangsfeld (*operand1*) ein leeres dynamisches Feld (`*LENGTH=0`) oder ein X-Array, das nicht erweitert wird, beendet das `SEPARATE`-Statement die Ausführung nach Rücksetzen folgender Felder:

- alle Zielfelder (*operand4*);
- das Feld (*operand7*), welches die Anzahl der gefüllten Zielfelder zurückgibt;
- das `REMAINDER`-Datenfeld (*operand5*);
- das `REMAINDER POSITION`-Feld (*operand9*).

Dasselbe gilt, wenn das Ausgangsfeld nur Leerstellen enthält.

Bereiche für STARTING FROM POSITION festlegen

Der zulässige Wertebereich für *operand8* in der STARTING FROM POSITION-Klausel ist 1:*n*. Dabei ist *n* das letzte Byte des Ausgangsfeldes.

Falls es sich bei dem Ausgangsfeld (*operand1*) um ein Array handelt, werden alle Ausprägungen, einschließlich der nachgezogenen Leerstellen, gezählt. Bei einem dynamischen Arry wird die Länge jedes Einzelfeldes bis zur angegebenen Position gezählt.

Beispiele für *operand8*:

Position 63 in #A (A100)	ist das the 63. Zeichen in #A.
Position 63 in #B (A20/1:10)	ist das dritte Zeichen in #B(4).
Position 63 in #C (A10/1:3,1:4)	ist das dritte Zeichen in #C(2,3).
Position 63 in #D (1:5) DYNAMIC mit *LENGTH(#D(*)) = (15,25,0,33,61)	ist das 23. Zeichen in #D(4).

Bei Angabe eines ungültigen Wertebereichs (ein negativer Wert oder ein Null-Wert oder ein Wert, der größer als die tatsächliche Feldlänge ist) werden die im Abschnitt ***Verarbeitung von Ausgangs- und Zielfeldern*** aufgeführten Rückgabefelder zurückgesetzt, jedoch tritt kein Laufzeitfehler auf. Da der STARTING FROM-Wert eine Position (und keinen Versatz) angibt, wird in *operand8* ein Mindestwert von 1 für die erste Ausführung benötigt.

Rückgabewert der REMAINDER POSITION-Klausel

Der von der REMAINDER POSITION-Klausel zurückgegebene Wert entspricht der Position, ab der ein REMAINDER-Datenfeld gefüllt wird.

Beispiel:

```
...
SEPARATE 'AB  CD' INTO #A REMAINDER #R
...
```

Dieses Statement gibt #A= 'AB' und #R= ' CD' zurück, weil der REMAINDER nach dem Trennzeichen (hier: Leerzeichen) unmittelbar nach AB beginnt. Verwendet man stattdessen die REMAINDER POSITION-Option, wird der Wert 4 zurückgegeben.

Zwar werden nachgezogene Leerstellen während des Trennvorgangs ignoriert, aber bei der Berechnung des REMAINDER POSITION-Wertes werden sie bei den Ausprägungen eines Array-Ausgangsfeldes berücksichtigt.

Wenn alle Ausgangsfeldabschnitte verarbeitet sind und das Ende des Ausgangsfelds erreicht ist, gibt die REMAINDER POSITION-Klausel den Wert Null zurück, was besagt, dass keine weiteren Daten mehr vorhanden sind.

Siehe auch *Beispiel 6 - Verwendung eines Array-Ausgangsfeldes mit den Optionen STARTING FROM und REMAINDER POSITION*.

Feldüberlappungen: REMAINDER und REMAINDER POSITION

Bei der Ausführung des `SEPARATE`-Statements werden die Ausgangsdaten normalerweise kopiert und aus einem Arbeitsfeld verarbeitet. Deshalb ist das `REMAINDER`-Ergebnis unabhängig von möglicherweise überlappenden Ausgangs- und Ergebnis-(INTO-)Feldern

Solche Feld-Sicherungskopien werden bei Verwendung einer `REMAINDER POSITION`-Klausel nicht erstellt. Der gesamte Trennvorgang arbeitet unabhängig davon, ob Sie den Ausgangs- und den Zieloperanden trennen, mit dem ursprünglichen Ausgangsoperanden. Operandenüberlappung werden weder beim Kompilieren noch bei der Ausführung zurückgewiesen, können aber zu unerwünschten Ergebnissen führen.

Begrenzungszeichen beim SEPARATE-Statement

Wenn Sie ein Feld mit einem einzelnen Wert aufteilen, begrenzt die Feldgrenze immer das letzte Wort. Dies gilt ebenfalls bei jeder einzelnen Ausprägung eines Array-Feldes. Das bedeutet, dass das folgende Einzel-Statement dasselbe bewirkt wie die Statement-Sequenz:

Bei Verwendung der `RETAINED DELIMITERS`-Option werden die Begrenzungszeichen ebenfalls in das Zielfeld gestellt. Dies gilt nur bei Begrenzungszeichen innerhalb einer Array-Ausprägung und nicht bei aufeinander folgenden Array-Ausprägungen, welche automatisch (ohne Begrenzungszeichen) begrenzt werden, wenn eine Ausprägung endet.

Siehe auch *Beispiel 4 - Verwendung eines Array-Ausgangsfeldes einer redefinierten Zeichenkette* und *Beispiel 5 - Verwendung eines Array-Ausgangsfeldes mit RETAIN-Option*.

Beispiele SEPARATE

- Beispiel 1 - Verschiedene Beispiele für den Gebrauch des SEPARATE-Statements
- Beispiel 2 - SEPARATE-Statement bei einem Array
- Beispiel 3 - Gebrauch der Optionen REMAINDER/RETAINED
- Example 4 - Beispiel 4 - Verwendung eines Array-Ausgangsfeldes einer redefinierten Zeichenkette
- Beispiel 5 - Verwendung eines Array-Ausgangsfeldes mit RETAIN-Option

- Beispiel 6 - Verwendung eines Array-Ausgangsfeldes mit den Optionen STARTING FROM und REMAINDER POSITION

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
*
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
```

Example 4 - Beispiel 4 - Verwendung eines Array-Ausgangsfeldes einer redefinierten Zeichenkette

```
**  Example 'SEPEX4': SEPARATE with source array
************************************************************************
* This example shows different results when separating a scalar string
* or a string array redefining the scalar string.
*
*
************************************************************************
*
*
DEFINE DATA LOCAL
1 #TEXT (A24) INIT <'VAL1 VAL2 VAL3 VAL4 VAL5'>
1 REDEFINE #TEXT
  2 #TEXTARRAY (A12/2)
1 #WORD1(A5/6)
1 #WORD2(A5/6)
END-DEFINE
*
SEPARATE #TEXT INTO #WORD1(*)
/* Redefinition may split original words into two parts
SEPARATE #TEXTARRAY(*) INTO #WORD2(*)
*
DISPLAY #TEXT #WORD1(*) #TEXTARRAY(*) #WORD2(*)
END
```

Ausgabe des Programms `SEPEX4`:

```
         #TEXT           #WORD1  #TEXTARRAY  #WORD2
----------------------- ------ ------------ ------

VAL1 VAL2 VAL3 VAL4 VAL5 VAL1   VAL1 VAL2 VA VAL1
                         VAL2   L3 VAL4 VAL5 VAL2
                         VAL3                VA
                         VAL4                L3
                         VAL5                VAL4
                                             VAL5
```

Beispiel 5 - Verwendung eines Array-Ausgangsfeldes mit RETAIN-Option

```
**  Example 'SEPEX5': SEPARATE with and without RETAINED DELIMITERS
************************************************************************
* This example shows different results with a source array
* when using the option RETAINED DELIMITERS or not.
*
*
************************************************************************
*
*
DEFINE DATA LOCAL
1 #TEXT(A20)        INIT <'VAL1,VAL2,VAL3,VAL4'>
1 #TEXTARRAY(A10/3) INIT <'VAL1,VAL2',
                          'VAL3',
                          'VAL4'>
1 #WORD1(A5/7)
1 #WORD2(A5/7)
END-DEFINE
*
SEPARATE #TEXT         INTO #WORD1(*)
SEPARATE #TEXTARRAY(*) INTO #WORD2(*)
DISPLAY #TEXT #WORD1(*) #TEXTARRAY(*) #WORD2(*)
*
SEPARATE #TEXT         INTO #WORD1(*) WITH RETAINED DELIMITERS
SEPARATE #TEXTARRAY(*) INTO #WORD2(*) WITH RETAINED DELIMITERS
DISPLAY #TEXT #WORD1(*) #TEXTARRAY(*) #WORD2(*)
*
END
```

Ausgabe des Programms `SEPEX5`:

```
       #TEXT          #WORD1 #TEXTARRAY #WORD2
-------------------- ------ ---------- ------

VAL1,VAL2,VAL3,VAL4  VAL1   VAL1,VAL2  VAL1
                     VAL2   VAL3       VAL2
                     VAL3   VAL4       VAL3
                     VAL4              VAL4



VAL1,VAL2,VAL3,VAL4  VAL1   VAL1,VAL2  VAL1
                     ,      VAL3       ,
                     VAL2   VAL4       VAL2
                     ,                 VAL3
                     VAL3              VAL4
                     ,
                     VAL4
```

Beispiel 6 - Verwendung eines Array-Ausgangsfeldes mit den Optionen STARTING FROM und REMAINDER POSITION

```
**  Example 'SEPEX6': SEPARATE with STARTING FROM and REMAINDER POSITION
************************************************************************
* This example shows how the options STARTING FROM POSITION and
* REMAINDER POSITION work together in a processing loop when
* separating a source array.
*
************************************************************************
*
*
DEFINE DATA LOCAL
1 #TEXT (A15/1:3) INIT <'VAL1 VAL2',
                        'VAL3',
                        'VAL4 VAL5 VAL6'>
1 #WORD (A5/1:4)
1 #POS  (I1) INIT <1>
END-DEFINE
*
WRITE '#TEXT(A15/1:3):      (1)             (2)             (3)'
  /  16T #TEXT(*)
  /  16T '----+----1----+ ----2----+----3 ----+----4----+'
  // '#WORD (A5/1:4): (1)   (2)   (3)   (4)      :  #POS'
     '(within #TEXT(*))'
*
REPEAT
  SEPARATE #TEXT(*) STARTING FROM POSITION #POS
      INTO #WORD(*) REMAINDER POSITION #POS
  WRITE 16T #WORD(*) 44T ': ' #POS
  UNTIL #POS = 0
END-REPEAT
END
```

Ausgabe des Programms `SEPEX6`:

```
#TEXT(A15/1:3):      (1)             (2)             (3)
               VAL1 VAL2       VAL3            VAL4 VAL5 VAL6
               ----+----1----+ ----2----+----3 ----+----4----+

#WORD (A5/1:4): (1)   (2)   (3)   (4)      :  #POS (within #TEXT(*))
               VAL1  VAL2  VAL3  VAL4      :    36
               VAL5  VAL6                  :     0
```

122 SET CONTROL

- Funktion SET CONTROL 1010
- Syntax-Beschreibung SET CONTROL 1010
- Beispiele SET CONTROL 1011

```
SET CONTROL operand1 ...
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion SET CONTROL

Mit dem Statement `SET CONTROL` können Sie ein Terminalkommando von einem Programm aus ausführen.

Syntax-Beschreibung SET CONTROL

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A											ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Auszuführende Terminalkommandos:** Das Terminalkommando (*operand1*) wird standardmäßig ohne das Kontrollzeichen (%) angegeben und kann als Textkonstante oder als Inhalt einer alphanumerischen Variablen angegeben werden. Informationen zu den einzelnen Terminalkommandos finden Sie in der *Terminalkommandos*-Dokumentation.

Beispiele SET CONTROL

- 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`*).

123 SET GLOBALS

- Funktion SET GLOBALS 1014
- Parameterangabe bei SET GLOBALS 1014
- Beispiel für SET GLOBALS-Statement 1016

```
SET GLOBALS parameter=value ...
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion SET GLOBALS

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 zur Laufzeit 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.

Ausnahme: Auf Großrechnern gilt ein SET GLOBALS-Statement in einem Subprogramm (d.h. einer Subroutine, einem Subprogramm oder einem mit FETCH RETURN aufgerufenen Programm) nur solange, bis die Kontrolle von dem Subprogramm wieder an das aufrufende Objekt übergeben wird; dann gelten wieder die für das aufrufende Objekt gesetzten Parameterwerte.

Parameterangabe bei SET GLOBALS

Wenn Sie mehrere Parameter angeben, müssen Sie diese durch ein oder mehrere Leerzeichen voneinander trennen. Die Reihenfolge der Parameter ist beliebig. Siehe ***Beispiel***.

Parameter	Funktion (Kurzbeschreibung*)	Auswertung
CC	Fehlerverarbeitung im Batch-Modus	zur Laufzeit
CF	Steuerzeichen für Terminalkommandos	zur Laufzeit
CPCVERR	Codepage-Umsetzungsfehler	zur Laufzeit
DC	Dezimalstellenzeichen	zur Laufzeit
DFOUT	Datumsformat für Ausgabe	zur Laufzeit
DFSTACK	Datumsformat für Stack	zur Laufzeit
DFTITLE	Datumsformat in Standard-Seitenüberschrift	zur Laufzeit
DO	Anzeige-Reihenfolge von Ausgabedaten	zur Laufzeit
DU	Dump-Generierung	zur Laufzeit
EJ	Seitenvorschub	zur Laufzeit
FCDP	Füllzeichen für dynamisch geschützte Felder	zur Laufzeit
FS	Format-Spezifikation für Benutzervariablen	zur Laufzeit
IA	Input-Zuweisungszeichen	zur Laufzeit
ID	Input-Begrenzungszeichen	zur Laufzeit
IM	Input-Modus	zur Laufzeit
LE	Reaktion auf Limit-Überschreitung bei Verarbeitungsschleifen	bei der Kompilierung
LS	Zeilenlänge	bei der Kompilierung
LT	Limit für Verarbeitungsschleifen	zur Laufzeit
MT	Maximale CPU-Zeit	zur Laufzeit
OPF	Überschreiben geschützter Felder durch Helproutinen	zur Laufzeit
PD	Seiten-Limit für NATPAGE	zur Laufzeit
PM	Druck-Modus	bei der Kompilierung
PS	Länge einer Reportseite	bei der Kompilierung und zur Laufzeit
REINP	Interner REINPUT bei ungültigen Daten	zur Laufzeit
SA	Terminal-Warnton	zur Laufzeit
SF	Spaltenabstand	bei der Kompilierung
TS	Umsetzung von Systemdatei-Programmausgaben	zur Laufzeit
WH	Warten auf Datensatz im Hold	zur Laufzeit
ZD	Division durch Null	zur Laufzeit
ZP	Anzeige von Nullwerten	zur Laufzeit

* Ausführliche Informationen zu den einzelnen Parametern sind in der *Parameter-Referenz*-Dokumentation enthalten.

Beispiel für SET GLOBALS-Statement

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 LS=74 LT=5000
...
```

124 SET KEY

- Funktion SET KEY 1018
- Syntax-Beschreibung SET KEY 1018
- Tasten programm-sensitiv machen und deaktivieren 1020
- Kommandos/Programme einer Taste zuweisen 1021
- Eingabedaten einer Taste zuweisen (DATA) 1022
- Tastenfunktion vorübergehend deaktivieren 1022
- Helproutine zuweisen (HELP) 1023
- Dynamische Funktionszuweisung (DYNAMIC) 1023
- GUI-Element-Zuweisung deaktivieren (DISABLED) 1024
- SET KEY-Statements auf verschiedenen Programmebenen 1024
- Namen zuweisen 1026
- Beispiel für SET KEY-Statement 1027

Dieser Abschnitt behandelt folgende Themen:

Funktion SET KEY

Das SET KEY-Statement dient dazu, den folgenden Tasten-Arten Funktionen zuzuweisen:

- Videoterminal PA-Tasten (Programmabruftasten)
- 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). Auf Großrechnern hängt die Verarbeitung von PA- und PF-Tasten auch davon ab, wie Ihr Natural-Administrator den Natural-Profilparameter KEY gesetzt hat.

Siehe auch *Verarbeitung aufgrund von Funktionstasten* im *Leitfaden zur Programmierung*.

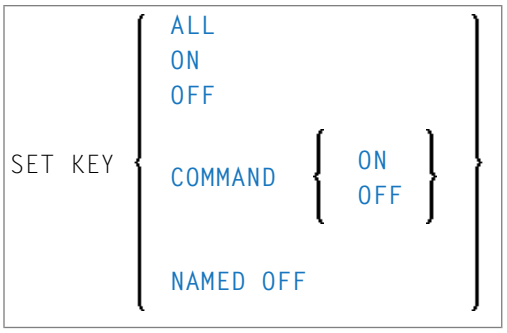
Programmierschnittstelle (API): USR4005N. Siehe auch *SYSEXT Utility* in der *Debugger und Dienstprogramme*-Dokumentation.

Syntax-Beschreibung SET KEY

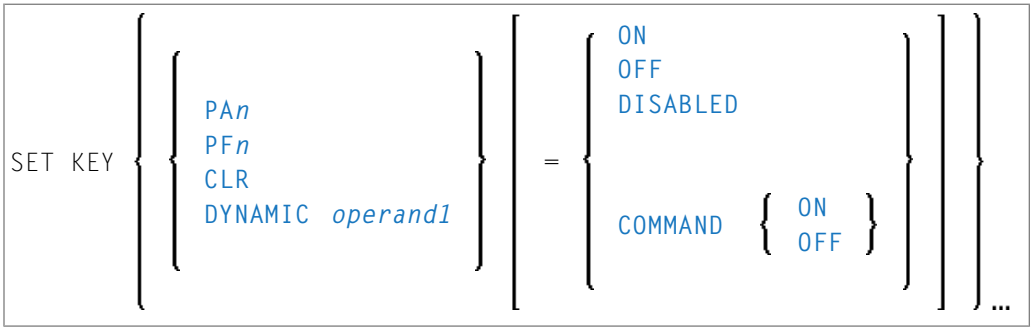
Mehrere Strukturen sind bei diesem Statement möglich.

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

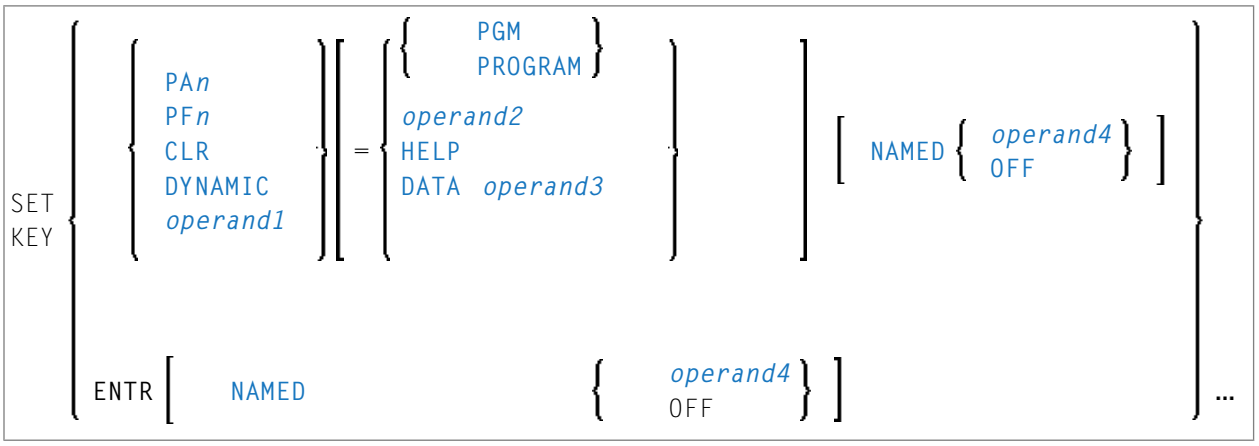
Syntax 1 - für alle Tasten:



Syntax 2 - für einzelne Tasten:



Syntax 3 - für einzelne Tasten:



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1		S				A											ja	nein
operand2	C	S				A	U										ja	nein
operand3	C	S				A	U										ja	nein
operand4	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 und diese Zuweisung auch wieder löschen. 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 SAVE wird PF4 zugewiesen.
SET KEY PF4=#XYX	Der in der Variablen #XYZ enthaltene Wert wird PF4 zugewiesen.
SET KEY PF6='LIST MAP *'	Das Kommando LIST, einschließlich der LIST-Kommando-Parameter MAP und * wird PF6 zugewiesen.
SET KEY PF2='%%'	Das Terminalkommando %% wird PF2 zugewiesen.
SET KEY PF9=' '	Das Kommando und der Name, die zuvor der Taste PF9 zugewiesen wurden, werden gelöscht.

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 PF*nn*='' 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 (PF*n*, PA*n*, 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 PF`*nn*`=ON` können Sie `SET KEY PF`*nn*`=DISABLED` wieder rückgängig machen.

Beispiel:

`SET KEY PF10=DISABLED`	Deaktiviert das PF10 zugewiesene GUI-Element.

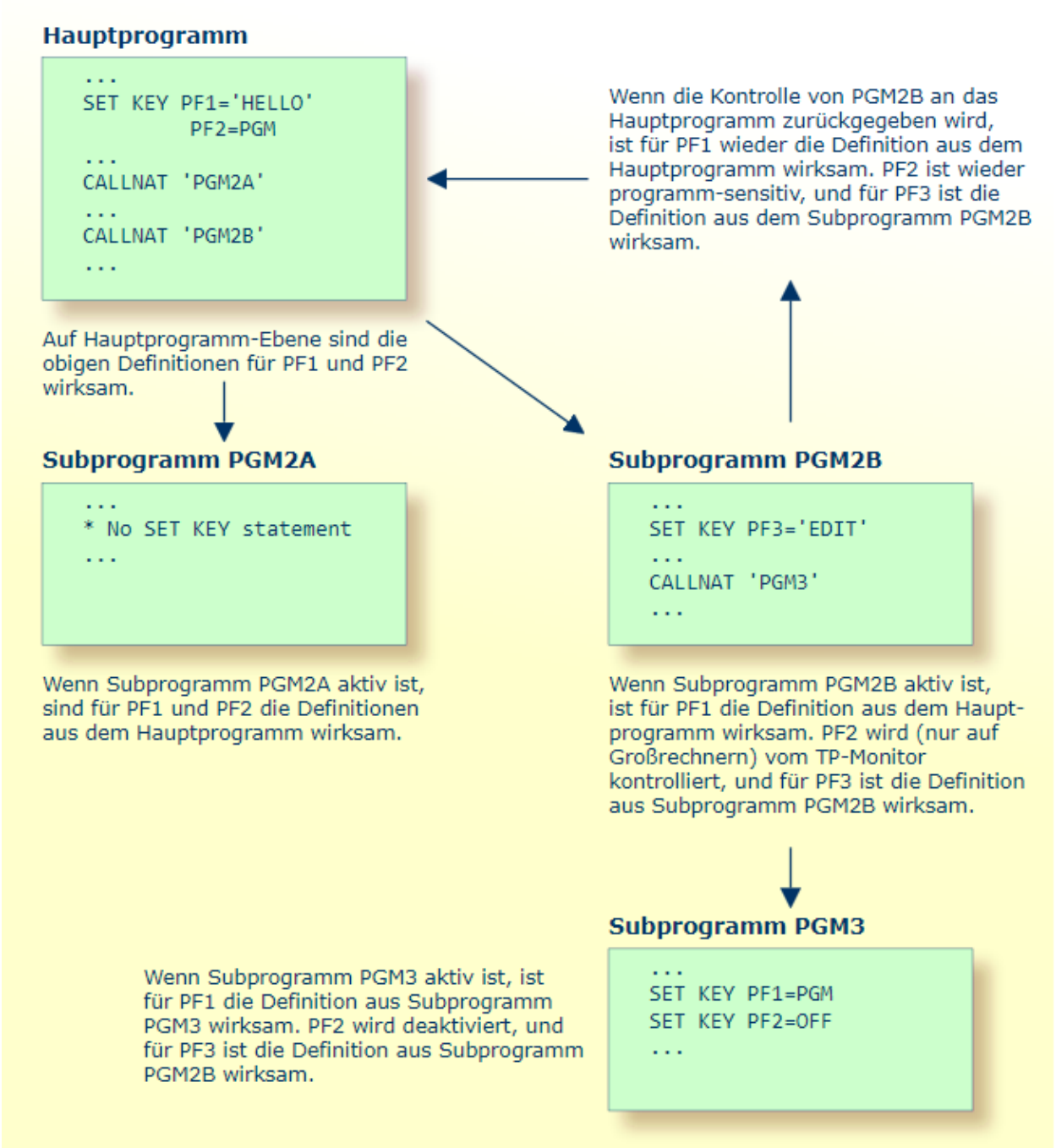
Die `DISABLED`-Option kann nur in Processing Rules verwendet werden.

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. Subprogramme) 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. Die Form der Anzeige der PF-Tastenleiste können Sie mit dem Terminalkommando %Y beeinflussen.

Der Name, den Sie einer Taste zuweisen, darf bis zu 10 Stellen lang sein. Im normalen tabellarischen PF-Tastenleistenformat (%YN) 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 (`%YN`) 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 (`%YS` oder `%YP`).

Wenn Sie sequentielles PF-Tastenleistenformat (`%YS` oder `%YP`) 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 für SET KEY-Statement

```
** 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 'Empl'
*
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'
```

```
  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
```

125 SET TIME

- Funktion SET TIME 1030
- Beispiel für SET TIME 1030



Dieser Abschnitt behandelt folgende Themen:

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

Funktion SET TIME

Das Statement `SET TIME` (oder `SETTIME`) wird in Verbindung mit der Natural-Systemvariablen `*TIMD` verwendet und dient dazu, die für die Ausführung eines bestimmten Programmteils 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 Quellcode-Zeilennummer oder mittels eines Statement-Labels.

Beispiel für SET TIME

```
** 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
```

126 SET WINDOW

- Funktion SET WINDOW 1034
- Syntax-Beschreibung SET WINDOW 1034
- Beispiel für SET WINDOW 1035

```
SET WINDOW  { 'window-name' }
            {     OFF     }
```

Dieser Abschnitt 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 SET WINDOW

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

Syntax-Element	Beschreibung
`SET WINDOW 'window-name'`	Mit `SET WINDOW 'window-name'` 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 `DEFINE WINDOW`-Statement definiert worden sein.
`SET WINDOW OFF`	Mit `SET WINDOW OFF` deaktivieren Sie das gerade aktive Fenster.

Beispiel für SET WINDOW

Siehe `DEFINE WINDOW`-Statement.

127 SKIP

- Funktion SKIP 1038
- Syntax-Beschreibung SKIP 1038
- Beispiel für SKIP-Statement 1039

```
SKIP [(rep)] operand1 [LINES]
```

Dieser Abschnitt 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 SKIP

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 SKIP

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1	C	S					N	P	I							ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(rep)	**Report-Spezifikation::** Mit der Notation *(rep)* kann ein bestimmter anderer Report angegeben werden, auf den sich das Statement SKIP 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 SKIP-Statement auf den ersten ausgegebenen Report (Report 0). Informationen darüber, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern können, finden Sie im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
operand1	**Anzahl der Leerzeilen:** *operand1* ist die Anzahl der zu generierenden Leerzeilen (1 - 250). Die Anzahl kann als numerische Konstante oder als Inhalt einer numerischen Variablen angegeben werden. Ist *operand1* größer als die für den Report definierte Seitenlänge, so löst das SKIP-Statement eine *Newpage*-Bedingung aus.

Beispiel für SKIP-Statement

```
** 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

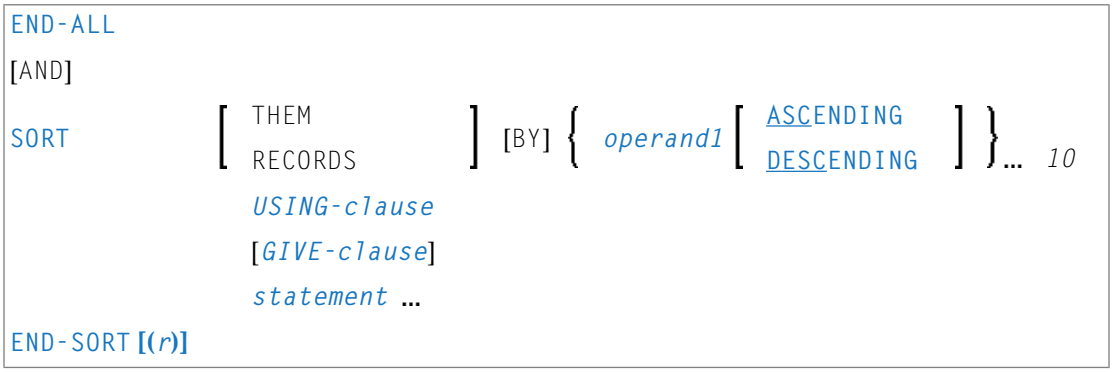

WEST BRIDGFORD       UK      ENTWHISTLE


WEST MIFFLIN         USA     WATSON
```

128 SORT

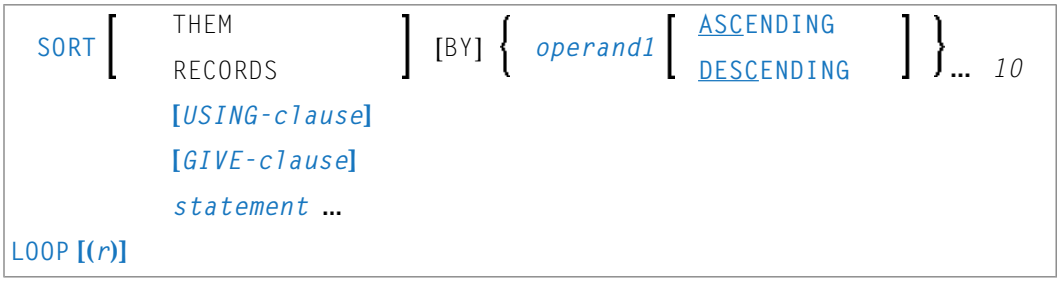
- Funktion SORT 1042
- Einschränkungen beim SORT-Statement 1043
- Syntax-Beschreibung SORT 1043
- Phasen der SORT-Verarbeitung 1046
- Beispiel für SORT-Statement 1047
- Benutzung externer Sortierprogramme 1052

Structured Mode-Syntax



* Wenn ein Statement-Label angegeben wird, muss es vor dem Schlüsselwort `SORT`, aber *nach* `END-ALL` (und `AND`) stehen.

Reporting Mode-Syntax



Dieser Abschnitt 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 SORT

Das Statement `SORT` dient dazu, eine Sortieroperation durchzuführen und die Datensätze aus allen Verarbeitungsschleifen, die zum Zeitpunkt der `SORT`-Ausführung aktiv sind, zu sortieren.

Für die Sortieroperation wird Naturals internes Sortierprogramm verwendet. Es kann stattdessen auch ein anderes, externes Sortierprogramm verwendet werden.

Einschränkungen beim SORT-Statement

- 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 SORT

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand1		S				A	N	P	I	F	B	D	T					nein	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
END-ALL	**Schließen aller zurzeit aktiven Schleifen:** 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. **Anmerkung:** Im Reporting Mode beendet das SORT-Statement alle noch aktiven Verarbeitungsschleifen und initiiert eine neue Verarbeitungsschleife.
operand1	**Sortierkriterien:** *operand1* 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. **Anmerkung:** Ein in den Sortierkriterien angegebenes Feld wird benutzt, um in der **Auswahlphase** (erste Phase) einen Wert in den Sortier-Datensatz zu stellen und um in der **Verarbeitungsphase** (dritte Phase) den sortierten Wert zu empfangen. Bitte beachten Sie, dass dies Adressierungsfehler verursachen kann, wenn indizierte Array-Felder benutzt werden, die einen korrekten Indexwert in der ersten Phase, aber einen außerhalb des

Syntax-Element	Beschreibung
	Bereichs liegenden Wert in der dritten Phase haben. Deshalb sind indizierte Array-Felder mit Vorsicht zu verwenden. Besser ist es, diese durch nicht-indizierte Felder (Skalar) zu ersetzen.
ASCENDING DESCENDING	**Sortierreihenfolge:** 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. Das Schlüsselwort ASCENDING bzw. DESCENDING kann für jedes Sortierfeld getrennt angegeben werden.
USING	**USING-Klausel:** Siehe ***USING-Klausel*** weiter unten. **Anmerkung:** Die unter *operand1* vorhandene Anmerkung gilt auch bei der USING-Klausel.
GIVE	**GIVE-Klausel:** Siehe ***GIVE-Klausel*** weiter unten.
END-SORT (*r*) LOOP (*r*)	**Ende des SORT-Statements:** Im Structured Mode muss das für Natural reservierte Wort END-SORT zum Beenden des SORT-Statements benutzt werden. Im Structured Mode können Sie bei END-SORT Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des SORT-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

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.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand2		S	A			A	N	P	I	F	B	D	T	L	C			nein	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`USING` *operand2*	Mit *`USING operand2`* können Sie weitere Felder angeben, die - zusätzlich zu den (als *operand1* angegebenen) Sortierfeldern - in den Sortier-Zwischenspeicher geschrieben werden sollen.
`USING KEYS`	Wenn Sie `USING KEYS` angeben, werden nur die als *operand1* angegebenen Sortierfelder in den Sortier-Zwischenspeicher geschrieben.

Im Reporting Mode:

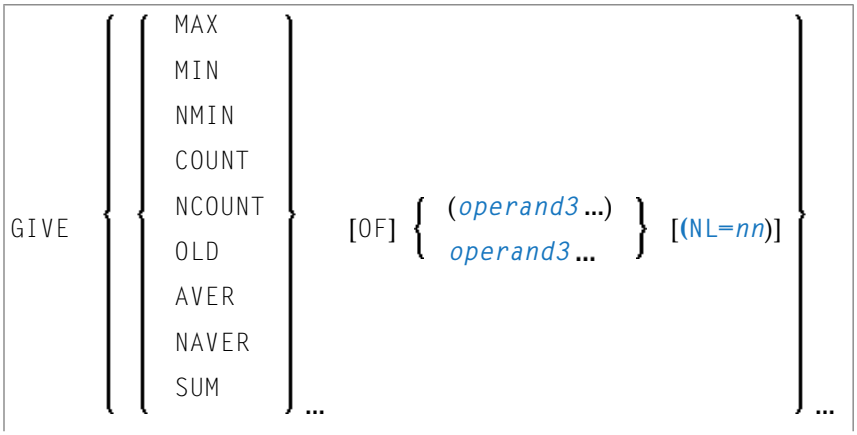
Verwenden Sie keine `USING`-Klausel, so werden alle Datenbankfelder aus vor dem `SORT`-Statement initiierten Verarbeitungsschleifen sowie alle vor dem `SORT`-Statement definierten Benutzervariablen in den Sortier-Zwischenspeicher geschrieben.

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)`.

Anmerkung: Anstelle des Schlüsselworts `GIVE` können Sie auch das Schlüsselwort `GIVING` verwenden.



```
TOTAL
```

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand3		S	A			*												ja	nein

* je nach Funktion

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`MAX | MIN | NMIN | COUNT | NCOUNT | OLD | AVER | NAVER | SUM | TOTAL`	Näheres zu den einzelnen Systemfunktionen finden Sie in der *Systemfunktionen*-Dokumentation.
operand3	*operand3* ist der Feldname.
`(NL=`*nn.m*`)`	Diese Option gilt nur für `AVER`, `NAVER`, `SUM` und `TOTAL`; für alle anderen Systemfunktionen wird sie ignoriert. Siehe auch Session-Parameter `NL` in der *Parameter Reference*-Dokumentation. Diese Option kann dazu verwendet werden, einen arithmetischen Überlauf bei der Auswertung von Systemfunktionen zu vermeiden; sie ist unter *Format- und Längenerfordernisse bei AVER, NAVER, SUM und TOTAL* in der *Systemfunktionen*-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 Quellcode-Zeilennummer erfolgen.

Beispiel für SORT-Statement

- Beispiel 1 - SORT
- Beispiel 2 - SORT

- Beispiel 3 - 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   ANNUAL     ANNUAL   #TOTAL-SALARY CURRENCY PERCENT
    ID       SALARY     SALARY                   CODE     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
--------------------
RENAULT          1980
                 1980
                 1982
                 1982
--------------------
```

Beispiel 3 - SORT

```
** Example 'SRTEX3': SORT values in an array
************************************************************************
DEFINE DATA LOCAL
1 #I   (I4)
1 #J   (I4)
1 #X   (I1)
1 #TAB (I1/1:6) INIT <2,4,6,5,3,1>
END-DEFINE
WRITE   'Array before SORT:' #TAB(*) /
*
FOR #I := 1 TO 6
  #X := #TAB(#I)
  WRITE #X '<-- Put into SORT record'
END-ALL
SORT #X USING KEYS
  WRITE #X '<-- Get from SORT'
  ADD 1 TO #J
  #TAB(#J) := #X
END-SORT
*
WRITE / 'Array after  SORT:' #TAB(*)
END
```

Ausgabe des Programms `SRTEX3`:

```
Array before SORT:    2    4    6    5    3    1

   2 <-- Put into SORT record
   4 <-- Put into SORT record
   6 <-- Put into SORT record
   5 <-- Put into SORT record
   3 <-- Put into SORT record
   1 <-- Put into SORT record
   1 <-- Get from SORT
   2 <-- Get from SORT
   3 <-- Get from SORT
   4 <-- Get from SORT
   5 <-- Get from SORT
   6 <-- Get from SORT

Array after  SORT:    1    2    3    4    5    6
```

Benutzung externer Sortierprogramme

In Natural werden Sortiervorgänge standardmäßig von Naturals internem Sortierprogramm verarbeitet, wie oben beschrieben. Allerdings kann auch ein externes Sortierprogramm benutzt werden. Dieses externe Sortierprogramm verarbeitet dann die Sortiervorgänge anstatt das internen Sortierprogramms von Natural.

Das zu benutzende externe Sortierprogramm wird vom Natural-Administrator im Makro `NTSORT` des Natural-Parametermoduls festgelegt. Zur Benutzung eines externen Sortierprogrammes ist zusätzliche JCL erforderlich. Wenn Sie Fragen haben, wenden Sie sich an Ihren Natural-Administrator.

129 STACK

- Funktion STACK 1054
- Syntax-Beschreibung STACK 1054
- Beispiel für STACK-Statement 1057

```
STACK [TOP]  { COMMAND operand1 [operand2 [(parameter)]] ...
             { [DATA] [FORMATTED] {operand2 [(parameter)]} ...
```

Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements: `INPUT` | `RELEASE`

Funktion STACK

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 Natural-Stack finden Sie im Kapitel *Weitere Programmieraspekte*, *Natural-Stack* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung STACK

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`TOP`	Normalerweise werden die Daten unten im Natural-Stack abgelegt. Das Schlüsselwort `TOP` bewirkt, dass die Daten oben auf dem Natural-Stack abgelegt werden. Beispiel: Mit diesem Statement wird der Inhalt der Variablen `#FIELDA` oben auf dem Natural-Stack abgelegt:

Syntax-Element	Beschreibung
	``` STACK TOP #FIELDA ```
DATA	Mit DATA (Standardeinstellung) legen Sie Daten auf dem Natural-Stack ab, die von einem INPUT-Statement als Eingabedaten verwendet werden. Begrenzungszeichen oder *Input Assign*-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 ***Eingabedaten aus dem Natural-Stack*** entnehmen. Beispiel: Mit den folgenden Statements wird der Inhalt der Variablen #FIELD1 und #FIELD2 auf dem Natural-Stack abgelegt: ``` MOVE 'ABC' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK #FIELD1 #FIELD2 ``` Diese Variablen werden als Eingabedaten an das nächste INPUT-Statement im Natural-Programm übergeben, und zwar im Begrenzungs-Modus: ``` INPUT #FIELD1 #FIELD2 ``` **Anmerkung:** Wenn *operand2* eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts auf dem Natural-Stack abgelegt, aber nicht die Datumskomponente.
FORMATTED	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. Beispiele: Mit den folgenden Statements wird ABC,DEF in #FIELD1 übertragen und XYZ in #FIELD2: ``` MOVE 'ABC,DEF' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK TOP DATA FORMATTED #FIELD1 #FIELD2 ... INPUT #FIELD1 #FIELD2 ``` 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:

Syntax-Element	Beschreibung
	``` MOVE 'ABC,DEF' TO #FIELD1 STACK TOP DATA #FIELD1 ... INPUT #FIELD1 #FIELD2 ``` **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.
COMMAND *operand1*	Um ein Kommando (bzw. einen Programmnamen) auf dem Natural-Stack abzulegen, geben Sie das Schlüsselwort COMMAND gefolgt von dem betreffenden Kommando (*operand1*) 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 auf dem Natural-Stack abgelegte Kommando aus. Beispiel: Mit dem folgenden Statement wird das Kommando RUN oben auf dem Natural-Stack abgelegt. Natural führt dieses Kommando aus, wenn normalerweise das nächstemal die NEXT-Zeile ausgegeben würde: ``` STACK TOP COMMAND 'RUN' ```
COMMAND *operand1* *operand2* ...	Zusammen mit einem Kommando (*operand1*) können Sie auch Daten (*operand2*) auf dem Natural-Stack ablegen. Diese Daten werden dann vom nächsten INPUT-Statement nach der Ausführung des Kommandos als Eingabedaten verarbeitet. Zusammen mit einem Kommando abgelegte Daten werden immer unformatiert abgelegt. **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 auf dem Natural-Stack ablegen möchten, müssen Sie hierzu zwei STACK-Statements verwenden: Ein STACK DATA *operand2*..., um die Daten abzulegen, und ein STACK COMMAND *operand1*, um das Kommando abzulegen.
parameter	Wenn *operand2* eine Datumsvariable ist, können Sie den Session-Parameter DF (Datumsformat) als *parameter* für diese Variable angeben.

Beispiel für STACK-Statement

```
** 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 SYSEXSYN

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
```

130 STOP

- Funktion STOP 1060
- Beispiel für STOP-Statement 1060

STOP

Dieser Abschnitt behandelt folgende Themen:

Funktion STOP

Mit dem Statement STOP können Sie die Ausführung eines Programmes abbrechen und erhalten dann eine Kommandoeingabezeile.

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.

Für Natural RPC: Siehe *Notes on Natural Statements on the Server* in der *Natural RPC (Remote Procedure Call)*-Dokumentation.

Beispiel für STOP-Statement

```
** 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:
```

XIII

- 131 STORE 1065
- 132 SUBTRACT 1073
- 133 SUSPEND IDENTICAL SUPPRESS 1079
- 134 TERMINATE 1085
- 135 UPDATE 1089
- 136 UPDATE (SQL) 1095
- 137 UPDATELOB 1103
- 138 UPLOAD PC FILE 1111
- 139 WRITE 1117
- 140 WRITE TITLE 1135
- 141 WRITE TRAILER 1143
- 142 WRITE WORK FILE 1153

131 STORE

- Funktion STORE 1066
- Datenbankspezifische Anmerkungen zu STORE 1067
- Syntax-Beschreibung STORE 1067
- Beispiel für STORE-Statement 1069

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    ]          ...              }
```

Dieser Abschnitt 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 | READLOB | RETRY | UPDATE | UPDATELOB

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Funktion STORE

Das Statement STORE dient dazu, auf einer Datenbank einen Datensatz hinzuzufügen.

Datenbankspezifische Anmerkungen zu STORE

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.
VSAM	Ist der Datensatz mit einem Primärschlüssel definiert, so muss ein Wert für das Primärschlüssel-Feld angegeben werden. Die Natural-Systemvariable enthält die Adabas-ISN bzw. VSAM-RBA oder -RRN, die dem neuen Datensatz als Ergebnis der `STORE`-Operation zugeteilt wurde. Um den Wert von `*ISN` zu erhalten, müssen Sie das betreffende `STORE`-Statement referenzieren (mittels Statement-Label oder Quellcode-Zeilennummer). Bei VSAM-Dateien gilt `*ISN` nur für ESDS- und RRDS-Dateien.

Syntax-Beschreibung STORE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A												ja	nein
operand2	C	S						N										ja	nein
operand3	C	S						N	P			B *						nein	ja
operand4		S	A			A		N	P	I	F	B	D	T	L			nein	nein
operand5	C	S	A			A		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:

Syntax-Element	Beschreibung
view-name	Als *view-name* 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. Im Reporting Mode ist *view-name* der Name eines DDM, falls kein DEFINE DATA LOCAL-Statement benutzt wird.
PASSWORD=*operand1*	Die PASSWORD-Klausel gilt nur bei Adabas- und VSAM-Datenbanken. Sie dient dazu, ein Passwort (*operand1*) anzugeben, um Daten auf einer passwortgeschützten Datei speichern zu können. Das Passwort (*operand1*) 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. Weitere Informationen siehe die Statements FIND und PASSW.
CIPHER=*operand2*	Die CIPHER-Klausel gilt nur bei Adabas- und VSAM-Datenbanken. Diese Klausel wird benutzt, um einen Chiffrierschlüssel (*operand2*) bei der Aktualisierung von Daten einer Datei anzugeben, die verschlüsselt ist. Der Chiffrierschlüssel (*operand2*) kann als eine numerische Konstante mit 8 Stellen oder als eine Benutzervariable mit Format/Länge N8 angegeben werden. Weitere Informationen siehe das Statement FIND.
USING NUMBER *operand3* oder GIVING NUMBER *operand3*	Diese Klausel kann nur bei Adabas- und VSAM-Datenbanken benutzt werden. Bei VSAM-Datenbanken gilt sie nur für VSAM-RRDS, wobei die vom Benutzer angegebene RRN (Relative Record Number) der ISN entspricht. Mit dieser Klausel können Sie für einen zu speichernden Datensatz eine eigene Adabas-ISN angeben. Ist die angegebene ISN bereits vergeben, wird das Programm abgebrochen und eine entsprechende Fehlermeldung ausgegeben, es sei denn, es ist eine ON ERROR-Verarbeitung definiert.
SET/WITH *operand4=operand5*	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. 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.
USING SAME (*r*)	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 (*r*) mittels Quellcode-Zeilennummer oder Statement-Label referenziert werden. 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 für STORE-Statement

```
** 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.

132 SUBTRACT

- Funktion SUBTRACT 1074
- Syntax 1 - SUBTRACT-Statement ohne GIVING-Klausel 1074
- Syntax 2 - SUBTRACT-Statement mit GIVING-Klausel 1075
- Beispiel für SUBTRACT-Statement 1076

Dieser Abschnitt behandelt folgende Themen:

Verwandte Statements: ADD | COMPRESS | COMPUTE | DIVIDE | EXAMINE | MOVE | MOVE ALL | MULTIPLY | RESET | SEPARATE

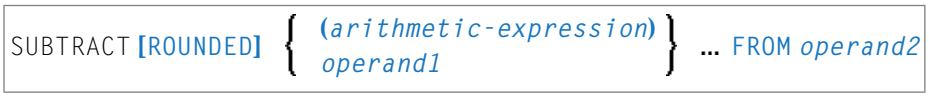
Gehört zur Funktionsgruppe: *Arithmetische Funktionen und Datenzuweisungen*

Funktion SUBTRACT

Mit dem Statement SUBTRACT können Sie die Werte zweier oder mehrerer Operanden voneinander abziehen.

Dieses Statement hat zwei verschiedene Syntax-Strukturen.

Syntax 1 - SUBTRACT-Statement ohne GIVING-Klausel



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

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
arithmetic-expression	Siehe *Arithmetischer Ausdruck* beim COMPUTE-Statement.
operand1 FROM *operand2*	**Operanden:** *operand1* ist der Minuend, *operand2* ist der Subtrahend. Somit ist das Statement gleichbedeutend mit: *operand2* := *operand2* - *operand1*

Syntax-Element	Beschreibung
	Bezüglich der Formate der Operanden siehe auch *Formatwahl im Hinblick auf die Verarbeitungszeit* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Syntax 2 - SUBTRACT-Statement mit GIVING-Klausel



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

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S	A		N			N	P	I	F		D	T					ja	nein
operand2	C	S	A		N			N	P	I	F		D	T					ja	nein
operand3		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:

Syntax Element	Description
arithmetic-expression	Siehe ***Arithmetischer Ausdruck*** beim COMPUTE-Statement.
GIVING	**GIVING-Klausel:** Wird die GIVING-Klausel benutzt, dann wird *operand2* *nicht* verändert, und das Ergebnis wird in *operand3* gespeichert.
operand1 FROM *operand2* GIVING *operand3*	**Operanden:** *operand2* ist der Minuend, *operand1* ist der Subtrahend, *operand3* ist das Ergebnisfeld. Somit ist das Statement gleichbedeutend mit: *operand3* := *operand2* - *operand1*

Syntax Element	Description
	Bezüglich der Formate der Operanden siehe auch *Formatwahl im Hinblick auf die Verarbeitungszeit* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.
ROUNDED	**ROUNDED-Option:** Wird das Schlüsselwort ROUNDED angegeben, dann wird das Ergebnis gerundet. Weitere Informationen siehe *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*.

Beispiel für SUBTRACT-Statement

```
** 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
```

133 SUSPEND IDENTICAL SUPPRESS

- Funktion SUSPEND IDENTICAL SUPPRESS 1080
- Syntax-Beschreibung SUSPEND IDENTICAL SUPPRESS 1080
- Beispiele SUSPEND IDENTICAL SUPPRESS 1081

```
SUSPEND IDENTICAL [SUPPRESS] [(rep)]
```

Dieser Abschnitt 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 SUSPEND IDENTICAL SUPPRESS

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 SUSPEND IDENTICAL SUPPRESS

Syntax-Element	Beschreibung
(*rep*)	**Report-Spezifikation:** Mit der Notation (*rep*) kann ein bestimmter anderer Report angegeben werden, auf den sich das SUSPEND IDENTICAL SUPPRESS-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 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 *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.

Beispiele SUSPEND IDENTICAL SUPPRESS

- 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`:

```
        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
                     GREGORY              FORD
                     EDWARD               ***NO CAR***
JOPER                MANFRED              ***NO CAR***
JOUSSELIN            DANIEL               RENAULT
JUBE                 GABRIEL              ***NO CAR***
JUNG                 ERNST                ***NO CAR***
JUNKIN               JEREMY               ***NO CAR***
```

134 TERMINATE

- Funktion TERMINATE 1086
- Syntax-Beschreibung TERMINATE 1086
- Kontrollübergabe nach Abbruch 1087
- Beispiel für TERMINATE-Statement 1087

```
TERMINATE [operand1 [operand2]]
```

Dieser Abschnitt behandelt folgende Themen:

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

Funktion TERMINATE

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.

Für Natural RPC: Siehe *Hinweise zu Natural-Statements auf dem Server* in der *Natural RPC (Remote Procedure Call)*-Dokumentation.

Syntax-Beschreibung TERMINATE

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
operand1	**Return Code:** *operand1* kann dazu verwendet werden, einen Return Code an das Programm zu übergeben, das die Kontrolle erhält, nachdem die Natural-Session abgebrochen wurde. Beispielsweise könnte ein Return Code an das Betriebssystem übergeben werden (über Register 15). Für *operand1* kann ein Wert von 0 bis 255 angegeben werden.
operand2	**Übergabe zusätzlicher Informationen:** *operand2* kann dazu verwendet werden, zusätzliche Informationen an das Programm zu übergeben, das nach dem Session-Abbruch die Kontrolle erhält.

Kontrollübergabe nach Abbruch

Nach dem Abbruch der Natural-Session erhält das Programm, dessen Name mit dem Profilparameter PROGRAM angegeben wurde, die Kontrolle.

Beispiel für TERMINATE-Statement

```
** 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
*
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
```

135 UPDATE

- Funktion UPDATE 1090
- Einschränkungen bei UPDATE 1091
- Datenbankspezifische Anmerkungen zu UPDATE 1091
- Syntax-Beschreibung UPDATE 1091
- Beispiel für UPDATE-Statement 1093

Structured Mode-Syntax

```
UPDATE [RECORD] [IN] [STATEMENT] [(r)]
```

Reporting Mode-Syntax

```
UPDATE    [RECORD] [IN] [STATEMENT] [(r)]
          [ SET    ]
          [ WITH   ] { SAME [RECORD]          }
          [ USING  ] { {operand1=operand2} ... }
```

Dieser Abschnitt 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 | READLOB | RETRY | STORE | UPDATELOB

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Funktion UPDATE

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 bei UPDATE

- Das UPDATE-Statement darf nicht in derselben Quellcode-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 zu UPDATE

VSAM	VSAM-Primärschlüssel können aufgrund von VSAM-Beschränkungen nicht aktualisiert werden.
SQL	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. 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. Bei den meisten SQL-Datenbanken kann eine mit FIND SORTED BY oder READ LOGICAL gelesene Reihe nicht aktualisiert werden.

Syntax-Beschreibung UPDATE

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(r)`	**Statement-Referenzierung:** Mit der Notation `(r)` können Sie das Statement referenzieren, mit dem der Datensatz, der aktualisiert werden soll, gelesen wurde. *r* kann als Statement-Label oder Quellcode-Zeilennummer angegeben werden. Falls keine Referenzierung erfolgt, bezieht sich das `UPDATE`-Statement auf die innerste aktive `READ`- bzw. `FIND`-Verarbeitungsschleife. Ist keine `READ`- oder `FIND`-Schleife aktiv, bezieht es sich auf das letzte vorhergehende `GET`-Statement (bzw. `STORE`-Statement). **Anmerkung:** Das `UPDATE`-Statement muss innerhalb der `READ`- bzw. `FIND`-Schleife stehen, auf die es sich bezieht.
`USING SAME`	**USING SAME-Klausel:** 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. Das Layout des Satzpuffers oder des Formatpuffers kann mit dem `OBTAIN`-Statement deklariert werden. 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. 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.
`SET/WITH` *operand1=operand2*	**SET/WITH-KLausel:** Mit dieser Klausel können Sie im Reporting Mode die Felder angeben, die geändert werden sollen, sowie die neuen Werte für diese Felder. 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.

Beispiel für UPDATE-Statement

```
** 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=O) /
        '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**.

136 UPDATE (SQL)

- Funktion UPDATE (SQL) 1096
- Syntax 1 - Searched UPDATE 1096
- Syntax 2 - Positioned UPDATE 1099
- Beispiele UPDATE (SQL) 1100

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Siehe auch *UPDATE - SQL* im Teil *Natural for Db2* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation:

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

Funktion UPDATE (SQL)

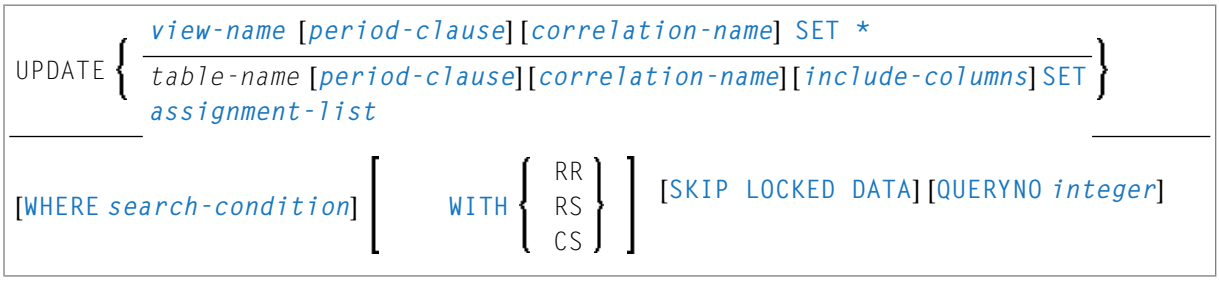
Das SQL-Statement `UPDATE` dient zum Aktualisieren von Zeilen in einer Tabelle ohne Benutzung eines Cursor (**„Searched“** `UPDATE`) oder Spalten in einer Zeile, auf die ein Cursor positioniert ist ((**„Positioned“** `UPDATE`).

Es sind zwei unterschiedliche Strukturen möglich:

Syntax 1 - Searched UPDATE

Das Searched `UPDATE`-Statement ist ein eigenständiges Statement, das ohne Referenzierung zu einem `SELECT`-Statement verwendet werden kann. Mit einem einzigen Statement können Sie keine, eine, mehrere oder alle Zeilen einer Tabelle ändern. Welche Zeilen geändert werden, bestimmen Sie mit der Suchbedingung (siehe *`search-condition`*), die auf die Tabelle angewendet wird. Optional ist es möglich, einem Tabellen- oder View-Namen einen Korrelationsnamen (siehe *`correlation-name`*) zuzuweisen.

Anmerkung: Die Anzahl der Zeilen, die mit einem Searched `UPDATE`-Statement tatsächlich geändert wurden, kann mit der Systemvariablen `*ROWCOUNT` (siehe *Systemvariablen*-Dokumentation) ermittelt werden.



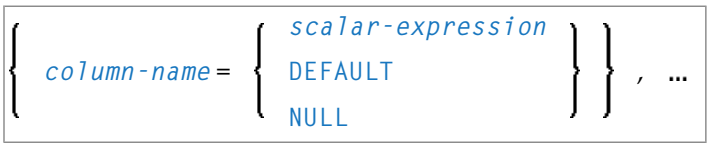
Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt ***Syntax Symbols***.

Syntax-Element-Beschreibung - Syntax 1:

Syntax-Element	Beschreibung
view-name	**View-Name:** *view-name* ist jeweils der Name eines im DEFINE DATA-Statement definierten Natural-Views. Weitere Informationen siehe *view-name* im Abschnitt ***Grundlegende Syntaxbestandteile***.
period-clause	**Period-Klausel:** Gibt an, dass für das Ziel der Aktualisierungsoperation eine Period-Klausel gilt. Weitere Informationen siehe ***Period-Klausel*** im Abschnitt ***Grundlegende Syntaxbestandteile***.
correlation-name	**Correlation-Name:** Das Element *correlation-name* ist ein Alias-Name für *table-name*. Weitere Informationen siehe *correlation-name* im Abschnitt ***Grundlegende Syntaxbestandteile***.
include-columns	**Include Columns-Klausel:** Gibt einen Satz Spalten an, die zusammen mit den *table-name*-Spalten in der Ergebnistabelle des UPDATE-Statement einbezogen werden, wenn das UPDATE-Statement in einer FROM-Klausel eines SELECT-Statement verschachtelt ist. Weitere Informationen siehe ***include-columns***.
SET	**SET-Klausel:** Wenn sich die Änderungen auf einen View beziehen, müssen Sie in der SET-Klausel einen Stern (*) angeben, weil alle Spalten des Views geändert werden müssen. Wenn sich die Änderungen auf eine Tabelle beziehen, können Sie in der SET-Klausel entweder eine Zuordnungsliste (*assignment-list*) angeben oder den Namen des View, der die zu ändernden Spalten enthält.
assignment-list	**Assignment List:** Siehe ***Assignment List*** weiter unten.
WHERE *search-condition*	**WHERE-Klausel:** In der WHERE-Klausel geben Sie die Auswahlkriterien für die Zeilen an, die geändert werden sollen. Wenn Sie keine WHERE-Klausel angeben, wird die gesamte Tabelle geändert.
WITH	**WITH - Isolation Level-Klausel:** Diese Klausel ermöglicht die explizite Angabe der beim Auffinden der zu aktualisierenden Zeile zu verwendenden Isolationsstufe. Weitere Informationen siehe ***WITH isolation-level*** in der Beschreibung des SELECT-Statement. CS — Cursorstabilität

Syntax-Element	Beschreibung	
	RR	Wiederholtes Lesen
	RS	Lesestabilität
SKIP LOCKED DATA	**SKIP LOCKED DATA-Klausel:** Gibt an, dass Zeilen übersprungen werden, wenn wegen anderer Transaktionen unverträgliche Sperren auf der Zeile liegen.	
QUERYNO *integer*	**QUERYNO-Klausel:** Diese Klausel ermöglicht Ihnen die explizite Angabe der in EXPLAIN-Ausgaben und Ablaufverfolgungssätzen für dieses Statement zu benutzenden Nummer. Die Nummer wird als QUERYNO-Spalte in der PLAN_TABLE für die Zeilen benutzt, die Informationen zu diesem Statement enthalten.	

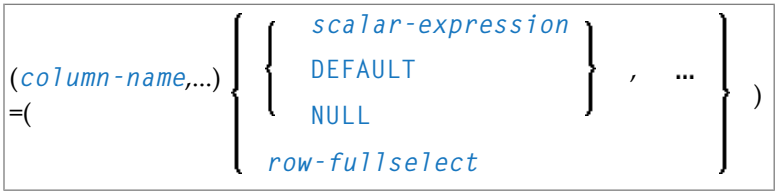
Zuordnungsliste



In einer Zuordnungsliste (*assignment-list*) können Sie einer oder mehreren Spalten Werte zuweisen. Ein Wert kann entweder ein Skalar-Ausdruck (*scalar-expression*), DEFAULT oder NULL sein. Weitere Informationen siehe ***Skalar-Ausdrücke***.

Wenn Sie NULL zuweisen, bedeutet dies, dass das betreffende Feld keinen Wert enthalten soll (auch nicht den Wert „0“ oder „leer“).

Alternative:



Syntax-Element-Beschreibung

Syntax-Element	Beschreibung
column-name	**Spaltenname:** Gibt den Namen einer Spalte der Ergebnistabelle des MERGE-Statements an, der nicht gleich dem Namen einer anderen einzubeziehenden Spalte oder einer Spalte in der Zieltabelle ist.
DEFAULT	**DEFAULT-Option:**

Syntax-Element	Beschreibung
	Gibt an, dass der Standardwert verwendet wird, je nachdem wie die entsprechende Spalte in der Tabelle definiert ist.
NULL	**NULL-Option:** Gibt den Wert NULL als neuen Wert für die Spalte an. Wenn der Wert NULL zugeordnet wurde, bedeutet dies, dass das adressierte Feld keinen Wert enthalten soll (auch nicht den Wert 0 oder „leer“.
row-fullselect	**Row Full Select-Option:** Gibt ein Full Select an, das eine einzelne Zeile zurückgibt. Die Spaltenwerte werden den entsprechenden Spaltennamen zugeordnet.

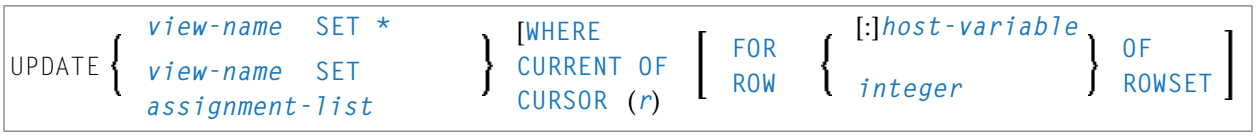
Syntax 2 - Positioned UPDATE

Das Positioned UPDATE-Statement 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 nicht-cursor-orientierter Selektion verwendet werden.

Common Set-Syntax:



Extended Set-Syntax:



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

Syntax-Element-Beschreibung - Syntax 2:

Syntax-Element	Beschreibung
view-name	**View-Name:** *view-name* ist jeweils der Name eines im DEFINE DATA-Statement definierten Natural-Views. Siehe *view-name* im Abschnitt ***Grundlegende Syntaxbestandteile***.
SET * SET *assignment-list*	**SET-Klausel:** Wenn sich die Änderungen auf einen View beziehen, müssen Sie in der SET-Klausel einen Stern (*) angeben, weil alle Spalten des Views geändert werden müssen. Wenn sich die Änderungen auf eine Tabelle beziehen, können Sie in der SET-Klausel entweder eine Zuordnungsliste (*assignment-list*) angeben oder den Namen eines Views, der die zu ändernden Spalten enthält.
WHERE CURRENT OF CURSOR (*r*)	**Statement-Referenz:** Mit der Notation (*r*) kann das Statement referenziert werden, das zur Selektion der zu ändernden Zeile benutzt wurde. Wird kein Statement angegeben, bezieht sich das UPDATE-Statement auf die jeweils innerste Verarbeitungsschleife, in der ein Datenbankdatensatz ausgewählt wurde.
FOR ROW … OF ROWSET	**FOR ROW … OF ROWSET-Klausel:** Diese Klausel gehört zum **SQL Extended Set**. Die optionale FOR ROW … OF ROWSET-Klausel für Positioned SQL UPDATE-Statements gibt an, welche Zeile der aktuellen Zeilengruppe (Rowset) aktualisiert werden muss. Sie sollte nur angegeben werden, wenn das UPDATE-Statement sich auf ein SELECT-Statement bezieht, das die Rowset-Positionierung verwendet und das Spalten-Arrays in der ***INTO-Klausel*** hat. Wenn diese Klausel weggelassen wird, werden alle Zeilen der aktuellen Zeilengruppe durch die Werte in der Zuordnungsliste (*assignment-list*) überschrieben. Diese Klausel kann nicht angegeben werden, wenn *view-name* SET * angegeben wird.

Beispiele UPDATE (SQL)

- 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
...
```

137 UPDATELOB

- Funktion UPDATELOB 1104
- Einschränkungen bei UPDATELOB 1104
- Syntax-Beschreibung UPDATELOB 1105
- Bei UPDATELOB verfügbare Systemvariable 1106
- Funktionstechnische Überlegungen zu UPDATELOB 1107
- Beispiele UPDATELOB 1107

```
UPDATELOB [OF] [RECORD] [(r)] [IN] [FILE] view-name
[PASSWORD=operand1]
[CIPHER=operand2]
[[STARTING] [AT] OFFSET [=] operand3]
[ TRUNCATE { [REMAINDER] } ]
           { [AT] OFFSET  }
```

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

Verwandte Statements: READ | FIND | GET | READLOB | UPDATE

Gehört zur Funktionsgruppe: ***Datenbankzugriffe und Datenbankänderungen***

Funktion UPDATELOB

Das Statement UPDATELOB dient zum Aktualisieren eines Datensegments eines LOB-Feldes (LOB = Large OBject) in einem Datenbankdatensatz. Die Stelle, an der die Wertänderung ausgeführt wird, kann frei gewählt werden. Der zu aktualisierende Datensatz muss zuvor mit einem FIND-, READ-, GET-Statement ausgewählt oder mit einem STORE-Statement angelegt worden sein.

Hold-Status

Die Verwendung des UPDATELOB-Statements bewirkt, dass jeder für die Verarbeitung im entsprechenden FIND-, READ-, GET-Statement gelesene Datensatz in exklusiven Hold gesetzt wird.

Weitere Informationen siehe *Datensatz-Kontrolle während einer Transaktion (Hold-Logik)* im *Leitfaden zur Programmierung*.

Einschränkungen bei UPDATELOB

Das UPDATELOB-Statement

- kann nur für den Zugriff auf Adabas-Datenbanken benutzt werden;
- darf nicht in der Zeile angegeben werden, in der das zur Auswahl des zu aktualisierenden Datensatzes benutzte Statement steht;
- ist nur für die Aktualisierung eines einzelnen LOB-Feldes verwendbar.

Syntax-Beschreibung UPDATELOB

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S				A												yes	no
operand2	C	S						N										yes	no
operand3	C	S						N	P	I		B *						yes	no

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(r)`	**Statement-Referenzierung:** Die Notation `(r)` können Sie benutzen, um das Statement anzugeben, in dem der Datensatz gelesen oder erstellt wurde. *r* kann als Statement-Label oder Quellcode-Zeilennummer angegeben werden. Sie können ein `FIND`-, `READ`-, `GET`- oder `STORE`-Statement referenzieren. Falls Sie keine Referenzierung angeben, bezieht sich das `UPDATELOB`-Statement auf die innerste aktive `READ`- bzw. `FIND`-Verarbeitungsschleife. Ist keine `READ`- oder `FIND`-Schleife aktiv, bezieht es sich auf das letzte vorhergehende `GET`-Statement. Um ein `STORE`-Statement zu referenzieren, müssen Sie immer `(r)`-Notation verwenden. **Anmerkung:** Das `UPDATELOB`-Statement muss innerhalb der `READ`- bzw. `FIND`-Statementschleife stehen, auf die es sich bezieht.
view-name	**View-Name:** Als *view-name* geben Sie den Namen eines View an, der entweder mit einem `DEFINE DATA`-Statement oder außerhalb des Programms in einer Global oder Local Data Area definiert worden sein muss. ■ Der View darf nur ein LOB-Feld mit einem einzigen Wert enthalten, zusätzliche Felder sind nicht zulässig. ■ Falls das LOB-Feld ein MU- oder PE-Feld ist, muss eine eindeutige Ausprägung angegeben sein; es ist keine Bereichsnotation zulässig. ■ Das LOB-Feld muss in dem View mit einer festen (nicht dynamischen) Länge definiert sein.
`PASSWORD=`*operand1*	**PASSWORD-Klausel und CIPHER-Klausel:**

Syntax-Element	Beschreibung
CIPHER=*operand2*	Die PASSWORD-Klausel dient zur Angabe eines Passwortes, wenn Daten aus einer passwortgeschützten Datei gelesen werden sollen. Die CIPHER-Klausel dient zur Angabe eines Chiffrierschlüssels, wenn Daten aus einer verschlüsselten Datei gelesen werden sollen. Weitere Informationen siehe Statements FIND und PASSW.
STARTING AT OFFSET=*operand3*	**STARTING AT OFFSET-Klausel:** Dient zur Angabe des Startversatzes (Offset) innerhalb des LOB-Feldes, ab wo die Operation ausgeführt werden soll. Das höchstwertige Byte des LOB-Feldes hat den Offset Null (0). *operand3* muss entweder in Form einer numerischen Konstante oder als benutzerdefinierte Variable ohne Genauigkeitsziffern angegeben werden. Das Feld wird durch die Ausführung des UPDATELOB-Statements nicht verändert. Falls der Versatzwert größer als die LOB-Feldlänge ist, wird die Lücke mit Leerzeichen aufgefüllt. Das bedeutet, dass ein LOB-Feld an einer Stelle jenseits seiner Länge aktualisiert werden kann. Wird die Klausel nicht angegeben, dann wird Null (0) als Start-Offset angenommen.
TRUNCATE REMAINDER oder TRUNCATE AT OFFSET	**TRUNCATE-Klausel:** Wenn Sie TRUNCATE REMAINDER angeben, werden die restlichen LOB-Felddaten abgeschnitten, nachdem das neue Segment in das LOB-Feld geschrieben worden ist. Dadurch wird das Ende des eingefügten Segments zum Ende des LOB-Feldes gemacht. Wenn Sie TRUNCATE AT OFFSET angeben, werden die Daten nach dem angegebenen Startversatz abgeschnitten. Es erfolgt keine Segmenteinfügung in das LOB-Feld. Danach ist die LOB-Feldlänge gleich *operand3*. Wird die Klausel weggelassen, bleiben die Daten nach dem eingefügten Segment erhalten.

Bei UPDATELOB verfügbare Systemvariable

Beim UPDATELOB-Statement steht die Natural-Systemvariable *NUMBER zur Verfügung.

Format/Länge dieser Systemvariablen ist P10. Die Format/Länge-Werte können nicht geändert werden.

Systemvariable	Erläuterung
*NUMBER	Die Systemvariable *NUMBER gibt die Summe aus Startversatz und Anzahl der eingefügten Zeichen zurück. Dieser Wert stellt den Startversatz für das nächste UPDATELOB dar, wenn ein darauffolgender Bereich des LOB-Feldes mittels mehrerer Aufrufe ersetzt wird. Die Anzahl der eingefügten Zeichen ist entweder die Byte-Länge des im View definierten LOB-Segments oder Null (0), falls die TRUNCATE AT OFFSET-Klausel angegeben wurde. Wenn es benutzt wird, muss das durch das UPDATELOB-Statement zurückgegebene *NUMBER-Feld immer mit einem Referenz-Label oder einer Referenzzeilennummer versehen sein, zum Beispiel *NUMBER(0430).

Funktionstechnische Überlegungen zu UPDATELOB

- Ein UPDATELOB bearbeitet eine Datensatz, der durch ein zugehöriges FIND-, READ-, GET- oder STORE-Statement in den Hold-Zustand gesetzt worden ist. Die Verbindung ist entweder implizit über die zurzeit aktive Referenzierung oder explizit mit (*r*)-Notation.
- Der von dem zugehörigen Statement benutzte View und der vom UPDATELOB-Statement benutzte View müssen auf dieselbe Datenbank- und Dateinummer zugreifen. Dies ist automatisch sichergestellt, wenn die Views vom selben DDM abgeleitet werden.
- Falls der in *operand3* angegebene Wert für die Einfügeposition größer als die LOB-Länge ist, wird die Lücke mit Leerzeichen aufgefüllt. Das bedeutet, dass Sie ein LOB-Feld an einer Stelle aktualisieren können, die jenseits seiner Länge liegt.
- Sie können nicht *m* Bytes durch *n* Bytes ersetzen - oder anders ausgedrückt: Es ist nicht zulässig, einen LOB-Teil durch ein Datensegment zu ersetzen, das eine andere Länge hat.
- Der durch die Systemvariable *NUMBER zurückgelieferte Wert ist die höchste Zuordnungsmarke, welche die Position innerhalb des LOB angibt, an der die letzte Einfügung endete. Wenn eine Anzahl aufeinanderfolgender Aktualisierungen gefordert wird, sollte dieser Wert immer als STARTING AT-Wert für die nächste UPDATELOB-Ausführung beibehalten werden.

Beispiele UPDATELOB

- Beispiel 1 - Neuen Datensatz abspeichern und LOB-Segment auffüllen
- Beispiel 2 - LOB-Daten stückweise zu einem vorhandenen Datensatz hinzufügen
- Beispiel 3 - LOB-Feld abschneiden

- Example 4 - LOB-Daten in existenten Datensatz lesen und LOB-Segment aktualisieren

Beispiel 1 - Neuen Datensatz abspeichern und LOB-Segment auffüllen

```
DEFINE DATA LOCAL
1 V1 VIEW OF ..
  2 PERSONNEL-ID
  2 NAME
1 V2 VIEW OF ..
  2 LOBFIELD_SEGMENT  /* LOB field defined in DDM with (A1024).
END-DEFINE
*
**=================================================
** Store new record
**=================================================

V1.PERSONNEL-ID := '12345678'
V1.NAME         := 'Smith'
LAB1.
STORE V1   /* Store new record with 2 fixed length fields.
*
MOVE ALL 'X' TO LOBFIELD_SEGMENT
**=================================================
** Update LOB field
**=================================================
UPDATELOB (LAB1.) IN FILE V2    /* Insert segment of 1KB (LOBFIELD_SEGMENT) in LOB.

     STARTING AT OFFSET = 2048  /* Store data in LOB range 2049-3072.

                               /* The first 2KB are auto filled with blanks by the
database.
END TRANSACTION
END
```

Beispiel 2 - LOB-Daten stückweise zu einem vorhandenen Datensatz hinzufügen

```
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES-V2009
  2 PERSONNEL-ID
  2 NAME
  2 L@PICTURE
1 V2 VIEW OF EMPLOYEES-V2009
  2 PICTURE_SEGMENT  /* LOB field defined in DDM with (A1024).

  2 REDEFINE PICTURE
    3 PICTURE_B (B1024)
1 #OFF   (I4)
END-DEFINE
*
**=================================================
```

```
** Read record to be updated
**================================================
LAB1.
READ (1) V1 BY PERSONNEL-ID = '60008339' /* Read record and set into exclusive hold. ↩

  RESET #OFF                  /* Start to overwrite LOB field from the beginning. ↩

  /*=================================================
  /* Read data from work file and put into LOB field
  /*=================================================
  READ WORK FILE 7 PICTURE_B  /* Start to read picture data (.jpg) from work file. ↩

LAB2.
    UPDATELOB (LAB1.) IN FILE V2
              STARTING AT OFFSET #OFF
    #OFF := *NUMBER(LAB2.)    /* Keep next position to append.
  END-WORK
END-READ
**================================================
END TRANSACTION
END
```

Beispiel 3 - LOB-Feld abschneiden

```
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES-V2009
  2 PERSONNEL-ID
  2 NAME
  2 L@PICTURE
1 V2 VIEW OF EMPLOYEES-V2009
1 V3 VIEW OF EMPLOYEES-V2009
  2 PICTURE_SEGMENT   /* LOB field defined in DDM with (A1024).                 ↩

END-DEFINE
*
**================================================
** Read record to be updated
**================================================
LAB1.
READ V1 BY PERSONNEL-ID        /* Read records.
  IF L@PICTURE > 10240 THEN    /* Check if LOB length is too high.
LAB2.
    GET V2 RECORD *ISN(LAB1.)  /* Set record to be updated into exclusive hold.
    UPDATELOB (LAB2.) IN FILE V3
              STARTING AT OFFSET 10240
              TRUNCATE AT OFFSET         /* Truncate LOB data beyond 10KB.
    END TRANSACTION
  END-IF
END-READ
END
```

Example 4 - LOB-Daten in existenten Datensatz lesen und LOB-Segment aktualisieren

```
DEFINE DATA LOCAL
1 V1 VIEW OF ..
  2 NAME
1 V2 VIEW OF ..
  2 DOCUMENT_SEGMENT      /* LOB field defined in DDM with (A100).
1 #ISN    (I4)
1 #POS    (I4)
1 #LENGTH (I4) INIT <100>
END-DEFINE
*
**=================================================
** Read record to be updated
**=================================================
INPUT (AD=T)
  / ' Read record (ISN):' #ISN
*
G1.
GET V1 RECORD #ISN          /* Get record with ISN and set into exclusive hold.  ↩

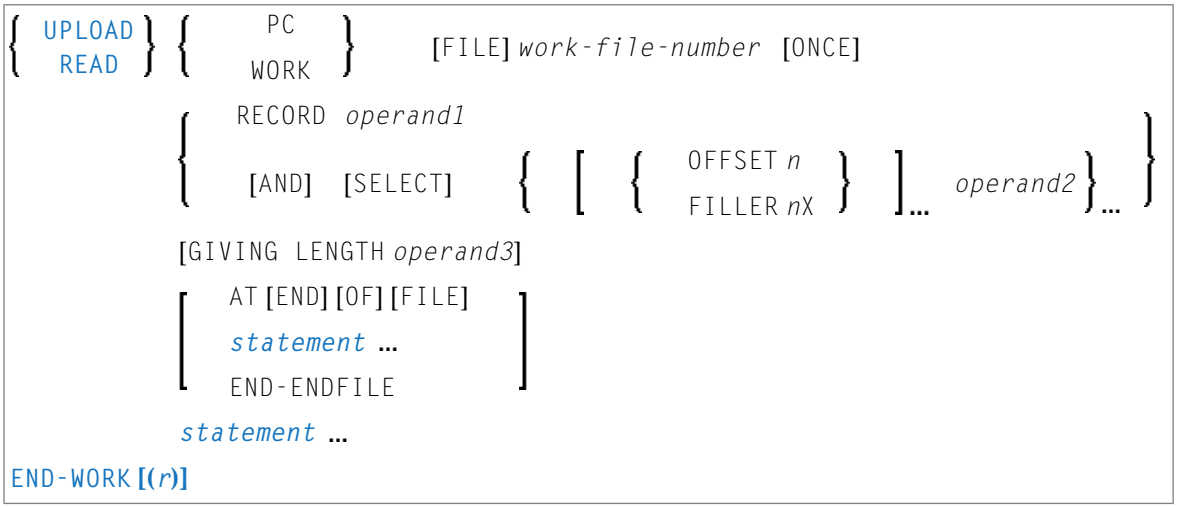
*
**=================================================
** Read LOB data and update segment of LOB field
**=================================================
R1.
READLOB V2 WITH ISN = #ISN
     STARTING AT OFFSET = 3000
   ..
   #POS := *NUMBER(R1.) - #LENGTH
   ..
   IF ..
     DOCUMENT_SEGMENT := ..
     UPDATELOB (G1.) IN FILE V2       /* Update current segment in LOB field.    ↩

                STARTING AT OFFSET #POS
   END-IF
   ..
END-READLOB
*
END TRANSACTION
END
```

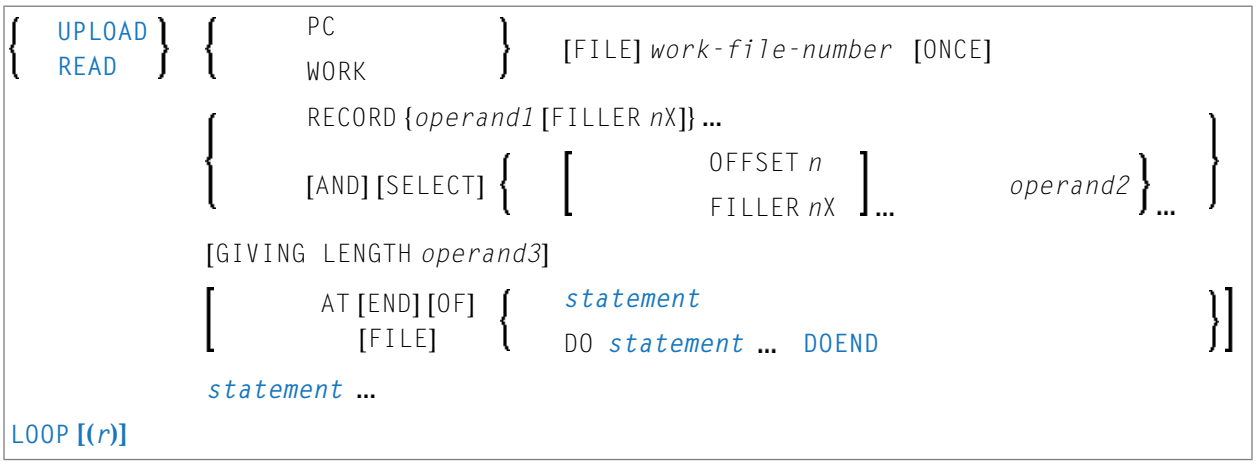
138 UPLOAD PC FILE

- Funktion UPLOAD PC FILE 1113
- Syntax-Beschreibung UPLOAD PC FILE 1113
- Beispiel für UPLOAD PC FILE-Statement 1114

Structured Mode-Syntax



Reporting Mode-Syntax



Dieser Abschnitt behandelt folgende Themen:

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

Verwandte Statements: CLOSE PC FILE | DOWNLOAD PC FILE | READ WORK FILE

Funktion UPLOAD PC FILE

Dieses Statement dient dazu, Daten vom PC zum Großrechner zu übertragen.

Siehe auch:

- *Natural Connection* und *Entire Connection*-Dokumentation
- `READ WORK FILE`-Statement

Syntax-Beschreibung UPLOAD PC FILE

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand1		S	A	G		A	U	N	P	I	F	B	D	T	L	C	ja	ja
operand2		S	A	G		A	U	N	P	I	F	B	D	T	L	C	ja	ja
operand3		S								I							ja	ja

Format C ist nicht gültig bei Natural Connection.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Zu benutzende Arbeitsdateinummer:** Diese Nummer muss der Nummer unter den Arbeitsdateinummern für den PC entsprechen, wie sie in Natural definiert sind.
operand1-2	**Feld-Spezifikation:** Mit *operand1* und *operand2* geben Sie die Felder an, die vom PC hochgeladen werden sollen. Die Felder können Datenbank- Felder oder benutzerdefinierte Variablen sein.
statement	Bei der `UPLOAD PC FILE`-Verarbeitung kann kein I/O-Statement abgesetzt werden.

`ONCE, SELECT, GIVING LENGTH`
 	
`RECORD`
	Optionen:
	
	Eine Beschreibung der Optionen `ONCE`, `SELECT`, `GIVING LENGTH` entnehmen Sie dem Statement `READ WORK FILE`.
	
	Die `RECORD`-Option ist für PC-Arbeitsdateien nicht zulässig. Sie wird zur Laufzeit zurückgewiesen.

Syntax-Element	Beschreibung
	Wenn Sie beim Hochladen von Daten ein Füllzeichen definieren möchten, müssen Sie eine Dummy-Variable anstatt der Standard-Füllzeichennotation benutzen.
END-WORK (*r*)	**Ende des UPLOAD PC FILE Statements:**
LOOP (*r*)	Im Structured Mode muss das für Natural reservierte Schlüsselwort END-WORK zum Beenden des UPLOAD PC FILE-Statements benutzt werden. Im Structured Mode können Sie bei END-WORK Labels oder Zeilennummern angeben. Im Reporting Mode wird das Natural-Statement LOOP zum Beenden des UPLOAD PC FILE-Statements benutzt. Im Reporting Mode können Sie bei LOOP Labels oder Zeilennummern angeben.

Beispiel für UPLOAD PC FILE-Statement

Das folgende Programm veranschaulicht die Benutzung das Statements UPLOAD PC FILE. Die Daten werden zunächst vom PC hochgeladen und dann auf dem Großrechner verarbeitet.

```
** Example 'PCUPEX1': UPLOAD PC FILE
**
** NOTE: Example requires that Natural Connection is installed.
** CAUTION: Executing this example will modify the database records!
************************************************************************
DEFINE DATA LOCAL
01 EMPL VIEW OF EMPLOYEES
   02 PERSONNEL-ID
   02 INCOME
      03 SALARY (1)
*
01 #PID (A8)                                  /* Personnel ID on PC
01 #NEW-INCREASE (N4)                         /* Increase for salary
END-DEFINE
*
UPLOAD PC FILE 7 #PID #NEW-INCREASE           /* Data upload
*
  FIND EMPL WITH PERSONNEL-ID = #PID          /* Data selection
    ADD #NEW-INCREASE TO SALARY (1)           /* Data update on host
    UPDATE
    END TRANSACTION
    ESCAPE BOTTOM
  END-FIND
*
```

```
END-WORK
END
```

Ausgabe des Programms `PCUPEX1`:

Wenn Sie das Programm starten, erscheint ein Fenster, in dem Sie den Namen der PC-Datei angeben, von dem die Daten hochgeladen werden sollen. Die Daten werden dann vom PC hochgeladen.

139 WRITE

- Funktion WRITE 1118
- Syntax 1 - Dynamische Formatierung 1119
- Syntax 1 - Beschreibung 1119
- Syntax 2 - Vordefinierte Form/Map benutzen 1127
- Syntax 2 - Beschreibung 1128
- Beispiele WRITE 1129

Dieser Abschnitt 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 WRITE

Das Statement `WRITE` dient dazu, Ausgaben in Freiformat zu erzeugen, die nicht bereits vorformatiert sind (vgl. `DISPLAY`-Statement).

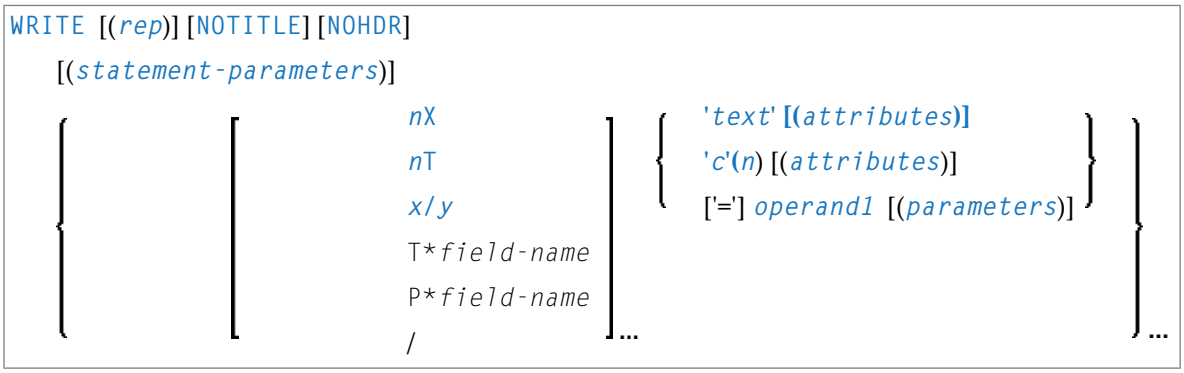
Das `WRITE`-Statement unterscheidet sich vom `DISPLAY`-Statement in folgenden Punkten:

- 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



Erläuterung der in dem Syntax-Diagramm verwendeten Symbole siehe Abschnitt *Syntax-Symbole*.

Syntax 1 - Beschreibung

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
(*rep*)	**Report-Spezifikation:** Mit der Notation (*rep*) kann ein bestimmter Report angegeben werden, wenn ein Programm mehrere Ausgaben erzeugen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem DEFINE PRINTER-Statement zugewiesen wurde, angegeben werden. Wenn (*rep*) nicht angegeben wird, bezieht sich das WRITE-Statement auf den ersten Report (Report 0). Wenn diese Druckdatei für Natural als PC definiert wird, wird der Report auf den PC heruntergeladen, siehe *Beispiel 6*. Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.

Syntax-Element	Beschreibung

NOTITLE	**Unterdrückung der Standard-Kopfzeile:**
	
	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.
	
	Beispiele:
	
	■ Ausgabe einer Standard-Kopfzeile:
	
	`WRITE NAME`
	
	■ Ausgabe einer eigenen Kopfzeile:
	
	`WRITE NAME WRITE TITLE 'user-title'`
	
	■ Ausgabe ohne Kopfzeile:
	
	`WRITE NOTITLE NAME`
	
	Anmerkung:
	
	1. Wenn die NOTITLE-Option verwendet wird, gilt sie für alle DISPLAY-, PRINT- und WRITE-Statements im selben Objekt, die Daten in 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	**Unterdrückung der Spaltenüberschrift:** 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. 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. Ohne NOHDR-Option würden auf dieser neuen Seite die DISPLAY-Spaltenüberschriften ausgegeben, mit NOHDR werden sie dort nicht ausgegeben.

Syntax-Element	Beschreibung
statement-parameters	**Parameter-Definition auf Statement-Ebene:** 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. Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Quellcode-Zeilen erstrecken. **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 *Parameter-Definition auf Element-Ebene*. Siehe auch: ■ *Liste der Parameter* ■ *Beispiel für die Benutzung von Parametern auf Statement- und Element-Ebene* ■ *Beispiel 5 - WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene*.
nX, *nT*, *x/y*, T**field-name*, P**field-name*, '=', / ,	**Notation Feld-Positionierung:** Siehe *Feld-Positionierung* im Abschnitt *Formatierung der Ausgabe*.
'text', *'c'(n)*, *attributes*, *operand1*, *parameters*	**Text/Attributzuweisung:** Siehe *Text-, Attribut-Zuweisung, Ausgabe-Elemente* im Abschnitt *Formatierung der Ausgabe*.

Liste der Parameter bei WRITE

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
BX	Box Definition	SE
CD	Color Definition	SE
CV	Control Variable	SE

Parameter, die mit dem WRITE-Statement angegeben werden können:		Spezifikation S = auf Statement-Ebene E =auf Element-Ebene
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
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

* Wenn die Anzahl der Ausprägungen eines Arrays den `PS`-Wert überschreitet, wird ein NAT0303-Fehler ausgegeben.

Ausführliche Beschreibungen der oben genannten Session-Parameter finden Sie in der *Parameter-Referenz*.

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

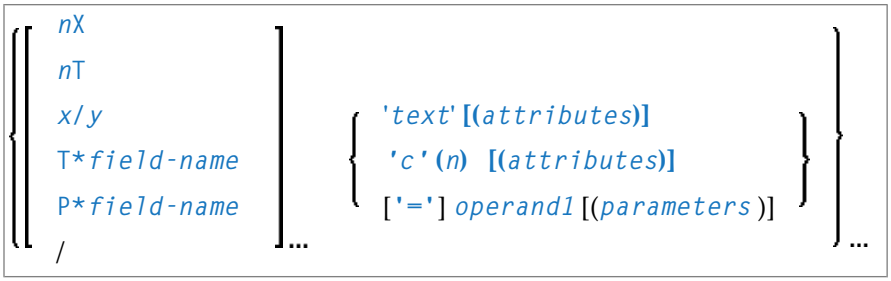
- *Spaltenüberschriften zentrieren - der HC-Parameter*
- *Breite von Spaltenüberschriften - der HW-Parameter*
- *Füllzeichen für Überschriften - die Parameter FC und GC*
- *Unterstreichungszeichen für Überschriften - der UC-Parameter*

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 (PM=I)    'Text'           VARI          /*    Text 4321
WRITE           'Text' (PM=I)    VARI (PM=I)   /*    txeT 4321
WRITE           'Text' (PM=I)    VARI          /*    txeT 1234
END
```

Siehe auch *Beispiel 5 - WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene*.

Formatierung der Ausgabe



Erläuterung der in dem Syntax-Diagramm verwendeten Symbole siehe Abschnitt *Syntax-Symbole*.

Feld-Positionierung

Syntax-Element	Beschreibung
*n*X	**Spaltenabstand:** Mit der *n*X Notation können Sie zwischen zwei Feldern *n* Leerzeichen einfügen. *n* darf nicht 0 sein. Beispiel: `WRITE NAME 5X SALARY` Siehe auch: ■ *Beispiel 2 - WRITE-Statement mit nX-, nT-Notation* (weiter unten) ■ *Spaltenabstand - der SF-Parameter und die Notation nX* (im *Leitfaden zur Programmierung*)

Syntax-Element	Beschreibung
*n*T	**Tabulator-Einstellungen:** Mit der Notation *n*T setzen Sie Tabulatoren, d.h. die Ausgabe eines Feldes beginnt ab Spalte *n*. Ein Tabulator, der bereits durch eine andere Ausgabe belegt ist, darf nicht gesetzt werden. In dem folgenden Beispiel wird das Feld NAME ab Spalte 25 und SALARY ab Spalte 50 ausgegeben: `WRITE 25T NAME 50T SALARY` Siehe auch: ■ ***Beispiel 2 - WRITE-Statement mit nX-, nT-Notation*** (weiter unten) ■ *Tabulator-Notation nT* (im *Leitfaden zur Programmierung*)
x/*y*	**x/y-Positionierung:** Mit der Notation *x*/*y* 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. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden. Siehe auch *Positionierungsnotation x/y* (im *Leitfaden zur Programmierung*).
T**field-name*	**Feldbezogene Positionierung:** Mit der Notation T* können Sie die WRITE-Ausgabe nach der Position eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (*field-name*) ausrichten. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren. Siehe auch: ■ ***Beispiel 3 - WRITE-Statement mit Notation T**** (weiter unten) ■ *Tabulator-Notation T*field* (im *Leitfaden zur Programmierung*)
P**field-name*	**Feld- und zeilenbezogene Positionierung:** Mit der Notation P* können Sie die WRITE-Ausgabe nach der Position und Zeile eines in einem vorangegangenen DISPLAY-Statement ausgegebenen Feldes (*field-name*) ausrichten. Diese Notation wird vor allem nach DISPLAY VERTICALLY-Statements verwendet. Es ist nicht erlaubt, auf eine bereits belegte Position zu positionieren. Siehe auch: ■ ***Beispiel 4 - WRITE-Statement mit Notation P**** (weiter unten) ■ *Tabulator-Notation P*field* (im *Leitfaden zur Programmierung*)
'='	**Feldinhalt hinter Feldüberschrift:**

Syntax-Element	Beschreibung
	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. Siehe auch: ■ ***WRITE-Statement mit '=', 'text', '/'*** ■ ***WRITE-Statement mit '=' und Parametern auf Statement/Element-Ebene***
`/`	**Zeilenvorschub - Schrägstrich-Notation:** Ein Schrägstrich (/) zwischen Feldern/Textelementen bewirkt einen Zeilenvorschub, d.h. die nachfolgenden Felder/ Textelemente werden in der nächsten Zeile ausgegeben. Beispiel: `WRITE NAME / SALARY` Für mehrfachen Zeilenvorschub geben Sie mehrere Schrägstriche an. Siehe auch: ■ ***WRITE-Statement mit '=', 'text', '/'*** (weiter unten) ■ *Zeilenvorschub - die Schrägstrich-Notation* (im *Leitfaden zur Programmierung*) ■ *Beispiel für Zeilenvorschub in WRITE-Statement* (im *Leitfaden zur Programmierung*)

Text-, Attribut-Zuweisung, Ausgabe-Elemente

Syntax-Element	Beschreibung
`'text'`	**Text-Zuweisung:** Der in Apostrophen stehende Text wird ausgegeben. Beispiel: `WRITE 'EMPLOYEE' NAME 'MARITAL/STATUS' MAR-STAT` Siehe auch: ■ ***WRITE-Statement mit '=', 'text', '/'*** (weiter unten) ■ *Text Notation, Mit einem Statement zu benutzenden Text definieren - die 'text'-Notation* (im *Leitfaden zur Programmierung*)
`'c'(n)`	**Zeichen-Wiederholung:**

Syntax-Element	Beschreibung

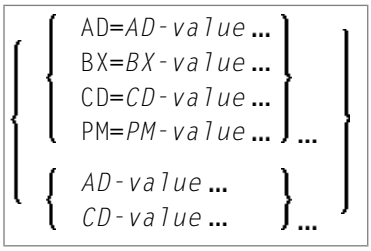
	Das in Apostrophen stehende Zeichen (**c**haracter) wird *n*-mal unmittelbar vor dem Feldwert ausgegeben.
	
	Zum Beispiel:
	
	`WRITE '*' (5) '=' NAME`
	
	führt zur Ausgabe von
	
	`***** SMITH`
	
	Siehe auch *Text-Notation, Vor einem Feldwert n mal anzuzeigendes Zeichen definieren - die 'c'(n)-Notation* (im *Leitfaden zur Programmierung*).

attributes	**Felddarstellung und Farbattribute:**
	
	Es ist möglich, den auszugebenden Feldern/Texten Anzeige- und Farbattribute zuzuordnen. Diese Attribute und die zu benutzende Syntax sind im Abschnitt *Ausgabeattribute* weiter unten beschrieben.
	
	Beispiele:
	
	`WRITE 'TEXT' (BGR)`
	`WRITE 'TEXT' (B)`
	`WRITE 'TEXT' (BBLC)`

operand1	**Name des auszugebenden Feldes:** *operand1* gibt das Feld an, dessen Inhalt an diese Stelle geschrieben wird. Arrays mit Bereichen, die es ermöglichen, die Anzahl der Ausprägungen zur Ausführungszeit zu variieren, dürfen nicht angegeben werden.
parameters	**Parameter-Definition auf Element-Ebene:** Unmittelbar nach *operand1* 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. Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine Parameterangabe darf sich jeweils nicht über zwei Quellcode-Zeilen erstrecken. Siehe auch: ■ ***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:

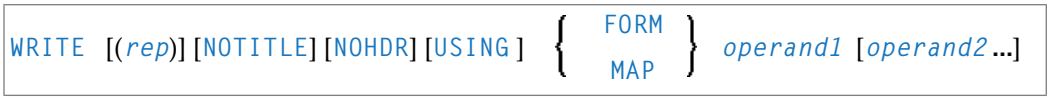


Die möglichen Parameterwerte sind in der *Parameter-Referenz* aufgeführt.

- *AD - Attribute Definition*, Abschnitt *Feldanzeige*
- *CD - Color Definition*
- *BX - Box 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



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
operand1	C	S				A												nein	nein
operand2		S	A	G	N	A	U	N	P	I	F	B	D	T	L			ja	nein

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FORM/MAP	**Benutzung des vordefinierten Form/Map-Layouts:** Diese Option verwenden Sie, wenn Sie für die Ausgabe eine (mit dem Natural-Masken-Editor (Map Editor) erstellte) Maske (Map) verwenden wollen. WRITE USING MAP bedeutet nicht, dass jedesmal, wenn die Maske ausgegeben wird, automatisch eine neue Seite ausgegeben wird. Für den Zeilenabstand muss der Parameter LS um ein Byte größer gesetzt werden als die Zeilenlänge der Map.
operand1	**Form/Map-Name:** *operand1* ist der Name der zu verwendenden Map.
operand2	**Auszugebendes Feld:** *operand2* ist der Name des auszugebenden Feldes bzw. der auszugebenden Felder. Ist *operand1* eine Konstante und wird *operand2* nicht angegeben, so werden bei der Kompilierung die Felder aus der Map-Quellcode übernommen. Die Felder müssen bezüglich Anzahl, Reihenfolge, Format, Länge und (bei Arrays) Anzahl der Ausprägungen mit den Feldern in dem referenzierten Layout bzw. der referenzierten Map übereinstimmen, andernfalls tritt ein Fehler auf.
NOTITLE/NOHDR	**Unterdrückung der Kopfzeile/Spaltenüberschrift:** Die Optionen NOTITLE und NOHDR sind unter ***Syntax 1*** des WRITE-Statements beschrieben.

Beispiele WRITE

- 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`:

```
        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-XXXXXXX)
END-READ
END
```

Ausgabe des Programms `WRTEX5`:

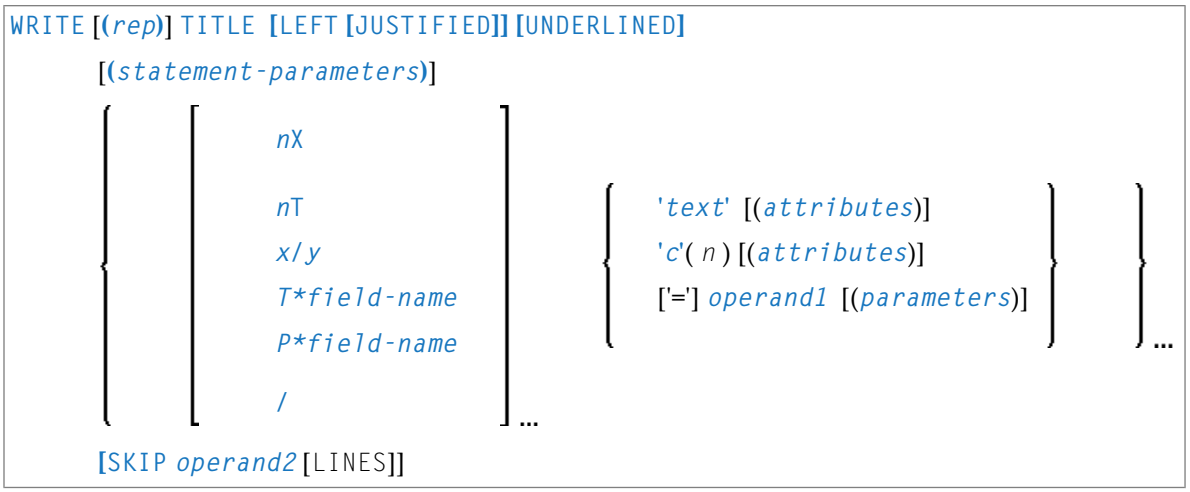
```
PERSONNEL ID: 60008339         NAME: ABELLAN          TELEPHONE: 435-6726
PERSONNEL ID: 30000231         NAME: ACHIESON         TELEPHONE: 523-341
```

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
```

140 WRITE TITLE

- Funktion WRITE TITLE 1136
- Einschränkungen bei WRITE TITLE 1137
- Syntax-Beschreibung WRITE TITLE 1137
- Beispiel für WRITE TITLE 1141



Dieser Abschnitt 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 WRITE TITLE

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 bei WRITE TITLE

- `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 WRITE TITLE

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(`*rep*`)`	**Report-Spezifikation:** Erzeugt ein Programm mehrere Reports, kann mit der Notation `(`*rep*`)` 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. Wenn `(`*rep*`)` nicht angegeben wird, bezieht sich das Statement `WRITE TITLE` auf den ersten Report (Report 0). Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.

Syntax-Element	Beschreibung
`LEFT JUSTIFIED` `UNDERLINED`	**Kopfzeilenausrichtung und Unterstreichung:** Normalerweise werden Seitenkopfzeilen zentriert und ohne Unterstreichung ausgegeben. Eine linksbündige Ausrichtung der Kopfzeile erreichen Sie durch Angabe des Schlüsselwortes `LEFT JUSTIFIED`. Eine unterstrichene Kopfzeile erhalten Sie durch Angabe des Schlüsselwortes `UNDERLINED`; als Unterstreichungszeichen wird das mit dem Parameter `UC` (auf Session-Ebene oder in einem `FORMAT`-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Kopfzeile (entsprechend der mit dem Parameter `LS` definierten Zeilenlänge). Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Zum Beispiel: Bei einer Tabulator-Notation von `10T` rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.
statement-parameters	**Parameter-Definition auf Statement-Ebene:** Unmittelbar nach dem `WRITE TITLE`-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 `GLOBALS`-Kommando, `SET GLOBALS`- (nur im Reporting Mode) oder `FORMAT`-Statement gesetzten Parameterwerten. Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Quellcode-Zeilen erstrecken. **Anmerkung:** Die hier gültigen Parameter-Einstellungen werden nur bei Variablenfeldern berücksichtigt haben und keine Auswirkung auf Textkonstanten. Wenn Sie Feldattribute für eine Textkonstante setzen möchten, müssen sie explizit für dieses Element gesetzt werden; siehe ***Parameter-Definition auf Element-Ebene***. Informationen zur Benutzung der Parameter, siehe ***Liste der Parameter*** (beim `WRITE`-Statement).
n`X` *n*`T` *x*/*y* *T*field-name* *P*field-name* /	**Format-Notation und Abstandselemente:** Siehe ***Format-Notation und Abstandselemente*** (weiter unten).
'*text*' '*c*' (*n*) *attributes*	**Text/Attribut-Zuweisung:** Siehe ***Text/Attribut-Zuweisungen*** (weiter unten).
operand1	**In der Überschrift anzuzeigendes Feld:**

Syntax-Element	Beschreibung
	Als *operand1* können Sie ein oder mehrere Feld/er angeben, die in der Kopfzeile ausgegeben werden sollen. Arrays mit Bereichen, die es ermöglichen, die Anzahl der Ausprägungen zur Ausführungszeit zu variieren, können nicht angegeben werden.
parameters	**Parameter-Definition auf Element-Ebene:** Ein einzelner oder mehrere in Anführungszeichen stehende Parameter kann/können auf Element-Ebene unmittelbar hinter *operand1* 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 Quellcode-Zeilen erstrecken. Informationen zur Benutzung der Parameter siehe ***Liste der Parameter*** (beim WRITE-Statement).
SKIP *operand2* LINES	**Einfügen von Leerzeilen nach der Kopfzeile:** 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. **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.

Format-Notation und Abstandselemente

Syntax-Element	Beschreibung
*n*X	**Spaltenabstand:** Mit dieser Notation fügen Sie *n* Leerzeichen zwischen den Spalten ein. *n* darf nicht 0 sein.
*n*T	**Setzen der Tabulatoren:** Die Notation *n*T bewirkt die Positionierung (Tabulierung) an die Druck-Position *n*. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.
x/*y*	***x*/*y*-Positionierung:** 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.

Syntax-Element	Beschreibung
	Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.
*T*field-name*	**Feldbezogene Positionierung:** Die Notation *T** bewirkt die Positionierung an eine *bestimmte Druckposition eines Feldes*, das in einem vorangegangenen DISPLAY-Statement benutzt wurde. Eine Rückwärtspositionierung ist nicht zulässig.
*P*field-name*	**Feld- und Zeilenbezogene Positionierung:** Die Notation *P** bewirkt die Positionierung an eine *spezifische Druckposition und Zeile eines Feldes*, das in einem vorangegangenen DISPLAY-Statement benutzt wurde. Sie wird am häufigsten in Verbindung mit dem vertikalen Druckmodus verwendet. Eine Rückwärtspositionierung ist nicht zulässig.
/	**Zeilenvorschub - Schrägstrich-Notation:** Ein Schrägstrich (/) bewirkt bei Platzierung zwischen Feldern oder Textelementen eine Positionierung an den Anfang der nächsten Druckzeile.

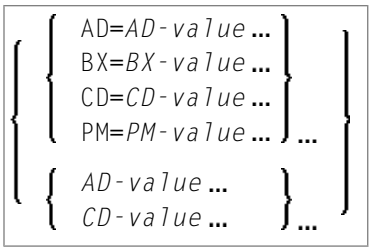
Text/Attribut-Zuweisungen

Syntax-Element	Beschreibung
'text'	**Text-Zuweisung:** Es wird die in Apostrophen stehende Zeichenkette angezeigt.
'c'(n)	**Zeichen-Wiederholung:** Das in Apostrophen stehende Zeichen (**c**haracter)wird unmittelbar vor dem Feldwert *n* mal angezeigt.

attributes	**Felddarstellung und Farbattribute:**
	
	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.
	
	Beispiele:
	
	`WRITE TITLE 'TEXT' (BGR)`
	`WRITE TITLE 'TEXT' (B)`
	`WRITE TITLE 'TEXT' (BBLC)`

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*
- *BX - Box 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 für WRITE TITLE

```
** 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`:

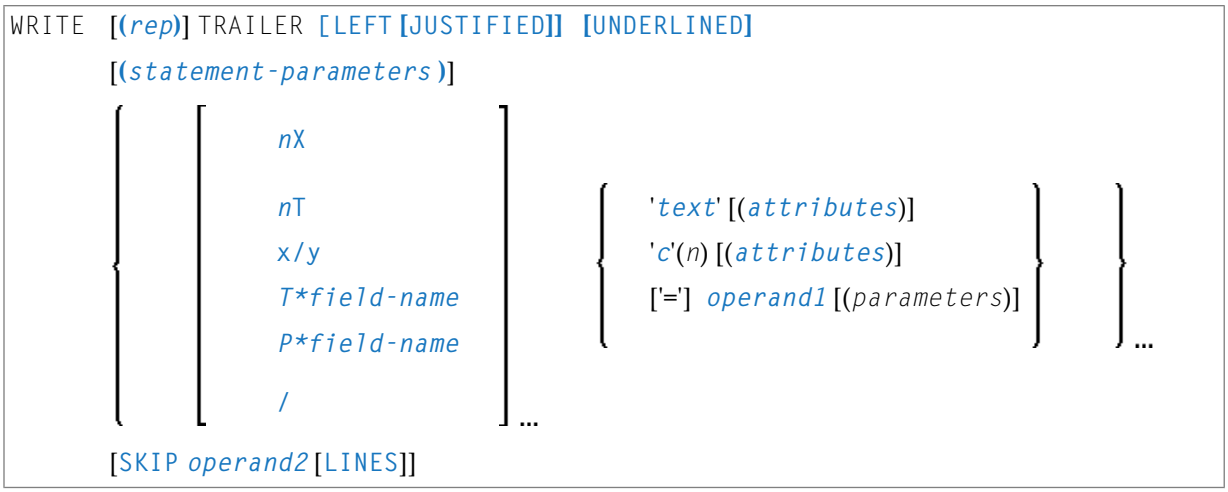
```
09:33:16.5   PEOPLE LIVING IN NEW YORK CITY           PAGE:      1
------------------------------------------------------------------------

        NAME              FIRST-NAME                CURRENT
                                                   POSITION
-------------------- --------------------   -------------------------

RUBIN                SYLVIA                SECRETARY
WALLACE              MARY                  ANALYST
```

141 WRITE TRAILER

- Funktion WRITE TRAILER 1144
- Einschränkungen bei WRITE TRAILER 1145
- Syntax-Beschreibung WRITE TRAILER 1145
- Beispiel für WRITE TRAILER 1150



Dieser Abschnitt 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 WRITE TRAILER

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 bei WRITE TRAILER

- `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 WRITE TRAILER

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`(rep)`	**Report-Spezifikation:** Erzeugt ein Programm mehrere Reports, kann mit der Notation `(rep)` ein bestimmter anderer Report angegeben werden, auf den sich das Statement `WRITE TRAILER` beziehen soll. Es kann ein Wert von 0 bis 31 oder ein logischer Name, der mit einem `DEFINE PRINTER`-Statement zugewiesen wurde, angegeben werden. Wenn `(rep)` nicht angegeben wird, bezieht sich das Statement `WRITE TRAILER` auf den ersten Report (Report 0). Informationen, wie Sie das Format eines mit Natural erstellten Ausgabe-Reports steuern, finden Sie im Abschnitt *Steuerung der Ausgabe von Daten* im *Leitfaden zur Programmierung*.
`LEFT JUSTIFIED` `UNDERLINED`	**Fußzeilenausrichtung und Unterstreichung:** Normalerweise werden Fußzeilen zentriert und ohne Unterstreichung ausgegeben. Eine linksbündige Ausrichtung der Fußzeile erreichen Sie durch die Angabe des Schlüsselwortes `LEFT JUSTIFIED`. Eine unterstrichene Fußzeile erhalten Sie durch Angabe des Schlüsselwortes `UNDERLINED`; als Unterstreichungszeichen wird das mit dem Parameter `UC` (auf Session-Ebene oder in einem `FORMAT`-Statement) definierte Zeichen verwendet. Die Unterstreichung erstreckt sich über die ganze Zeile unter der Fußzeile (entsprechend der mit dem Parameter LS definierten Zeilenlänge). Bevor die Zeile zentriert wird, führt Natural erst alle Leerstellen- und Tabulatoranweisungen aus. Bei einer Tabulator-Notation von `10T`, zum Beispiel, rückt der Text bei anschließender Zentrierung in eine Position 5 Stellen rechts von der Mitte.
statement-parameters	**Parameter-Definition auf Statement-Ebene:** Unmittelbar nach dem `WRITE TRAILER`-Statement können Sie in Klammern einzelne Session-Parameter setzen. Diese Parameterwerte haben dann für das betreffende Statement Gültigkeit vor auf übergeordneter Ebene mittels `GLOBALS`-Kommando, `SET GLOBALS`- (nur im Reporting Mode) oder `FORMAT`-Statement gesetzten Parameterwerten.

Syntax-Element	Beschreibung
	Wenn Sie mehrere Parameter angeben, müssen Sie sie durch ein oder mehrere Leerzeichen voneinander trennen. Eine einzelne Parameterangabe darf sich nicht über zwei Quellcode-Zeilen erstrecken. **Anmerkung:** Die hier gültigen Parameter-Einstellungen werden nur bei Variablenfeldern berücksichtigt, haben und keine Auswirkung auf Textkonstanten. Wenn Sie Feldattribute für eine Textkonstante setzen möchten, müssen sie explizit für dieses Element gesetzt werden; siehe *Parameter-Definition auf Element-Ebene*. Informationen zur Benutzung der Parameter siehe *Liste der Parameter* in der WRITE-Statement-Dokumentation.
*n*X *n*T *x*/*y* T**field-name* P**field-name* /	**Format-Notation und Abstandselemente:** Siehe *Format-Notation und Abstandselemente* (weiter unten).
'*text*' '*c*'(*n*) *attributes*	**Text/Attribut-Zuweisung:** Siehe *Text/Attribut-Zuweisungen* (weiter unten).
operand1	**Fußzeilen-Informationen:** Als *operand1* können Sie ein oder mehrere Feld/er angeben, die in der Fußzeile ausgegeben werden sollen. Arrays mit Bereichen, die es ermöglichen, die Anzahl der Ausprägungen zur Ausführungszeit zu variieren, können nicht angegeben werden.
parameters	**Parameter-Definition auf Element-Ebene:** Ein einzelner oder mehrere in Anführungszeichen stehende Parameter können auf Element-Ebene unmittelbar hinter *operand1* 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 Quellcode-Zeilen erstrecken. Informationen zur Benutzung der Parameter; siehe *Liste der Parameter* (beim WRITE-Statement).
SKIP *operand2* LINES	**Einfügen von Leerzeilen nach der Fußzeile:** Mit der SKIP-Klausel können Sie nach der Fußzeile Leerzeilen einfügen. Mit *operand2* geben Sie an, wieviele Leerzeilen auf die Fußzeile folgen sollen; dies kann entweder eine numerische Konstante oder der Inhalt einer numerischen Variablen sein.

Syntax-Element	Beschreibung
	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

Syntax-Element	Beschreibung
n`X`	**Spaltenabstand:** Mit dieser Notation fügen Sie *n* Leerzeichen zwischen den Spalten ein. *n* darf nicht 0 sein.
n`T`	**Setzen der Tabulatoren:** Die Notation *n*`T` bewirkt die Positionierung (Tabulierung) an die Druck-Position *n*. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.
x`/`*y*	**x/y-Positionierung:** 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. Eine Spalte, die in derselben Ausgabezeile bereits belegt ist, darf nicht angegeben werden.

Text/Attribut-Zuweisungen

Syntax-Element	Beschreibung
`'`*text*`'`	**Text-Zuweisung:** Es wird die in Apostrophen stehende Zeichenkette angezeigt.
`'`*c*`'(`*n*`)`	**Zeichen-Wiederholung:** Das in Apostrophen stehende Zeichen (**c**haracter)wird unmittelbar vor dem Feldwert *n* mal angezeigt.
attributes	**Felddarstellung und Farbattribute:** 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. Beispiele:

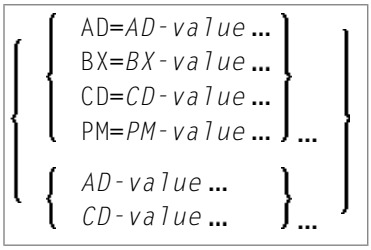
Syntax-Element	Beschreibung

	`WRITE TRAILER 'TEXT' (BGR)`
	`WRITE TRAILER 'TEXT' (B)`
	`WRITE TRAILER 'TEXT' (BBLC)`

*T*field-name*	**Feldbezogene Positionierung:** Die Notation *T** bewirkt die Positionierung an eine *bestimmte Druckposition eines Feldes*, das in einem vorangegangenen `DISPLAY`-Statement benutzt wurde. Eine Rückwärtspositionierung ist nicht zulässig.
*P*field-name*	**Feld- und Zeilenbezogene Positionierung:** Die Notation *P** bewirkt die Positionierung an eine *spezifische Druckposition und Zeile eines Feldes*, das in einem vorangegangenen `DISPLAY`-Statement benutzt wurde. Sie wird am häufigsten in Verbindung mit dem vertikalen Druckmodus verwendet. Eine Rückwärtspositionierung ist nicht zulässig.
/	**Zeilenvorschub - Schrägstrich-Notation:** Ein Schrägstrich (/) bewirkt bei Platzierung zwischen Feldern oder Textelementen eine Positionierung an den Anfang der nächsten Druckzeile.

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*
- *BX - Box 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 für WRITE TRAILER

```
** 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
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
------------------------------------------------------------------------------
```

142 WRITE WORK FILE

- Funktion WRITE WORK FILE 1154
- Syntax-Beschreibung WRITE WORK FILE 1154
- Externe Darstellung der Felder 1157
- Besonderheiten bei Systemfunktionen 1158
- Verarbeitung großer und dynamischer Variablen 1158
- Beispiele WRITE WORK FILE 1158

```
WRITE WORK [FILE] work-file-number [VARIABLE] operand1 [(parameter)] ...
```

Dieser Abschnitt 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` | `DOWNLOAD PC FILE`

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

Funktion WRITE WORK FILE

Das Statement `WRITE WORK FILE` dient dazu, Datensätze auf eine physisch-sequentielle Arbeitsdatei (Work File) zu schreiben.

Dieses Statement kann nur unter Com-plete, CICS und TSO oder im Batch-Modus verwendet werden. Entsprechend JCL- oder Systemkommandos müssen ausgeführt werden, um die Arbeitsdatei zuzuordnen. Weitere Informationen siehe *Operations*-Dokumentation. Siehe auch Profilparameter `WORK` in der *Parameter-Referenz*.

Es ist möglich, in einem Programm oder einer Verarbeitungsschleife eine Arbeitsdatei zu erstellen und diese dann in einer anderen eigenständigen Verarbeitungsschleife oder einem anderen Programm mit einem `READ WORK FILE`-Statement zu lesen.

Informationen zur Unicode- und Codepage-Unterstützung siehe *Arbeitsdateien und Druckdateien* in der *Unicode- und Codepage-Unterstützung*-Dokumentation.

Syntax-Beschreibung WRITE WORK FILE

Operanden-Definitionstabelle:

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

Anmerkung: Bei Natural Connection gilt weder Format C noch Format G.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
work-file-number	**Arbeitsdateinummer:** Gibt die für Natural definierte Nummer der Arbeitsdatei an, die gelesen werden soll. Die Nummer der Arbeitsdatei ist entweder ■ eine numerische Konstante im Wertebereich 1:32 oder ■ eine mit einer CONSTANT-Klausel definierte numerische Variable des Typs B/N/P/I, die einen Wert im Bereich 1:32 zuweist. Dezimalstellen bei den Typen (N/P) sind nicht erlaubt.
VARIABLE	**Variablen-Eintrag:** Es ist möglich, mittels verschiedener WRITE WORK FILE-Statements Datensätze mit verschiedenen Feldern in dieselbe Arbeitsdatei zu schreiben. In diesem Fall muss in allen betreffenden WRITE WORK FILE-Statements das Schlüsselwort VARIABLE angegeben werden. Die Datensätze werden dann mit variablem Format in die externe Datei geschrieben. Natural schreibt alle Ausgabedateien in variablen Blöcken (es sei denn, Sie geben in der Ausführungs-JCL Datensatzformat und Blockgröße an). 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. **Variabler Indexbereich:** Wenn Sie ein Array auf eine Arbeitsdatei schreiben, können Sie für das Array einen variablen Indexbereich angeben. Zum Beispiel: `WRITE WORK FILE work-file-number VARIABLE #ARRAY (I:J)`
operand1	**In die Arbeitsdatei zu schreibende Felder:** Als *operand1* geben Sie die Felder an, die in 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. Ein Array kann vollständig oder teilweise referenziert werden, um die Ausprägungen auszuwählen, die in die Arbeitsdatei geschrieben werden sollen. **In die Arbeitsdatei zu schreibende Gruppenoperanden:** Eine Gruppe von Feldern kann unter Angabe des Gruppennamens referenziert werden. Alle zu der referenzierten Gruppe gehörenden Felder werden in die Arbeitsdatei geschrieben. Die Reihenfolge, in der dies geschieht, wird durch die Reihenfolge der Felder in der Gruppe bestimmt. Aus einer Redefinition der referenzierten Gruppe resultierende Felder werden *nicht* in die Arbeitsdatei geschrieben. Falls die referenzierte Gruppe als Array definiert ist, werden die einzelnen

Syntax-Element	Beschreibung	

	Felder der Gruppe als Arrays in der Reihenfolge der Definition in die Arbeitsdatei geschrieben.
	
	Für die Gruppendefinition
	
	`1 GROUP1 (1:3)`
	`  2 FIELD1 (A2)`
	`  2 FIELD2 (A3)`
	`1 REDEFINE GROUP1`
	`  2 FIELD3 (A15)`
	
	ist das Statement
	
	`WRITE WORK FILE 1 GROUP1(*) ↩`
	
	gleichbedeutend mit
	
	`WRITE WORK FILE 1 GROUP1.FIELD1(*) GROUP1.FIELD2(*)`
	
	Das Statement
	
	`WRITE WORK FILE 1 GROUP1.FIELD3`
	
	ist gleichbedeutend mit
	
	`WRITE WORK FILE 1 GROUP1.FIELD1(1) GROUP1.FIELD2(1)`
	`GROUP1.FIELD1(2) GROUP1.FIELD2(2)`
	`GROUP1.FIELD1(3) GROUP1.FIELD2(3)`	

parameter	**Editiermaske-Parameter:** Als *parameter* können Sie den Session-Parameter EM bzw. EMU angeben:	
	EM=	**Editiermaske** Einzelheiten zu Editiermasken finden Sie unter dem Session-Parameter EM in der *Parameter-Referenz*-Dokumentation. Der EM-Parameter kann nicht bei Gruppenoperanden angewendet werden.
	EMU=	**Unicode-Editiermaske** Einzelheiten zu Unicode-Editiermasken finden Sie unter dem Session-Parameter EMU in der *Parameter-Referenz*-Dokumentation. Der EMU-Parameter kann nicht bei Gruppenoperanden angewendet werden.

Externe Darstellung der Felder

In der externen Datei werden Felder, die mit einem `WRITE WORK FILE`-Statement geschrieben werden, entsprechend ihrer internen Definition dargestellt, es sei denn, es wird eine Editiermaske angewendet.

- Felder ohne Eingabemasken
- Felder mit Editiermasken

Felder ohne Eingabemasken

Bei Feldern ohne Editiermasken werden die Feldwerte nicht editiert, sondern in ihrem internen Format geschrieben.

- **Format A und B**
  Die Anzahl der Bytes in der externen Datei entspricht der im Natural-Programm definierten internen Länge. Es erfolgt keine Editierung und der Wert enthält kein Dezimalzeichen.
- **Format N**
  Die Anzahl der Bytes in der externen Datei entspricht der Summe der internen Stellen vor und nach dem Dezimalzeichen. Das Dezimalzeichen ist in der Ausgabe nicht enthalten.
- **Format P**
  Die Anzahl der Bytes in der externen Datei entspricht der Summe der Stellen vor und nach dem Dezimalzeichen, plus 1 für das Vorzeichen, geteilt durch 2 und aufgerundet auf das nächste volle Byte.

Anmerkung: Bei Feldern, die ohne Editiermaske geschrieben werden, wird keine Formatkonvertierung durchgeführt.

Beispiele für Felddarstellung (ohne Editiermasken)

Felddefinition	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

Felder mit Editiermasken

Bei Anwendung einer Editiermaske wird das Feld zunächst gemäß der angegebenen Editiermaske formatiert, und der resultierende formatierte Wert wird in die Arbeitsdatei geschrieben.

- In diesem Fall wird die interne Darstellung nicht verwendet.
- Die Länge des in die Datei geschriebenen Wertes entspricht der Länge der resultierenden formatierten Zeichenkette.

Besonderheiten bei Systemfunktionen

Besonderheiten bei der Verwendung von `WRITE WORK FILE` für die Natural-Systemfunktionen `AVER`, `NAVER`, `SUM` oder `TOTAL` finden Sie unter *Format- und Längenerfordernisse bei AVER, NAVER, SUM und TOTAL* in der *Systemfunktionen*-Dokumentation.

Verarbeitung großer und dynamischer Variablen

Arbeitsdateityp	Verarbeitung
`UNFORMATTED`	Der Arbeitsdateityp `UNFORMATTED` kann zum Schreiben von Variablen benutzt werden, deren Länge die maximale Datensatzlänge überschreitet. Siehe auch den Abschnitt *Workfile-Zugriff bei dynamischen und großen Variablen* unter *Dynamische und große Variablen benutzen* im *Leitfaden zur Programmierung*.
`FORMATTED`	Eine dynamische Variable wird in ihrer aktuell definierten Länge (einschließlich Länge `0`) geschrieben.

Beispiele WRITE WORK FILE

- Beispiel 1 - WRITE WORK FILE ohne Editiermaske

- Beispiel 2 - WRITE WORK FILE mit Editiermaske

Beispiel 1 - WRITE WORK FILE ohne Editiermaske

```
** 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
```

Beispiel 2 - WRITE WORK FILE mit Editiermaske

```
** Example 'WWFEX2': WRITE WORK FILE with Edit Mask EM and EMU
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
  2 BIRTH
  2 SALARY (1)
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  WRITE WORK FILE 1
        PERSONNEL-ID NAME
        BIRTH     (EM=ZD.' 'L(10)' 'YYYY)
        SALARY(1) (EMU=999,999€)
END-FIND
*
END ↩
```

XIV Natural-SQL-Statements benutzen

Neben den „eigentlichen“ Natural-Statements, d.h., den Statements der Natural Data Manipulation Language (DML), bietet Natural „Natural SQL Statements“ für die Verwendung in Natural-Programmen, mit denen Daten verwaltet werden, die in SQL-Datenbanken enthalten sind.

Common Set und Extended Set

Grundlegende Syntaxbestandteile

Das Natural-View-Konzept

Skalar-Ausdrücke

Suchbedingungen

SELECT-Ausdrücke

Flexible SQL

Übersicht über die SQL-Statements:

```
CALLDBPROC | COMMIT | DELETE | INSERT | MERGE | PROCESS SQL | READ RESULT SET | ROLLBACK
| SELECT | UPDATE
```

143 Common Set und Extended Set

Die in Natural verfügbaren SQL-Statements umfassen zwei Syntax-Sätze:

- **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.
- **ein erweiterter (Extended Set)**
  Der Extended Set bietet darüber hinaus einige spezielle Erweiterungen zur Unterstützung von bestimmten Funktionen verschiedener von Natural unterstützter Datenbanksysteme. Die zur Verfügung stehenden Teile des Extended Set sind von Datenbanksystem zu Datenbanksystem unterschiedlich.

Das Kapitel *Natural SQL Statements* beschreibt in erster Linie den Common Set. Die Statement-Syntax entspricht weitestmöglich der in der betreffenden SQL-Literatur beschriebenen Syntax; Einzelheiten finden Sie in dieser Literatur. Einzelheiten zum Extended Set finden Sie auch in der Dokumentation der Natural-Schnittstelle zu dem von Ihnen verwendeten Datenbanksystem, siehe *Natural for Db2*.

144 Grundlegende Syntaxbestandteile

- Konstanten 1166
- Namen 1167
- Parameter 1170
- Include Columns-Klausel 1174
- Period-Klausel 1175
- Natural-Formate und SQL-Datentypen 1176

Dieser Abschnitt behandelt grundlegende Syntaxbestandteile, die dann in den Beschreibungen der einzelnen Statements nicht mehr näher erläutert werden.

Konstanten

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Konstanten sind:

constant	Das Element *constant* bezieht sich immer auf eine Natural-Konstante.
integer	Das Element *integer* 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 zu falschen Ergebnissen kommen kann.

Ungültige Syntax:	Gültige Syntax:
`VALUES (1,'A')` führt zu einem Syntaxfehler.	`VALUES (1 ,'A')`
`VALUES (1,2,3)` führt zu falschen Ergebnissen.	`VALUES (1 ,2 ,3)`

SQL-Datetime-Konstanten

Eine SQL-Datetime-Konstante ist eine Zeichenkettenkonstante mit besonderem Format, die Folgendes angibt:

`DATE` *string-constant*	SQL-Datumskonstante, zum Beispiel: `DATE '2013-15-01'`.
`TIME` *string-constant*	SQL-Uhrzeitkonstante, zum Beispiel: `TIME '10:30:15'`.
`TIMESTAMP` *string-constant*	SQL-Zeitstempelkonstante, zum Beispiel: `TIMESTAMP '2014-15-01 10:20:15.123456'`.

Informationen zu den gültigen *string-constant*-Formaten siehe IBM's *Db2 SQL reference information*

Namen

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Namen sind nachfolgend beschrieben.

- authorization-identifier
- ddm-name
- view-name
- column-name
- location-name
- table-name
- correlation-name

authorization-identifier

Ein *authorization-identifier*, der auch „creator name“ genannt wird, dient zur Qualifizierung von Datenbanktabellen und Views (Datensichten). Siehe auch *authorization-identifier* unter *table-name* weiter unten.

ddm-name

Ein *ddm-name* ist immer der Name eines mit der Natural-Utility SYSDDM erzeugten Natural Data Definition Module (DDM).

view-name

Ein *view-name* ist immer der Name eines im DEFINE DATA-Statement definierten Natural-View (Datensicht).

column-name

Ein *column-name* ist immer der Name einer physischen Datenbankspalte.

location-name

Ein *location-name* bezeichnet den Standort einer Tabelle. Die Angabe des *location-name* ist optional (gehört zum **SQL Extended Set**).

table-name

Das Element *table-name* in diesem Kapitel dient zur Referenzierung von SQL-Basistabellen und SQL-Viewed-Tabellen.

Syntax:

```
[[location-name.]authorization-identifier.]ddm-name
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ddm-name	Für jede Tabelle muss ein entsprechendes Natural-DDM existieren. Der Name dieses DDM muss mit dem Namen der entsprechenden physischen Datenbanktabelle bzw. der View (Datensicht) identisch sein.
location-name	Dieses optionale Element bezeichnet den Standort der Tabelle, auf die zugegriffen werden soll.

authorization-identifier	Es gibt zwei Arten, den *authorization-identifier* einer Datenbanktabelle bzw. einer View (Datensicht) anzugeben.
	
	Die eine Art entspricht der Standard-SQL-Syntax: *authorization-identifier* und Tabellenname werden durch einen Punkt miteinander verbunden. Hierbei muss der DDM-Name dem Namen der physischen Datenbanktabelle (ohne den *authorization-identifier*) entsprechen.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF PERSONNEL`
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM SQL.PERSONNEL`
	`...`
	
	Die andere Möglichkeit besteht darin, den *authorization-identifier* als Teil des DDM-Namens selbst zu definieren. Der DDM-Name besteht dann aus dem *authorization-identifier* gefolgt von einem Bindestrich (-) und gefolgt vom Namen der Datenbanktabelle. Intern wird der Bindestrich zwischen *authorization-identifier* und Tabellennamen in einen Punkt umgesetzt.
	
	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.

Syntax-Element	Beschreibung

	Beispiel:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF SQL-PERSONNEL`
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM SQL-PERSONNEL`
	`...`
	
	Wenn der *authorization-identifier* weder explizit noch als Teil des DDM-Namens angegeben wird, wird er vom betreffenden SQL-Datenbanksystem bestimmt.
	
	Die *table-names* können nicht nur in SELECT-Statements verwendet werden, sondern auch in den Statements DELETE, INSERT und UPDATE.
	
	Beispiel:
	
	`...`
	`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`
	`...`

correlation-name

Der *correlation-name* ist ein Alias-Name für einen *table-name*. Er kann zur Qualifizierung von Spaltennamen verwendet werden. Außerdem dient er dazu, implizit Felder in einer Natural-View (Datensicht) zu qualifizieren, der in der INTO-Klausel eines SELECT-Statements verwendet wird.

Beispiel:

```
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
...
```

Die Verwendung von *correlation-names* ist zwar in der Regel nicht nötig, kann aber helfen, die Lesbarkeit eines Statements zu erleichtern.

Parameter

Syntax des Elements *parameter*:

```
[[:sql-type]:] host-variable [INDICATOR [:] host-variable] [LINDICATOR [:] host-variable]
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
sql-type	Ein *sql-type* gibt den SQL-Datentyp der *host-variable* an, wenn sie für den Zugriff auf Db2 verwendet wird. Die Angabe des *sql-type* ist optional, weil die meisten SQL-Datentypen implizit zu Natural-Host-Variablen zugeordnet werden. Für einige Natural-Host-Variablen kann der SQL-Datentyp jedoch nicht implizit zugeordnet werden. *sql-type* gehört zum **SQL Extended Set**. Wenn ein *sql-type* angegeben wird, muss er mit Doppelpunkten (:) umgeben sein. Gültige SQL-Datentypen sind:

sql-type	Natural-Format	Db2 SQL-Datentyp
BLOBFILE	group(I4,I4,I4,A255)	BLOB file reference(916/917)
CLOBFILE	group(I4,I4,I4,A255)	CLOB file reference(920/921)
DBCLOBFILE	group(I4,I4,I4,A255)	DBCLOB file reference(924/925)
BLOBLOC	(I4)	BLOB locator960/961)
CLOBLOC	(I4)	CLOB locator(964/965)
DBCLOBLOC	(I4)	DBCLOB locator(968/969)

Siehe auch *Natural-Formate und SQL-Datentypen*.

Syntax-Element	Beschreibung
host-variable	Eine *host-variable* ist eine in einem SQL-Statement referenzierte Natural-Programmvariable (keine Systemvariable), die entweder ein eigenständiges Feld oder Teil einer View (Datensicht) sein kann. Wenn sie als empfangendes Feld (z.B. in einer INTO-Klausel) definiert wird, ist die *host-variable* ein Feld, dem vom Datenbanksystem einen Wert zugeordnet wird. Wenn sie als sendendes Feld (z.B. in einer WHERE-Klausel) definiert wird, ist die *host-variable* ein Feld, dessen Wert vom Programm an das Datenbanksystem übergeben wird. Siehe auch ***Natural-Formate und SQL-Datentypen***.

[:]	**Doppelpunkt:**
	
	Gemäß den SQL-Standards kann einer *host-variable* ein Doppelpunkt (:) vorangestellt werden. Bei der Verwendung mit flexibler SQL muss ihr ein Doppelpunkt vorangestellt werden.
	
	Beispiel:
	
	`SELECT NAME INTO :#NAME FROM PERSONNEL`
	`WHERE AGE = :VALUE`
	
	Wenn ein Variablenname mit einem für SQL reservierten Wort identisch ist, ist der Doppelpunkt ebenfalls erforderlich. In Situationen, in denen entweder eine *host-variable* oder eine Spalte referenziert werden kann, wird ein Name ohne Doppelpunkt als Referenz auf eine Spalte interpretiert.

INDICATOR	**INDICATOR-Klausel:**
	
	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.
	
	Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, dient die INDICATOR *host-variable* (Null-Indikatorfeld) dazu, herauszufinden, ob eine zu lesende Spalte „Null“ ist.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 NAME      (A20)`
	`1 NAMEIND   (I2)`
	`END-DEFINE`
	`SELECT *`
	`  INTO NAME INDICATOR NAMEIND`
	`...`
	
	In diesem Beispiel ist NAME die empfangende *host-variable* und NAMEIND das Null-Indikatorfeld.

Syntax-Element	Beschreibung

	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 als oder gleich Null (0).
	
	Wenn sie mit einer sendenden *host-variable* (Quelldatenfeld) verwendet wird, dient das Null-Indikatorfeld dazu, diesem Feld einen Nullwert zuzuweisen.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 NAME (A20)`
	`1 NAMEIND (I2)`
	`UPDATE ...`
	`SET NAME = :NAME INDICATOR :NAMEIND`
	`WHERE ...`
	
	In diesem Beispiel ist `:NAME` die sendende *host-variable* und `:NAMEIND` das Null-Indikatorfeld. Durch Eingabe eines negativen Wertes für das Null-Indikatorfeld wird der Datenbankspalte ein Nullwert zugeordnet.
	
	Eine `INDICATOR` *host-variable* hat Format/Länge I2.

`LINDICATOR`	**LINDICATOR-Klausel:**
	
	Diese Klausel ist optional und dient zur Unterstützung von Spalten mit variabler Länge, z.B. des Typs `VARCHAR` oder `LONG VARCHAR`.
	
	Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, enthält die `LINDICATOR` *host-variable* (Längen-Indikatorfeld) die Anzahl der tatsächlich von der Datenbank in das Zielfeld geschriebenen Zeichen. Das Zielfeld wird immer mit Leerzeichen aufgefüllt.
	
	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.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 ADDRESSLIND (I2)`
	`1 ADDRESS (A50/1:6)`
	`END-DEFINE`
	`SELECT *`
	`  INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND`
	`  ...`
	
	In diesem Beispiel erhält `:ADDRESS(*)` die ersten 300 Bytes (falls vorhanden) der adressierten `VARCHAR`- bzw. `LONG VARCHAR`-Spalte, und `:ADDRESSLIND` ist das Längen-Indikatorfeld, das die Anzahl der tatsächlich von der Datenbank zurückgelieferten Zeichen enthält.

Syntax-Element	Beschreibung

	Wenn es mit einer sendenden *host-variable* (Quellendatenfeld) verwendet wird, gibt das Längen-Indikatorfeld an, wieviele Zeichen des Quellendatenfeldes an die Datenbank übergeben werden sollen.
	
	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 ist, 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).

Include Columns-Klausel

Diese Klausel gehört zum **SQL Extended Set**. Sie steht in den folgenden Statements zur Verfügung: DELETE, INSERT, MERGE und UPDATE.

Syntax der *include-columns*-Klausel:

```
INCLUDE (column-name data-type,...)
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
INCLUDE	Mit dem Schlüsselwort INCLUDE wird eine Liste mit Spalten eingeleitet, welche in die Ergebnistabelle eines DELETE-, INSERT-, MERGE- oder UPDATE-Statements aufgenommen werden soll. INCLUDE kann nur dann angegeben werden, wenn ein DELETE-, INSERT-, MERGE- oder UPDATE-Statement in der FROM-Klausel eines SELECT-Statements verschachtelt ist.
column-name	Gibt den Namen einer Spalte der Ergebnistabelle des MERGE-Statements an, der nicht derselbe Name ist wie der einer anderen *include-column* oder einer Spalte in der Zieltabelle.
data-type	Gibt den Datentyp der *include-column* an. Siehe *data-type* weiter unten.

data-type

```
{ built-in-type  }
{ distinct-type  }
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
built-in-type	Dient zur Angabe eines *built-in-type*-Datentyps. Beschreibung der *built-in-types* siehe IBM *Db2 for z/OS*-Dokumentation.
distinct-type	Dient zur Angabe eines *distinct-type*-Datentyps.

Period-Klausel

Diese Klausel gehört zum **SQL Extended Set**. Sie steht in den folgenden Statements zur Verfügung: **Searched** `DELETE` und **Searched** `UPDATE`.

Syntax:

```
FOR PORTION OF BUSINESS_TIME FROM expr1 TO expr2
```

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
`FOR PORTION OF BUSINESS_TIME`	Gibt an, dass das `DELETE`- oder `UPDATE`-Statement nur für Zeilenwerte für den Teil des `BUSINESS_TIME`-Zeitraums in der Zeile gilt, der durch die Period-Klausel angegeben wird. `BUSINESS_TIME` muss ein Zeitraum sein, der für die im `DELETE`- und `UPDATE`-Statement referenzierte Tabelle definiert ist.
`FROM expr1 TO expr2`	Gibt an, dass die Aktualisierung für Spalten für den Zeitraum gilt, der mit `FROM expr1 TO expr2` angegeben wird. Wenn der Zeitraum, der durch den Startwert und den Endewert für die `BUSINESS_TIME` einer Zeile angegeben ist, vollständig in dem angegebenen Zeitraum enthalten ist (wenn der Startwert für den Zeitraum in der Zeile kleiner ist als *expr2* und der Endewert für den Zeitraum in der Zeile größer ist als *expr1*), wird diese Zeile aktualisiert bzw. gelöscht, und die Start- und Endewerte für diesen `BUSINESS_TIME`-Zeitraum bleiben unverändert. Wenn der Zeitraum, der durch den Startwert und den Endewert für die `BUSINESS_TIME` einer Zeile angegeben ist, nur teilweise in dem angegebenen Zeitraum enthalten ist (wenn der Startwert für den Zeitraum in der Zeile größer als *expr2* oder der Endewert für den Zeitraum in der Zeile kleiner als *expr1* ist), wird diese Zeile aktualisiert oder gelöscht, und anschließend werden eine oder zwei zusätzliche Zeilen eingefügt. Die eingefügten Zeilen stellen die ursprünglichen Zeilenwerte für die Zeiträume dar, die durch den Aktualisierungsvorgang nicht aktualisiert oder gelöscht wurden. Für die eingefügten Zeilen werden der Startwert und der Endewert für die `BUSINESS_TIME` so gesetzt, dass entweder der Startwert für die `BUSINESS_TIME` der Startwert für die `BUSINESS_TIME` der ursprünglichen Zeile und der Endewert expr1 ist, oder der Startwert ist *expr2* und der Endewert ist der Endewert für die `BUSINESS_TIME` der ursprünglichen Zeile.
expr1 und *expr2*	Geben Sie Ausdrücke an, die einen Wert eines *built-in*-Datentyps zurückliefern. Das Ergebnis eines jeden Ausdrucks muss mit dem Datentyp der Spalten des angegebenen Zeitraums vergleichbar sein. Ein Zeitstempel mit `TIME ZONE` ist als der Ergebnisdatentyp für *expr1* oder *expr2* nicht zulässig.

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*, A DYNAMIC	CHAR (*n*) , VARCHAR(*n*), CLOB(*n*)
B2	SMALLINT
B4	INT
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
N*nn.m*	NUMERIC (*nn*+*m*,*m*)
P*nn.m*	NUMERIC (*nn*+*m*,*m*)
T, A8	TIME
T (COMPOPT DB2TSTI=ON)	TIMESTAMP
D, A10	DATE
A26	TIMESTAMP
A19	TIMESTAMP(*0*)
A20+*n*	TIMESTAMP(*n*) (1<=*n*<=12)
A25	TIMESTAMP(*0*) WITH TIMEZONE
A26+*n*	TIMESTAMP(n) WITH TIMEZONE (1<=*n*<=12)
G*n*; nur für View-Felder	GRAPHIC (*n*)
U*n*, U DYNAMIC	GRAPHIC (*n*) ,VARGRAPHIC(*n*), DBCLOB(*n*) CCSID 1200
B*n*, B DYNAMIC	BINARY(*n*), VARBINARY(*n*), BLOB(*n*)
P19.0	BIGINT
F8	DECFLOAT(*n*)
A DYNAMIC, B DYNAMIC, U DYNAMIC	XML
Gruppenstruktur (I4,I4,I4,A255) prefixed with :BLOBFILE:	BLOB-Datei-Referenz
Gruppenstruktur (I4,I4,I4,A255) prefixed with :CLOBFILE:	CLOB-Datei-Referenz
Gruppenstruktur (I4,I4,I4,A255) prefixed with :DBCLOBFILE:	DBCLOB-Datei-Referenz
I4 mit Präfix :BLOBLOC:	BLOB-Lokator

Natural-Format/Länge	SQL-Datentyp
I4 mit Präfix :CLOBLOC:	CLOB-Lokator
I4 mit Präfix :DBCLOBLOC:	DBCLOB-Lokator

Natural überprüft nicht, ob der konvertierte 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 einem Stern (*) referenziert werden. Der entsprechende SQL-Datentyp ist `CHAR (n)`, wobei *n* die Gesamtanzahl der Bytes im Array 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 bei Natural for Db2 verwendet werden. Sie werden in die entsprechenden datenbank-spezifischen Formate Db2 `DATE` und `TIME` umgesetzt.

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.

145 Das Natural-View-Konzept

Einige Natural-SQL-Statements erlauben auch die Verwendung von Natural-Views (Datensichten).

Eine Natural-View kann anstelle einer Parameterliste angegeben werden, wobei jedes Feld in der View (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 in der View jeweils unmittelbar vor dem Feld, auf das es sich bezieht, definiert werden.

Beispiel:

```
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
```

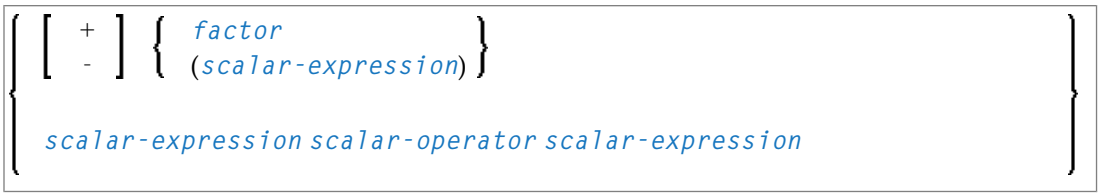
```
   ...
END-SELECT
```

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
```

146 Skalar-Ausdrücke

- scalar-expression 1182
- scalar-operator 1182
- factor 1183
- row-value-expression 1194



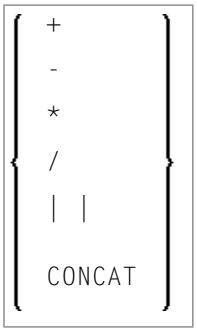
scalar-expression

Ein *scalar-expression* besteht aus einem *factor* und anderen *scalar-expressions* einschließlich *scalar-operators*.

Hinsichtlich der 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*).

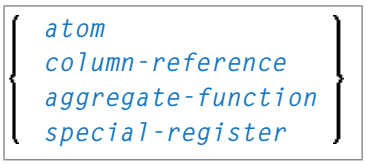
scalar-operator



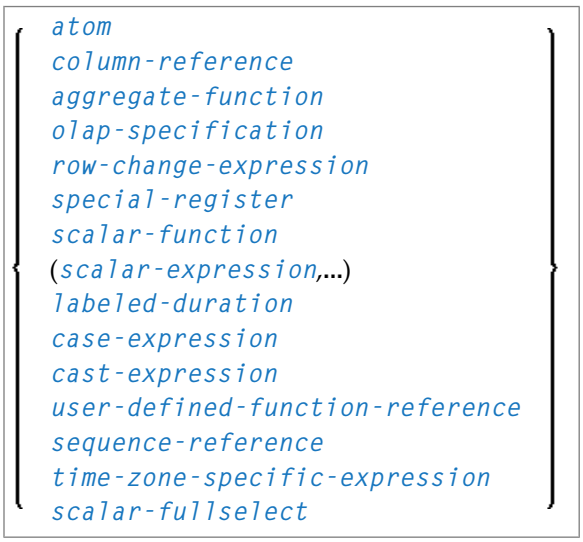
Ein *scalar-operator* kann jeder der oben aufgeführten Operatoren sein. Die Operatoren Minus (-) und Schrägstrich (/) müssen durch mindestens ein Leerzeichen von den vorangehenden Operatoren getrennt sein.

factor

Common Set-Syntax:



Extended Set-Syntax:



Ein *factor* kann eines der obigen Elemente sein, die im Folgenden beschrieben sind.

atom



Ein *atom* kann entweder ein *parameter* oder eine Konstante (*constant*) sein.

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***).

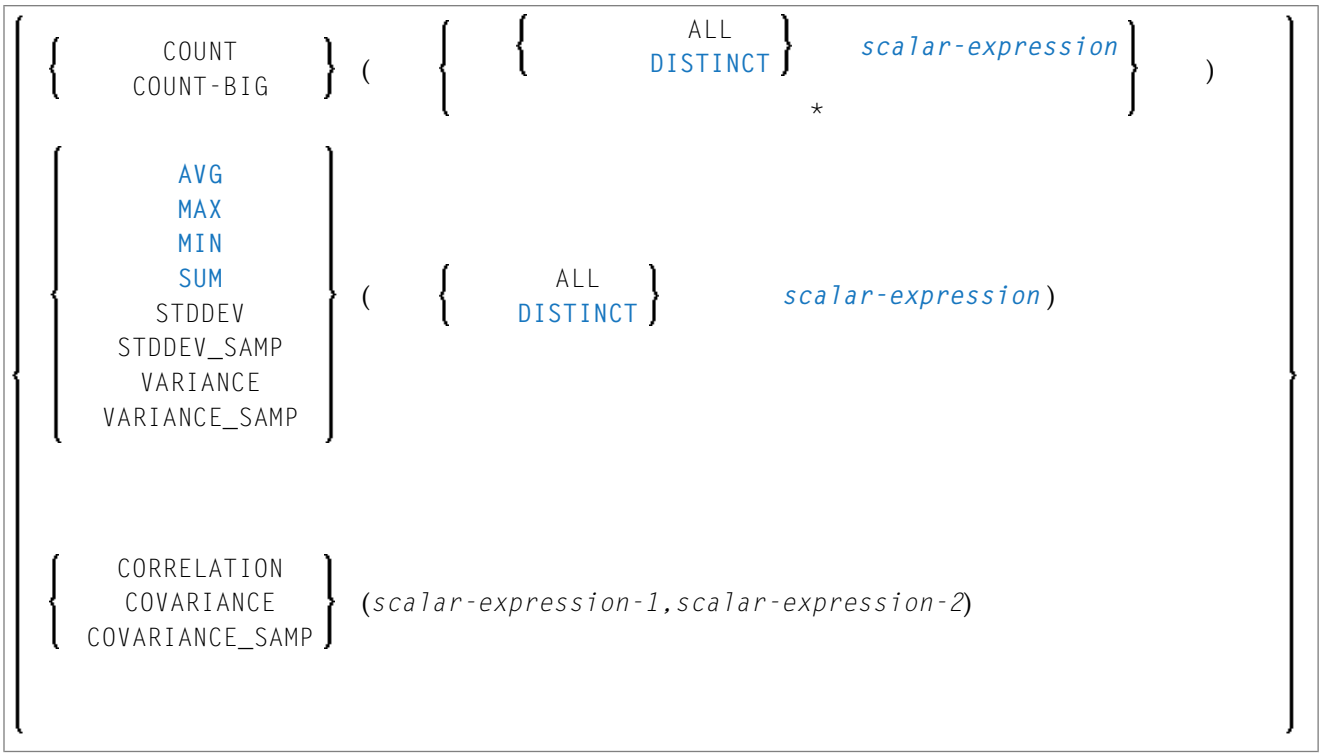
aggregate-function

Common Set-Syntax:

```
{ COUNT    { (*)                          }
           { (DISTINCT column-reference)  }

{ { AVG }
{ { MAX }  { (DISTINCT column-reference)  }
{ { MIN }  { ([ALL] scalar-expression)    }
{ { SUM }
```

Extended Set-Syntax:



SQL bietet eine Anzahl 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
  ...
```

DISTINCT

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 ein allgemeiner *scalar-expression* sein.

`DISTINCT` ist nicht erlaubt bei der Funktion `COUNT(*)`, mit der alle Zeilen in einer Tabelle ohne Eliminierung doppelt vorkommender Zeilen gezählt werden.

ROW CHANGE-expression



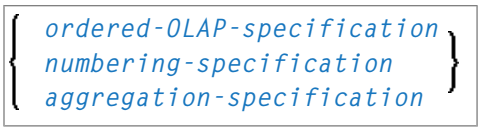
Ein `ROW CHANGE`-Ausdruck liefert eine Zeichenfolge (Token) oder einen Zeitstempel, das bzw. der für die letzte Änderung an einer Zeile steht.

`TIMESTAMP`	Gibt an, dass ein Zeitstempel zurückgegeben wird, der für die letztmalige Änderung an einer Zeile steht.
`TOKEN`	Gibt an, dass einen Zeichenfolge (Token) des Typs `BIGINT` zurückgegeben wird, das für einen relativen Punkt in der Änderungsabfolge einer Zeile steht.
`FOR` *table-designator*	Bezeichnet die Tabelle, in welcher der Ausdruck referenziert wird. *table-designator* muss ein gültiges Natural SQL DDM sein.

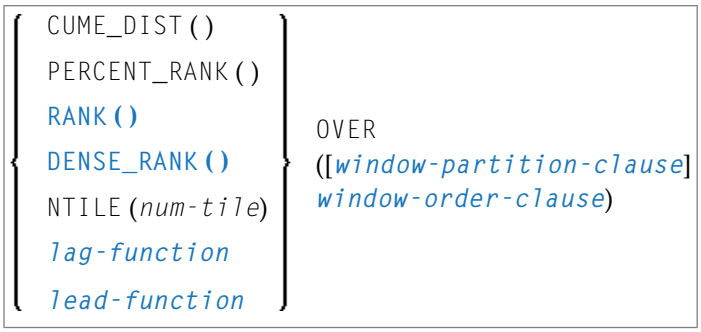
OLAP-specification

Für die folgenden Klauseln des *OLAP-expression* ist der IBM Db2 Analytics Accelerator for z/OS erforderlich:

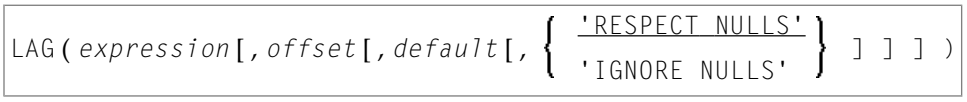
```
CUME_DIST
PERCENT_RANK
NTILE
LAG
LEAD
FIRST_VALUE
LAST_VALUE
NTH_VALUE
RATIO_TO_REPORT
```



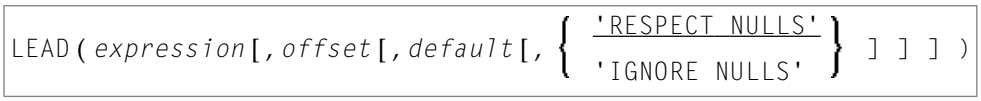
ordered-OLAP-specification



lag-function



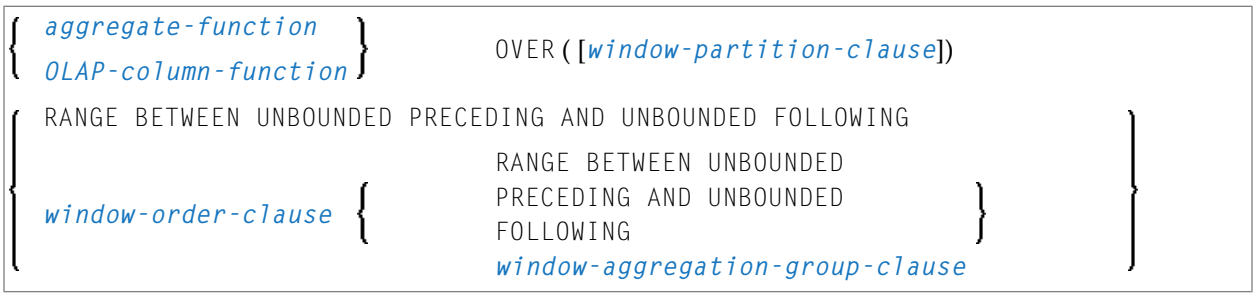
lead-function



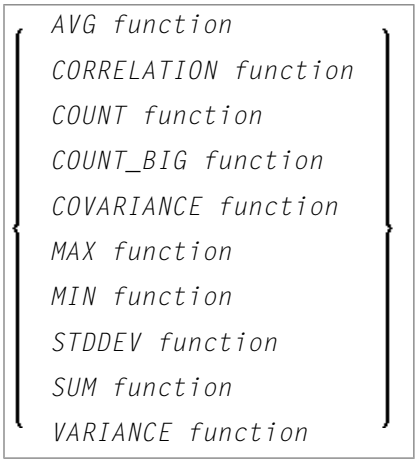
numbering-specification



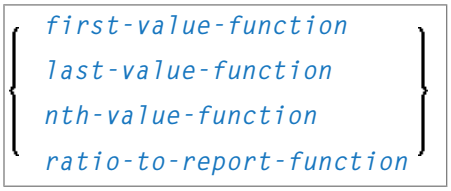
aggregation-specification



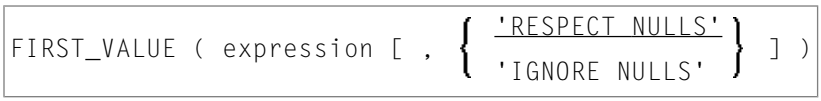
aggregate-function



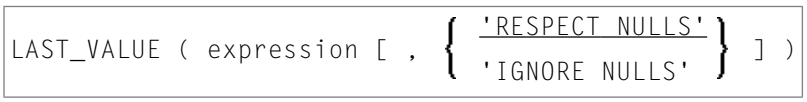
OLAP-column-function



first-value-function



last-value-function



nth-value-function



ratio-to-report-function

```
RATIO_TO_REPORT ( expression )
```

window-aggregation-group-clause

```
{ ROWS  } { group-start   }
{ RANGE } { group-between }
          { group-end     }
```

group-start

```
{ UNBOUNDED PRECEDING          }
{ unsigned-constant PRECEDING  }
{ CURRENT ROW                  }
```

group-between

```
BETWEEN group-bound-1 AND group-bound-2
```

group-bound-1

```
{ UNBOUNDED PRECEDING          }
{ unsigned-constant PRECEDING  }
{ unsigned-constant FOLLOWING  }
{ CURRENT ROW                  }
```

group-bound-2

```
{ UNBOUNDED FOLLOWING          }
{ unsigned-constant PRECEDING  }
{ unsigned-constant FOLLOWING  }
{ CURRENT ROW                  }
```

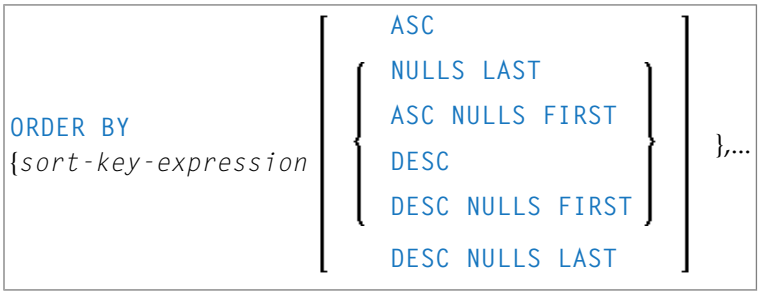
group-end

```
{ UNBOUNDED FOLLOWING          }
{ unsigned-constant FOLLOWING  }
```

window-partition-clause



window-order-clause



OLAP-Angaben (Online Analytical Processing) bieten die Möglichkeit, im Ergebnis einer Abfrage Informationen zur Rangfolge, Zeilennummerierung und Aggregation als Skalarwert zurückzugeben. Eine OLAP-Angabe kann enthalten sein in einem Ausdruck, in einer *select-list* oder in der ORDER BY-Klausel eines SELECT-Statements. Das Abfrageergebnis, auf das die OLAP-Angaben angewendet werden, ist die Ergebnistabelle der innersten Unterauswahl (*subselect*), in der die OLAP-Angaben enthalten sind.

RANK	Gibt an, dass der Rang einer Zeile definiert ist als 1 plus Anzahl der Zeilen, die der Zeile unmittelbar vorangehen.
DENSE_RANK	Gibt an, dass der Rang einer Zeile definiert ist als 1 plus Anzahl der vorangehenden Zeilen, die sich hinsichtlich der Reihenfolge unterscheiden.
ROW_NUMBER	Gibt an, dass Zeilenfolgenummer für die Zeile berechnet wird, die durch die Reihenfolge definiert ist, wobei mit der 1 für die erste Zeile begonnen wird.
PARTITION BY	Definiert die Partition, in der die OAP-Operation angewendet wird.
ORDER BY	Definiert die Reihenfolge von Zeilen innerhalb einer Partition, die zur Bestimmung des Werts der OLAP-Angabe verwendet wird.
ASC	Gibt an, dass die *sort-key-expression*-Werte in aufsteigender Reihenfolge verwendet werden.
DESC	Gibt an, dass die *sort-key-expression*-Werte in absteigender Reihenfolge verwendet werden.
NULLS_FIRST	Gibt an, dass hinsichtlich der Window-Reihenfolge Nullwerte vor allen Werten ungleich Null bei der Sortierreihenfolge berücksichtigt werden.
NULLS LAST	Gibt an, dass hinsichtlich der Window-Reihenfolge Nullwerte nach allen Werten ungleich Null bei der Sortierreihenfolge berücksichtigt werden.

Beispiel:

Das folgende Beispiel zeigt die Rangordnung der Mitarbeiter, sortiert nach Nachnamen, die ein Gesamtgehalt von mehr als $30,000 haben.

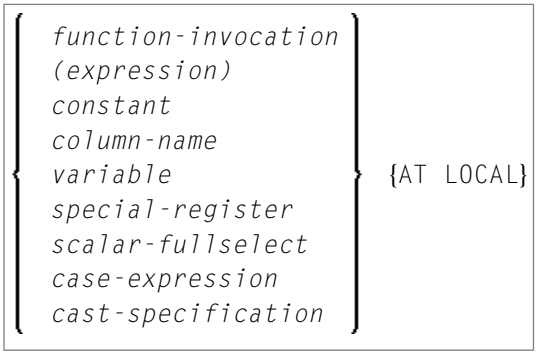
```
SELECT EMPNO, LASTNAME, FIRSTNME, SALARY+BONUS AS TOTAL_SALARY,
  RANK() OVER(ORDER BY SALARY+BONUS DESC) AS RANK_SALARY
  FROM DSN8910-EMP WHERE SALARY+BONUS > 30000
  ORDER BY LASTNAME;
```

time-zone-specific-expression

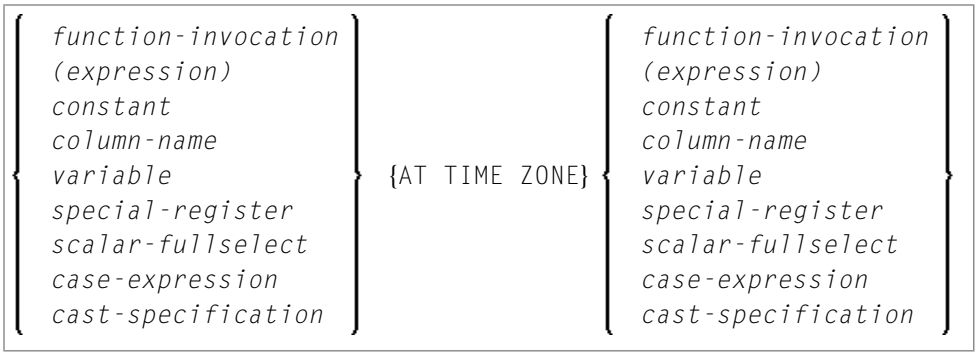
time-zone-specific-expression

Gibt einen Zeitstempel mit Zeitzonenwert an: `AT LOCAL` oder `AT TIME ZONE`

`AT LOCAL`



`AT TIME ZONE`



special-register

Eine Referenz auf ein *special-register* gibt einen Skalarwert zurück.

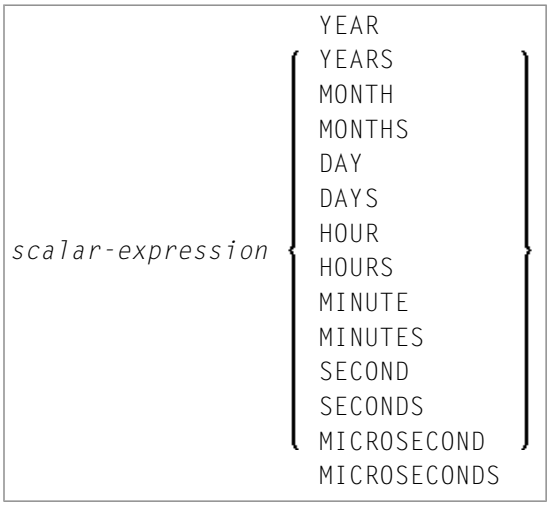
Weitere Informationen zu den von Natural unterstützten speziellen Registern siehe *special-register* im Abschnitt *Gemeinsame syntaktische Elemente für Natural SQL-Statements* in der *Datenbankmanagementsystem-Schnittstellen*-Dokumentation.

scalar-function

Eine *scalar-function* ist eine eingebaute Funktion, die zur Erstellung skalarer Berechnungsausdrücke benutzt werden kann.

Informationen zu den Skalarfunktionen, die vom Natural **SQL Extended Set** unterstützt werden, siehe *Gemeinsame syntaktische Elemente für Natural SQL-Statements*, *scalar-function* in der *Natural for Db2*-Dokumentation.

labeled-duration



Eine *labeled-duration* zeigt eine spezifische Zeiteinheit an, die durch eine Zahl ausgedrückt wird, welche ein Ausdruck sein kann, auf den eines der Schlüsselwörter für die Dauer folgt.

labeled-duration entspricht nicht dem Standard SQL und wird daher nur vom Natural **SQL Extended Set** unterstützt.

case-expression



case-expression entspricht nicht dem Standard SQL und wird daher nur vom Natural **SQL Extended Set** unterstützt.

WHEN-Klausel mit Suchbedingung

```
WHEN search-condition THEN { NULL
                           { scalar-expression }
```

Eine WHEN-Klausel mit Suchbedingung entspricht nicht dem Standard SQL und wird daher nur vom Natural **SQL Extended Set** unterstützt.

Weitere Informationen siehe *search-condition*.

Einfache WHEN-Klausel

```
scalar-expression { WHEN scalar-expression THEN { NULL                } }...
                  {                             { scalar-expression } }
```

Eine einfache WHEN-Klausel entspricht nicht dem Standard SQL und wird daher nur vom Natural **SQL Extended Set** unterstützt.

cast-expression

```
CAST (scalar-expression AS data-type)
```

Ein *cast-expression* entspricht nicht dem Standard SQL und wird daher nur vom Natural **SQL Extended Set** unterstützt.

user-defined-function-reference

Die Option *user-defined-function-reference* gehört zum Natural **SQL Extended Set**. Mit dieser Option können Sie beliebige benutzerdefinierte Funktionen aufrufen. Argumente müssen durch Kommas abgetrennt und in Klammern gesetzt werden. Die benutzerdefinierte Funktion muss in der Ziel-RDBMS deklariert werden.

sequence-reference

Die Option *sequence-reference* gehört zum Natural **SQL Extended Set**.

```
{ NEXT VALUE FOR sequence-name     }
{ PREVIOUS VALUE FOR sequence-name }
```

Mit dieser Option können Sie den nachfolgenden oder den vorhergehenden Wert eines Abfolgeobjekts referenzieren. Das Abfolgeobjekt muss in der Ziel-RDBMS erstellt werden, ehe es zur Laufzeit referenziert wird.

scalar-fullselect

```
(fullselect)
```

Die Option *scalar-fullselect* gehört zum Natural **SQL Extended Set**.

Ein *scalar-fullselect*, der in einem Ausdruck unterstützt wird, ist ein in Klammern stehendes *fullselect*, das eine einzelne, aus einem einzigen Spaltenwert bestehende Zeile zurückgibt. Wenn der *fullselect* keine Zeile zurückgibt, ist das Ergebnis des Ausdrucks der Nullwert. Wenn für einen *scalar-fullselect* mehr als eine Zeile zurückgegeben werden soll, tritt ein Fehler auf.

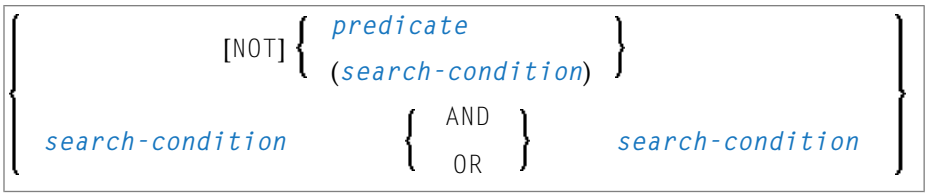
row-value-expression

```
(scalar-expression,...)
```

Ein *row-value-expression* gibt eine einzelne Zeile zurück, die aus einem oder mehreren Spaltenwerten besteht. Die Werte können in Form einer Liste von Ausdrücken angegeben werden. Die Anzahl der vom *row-value-expression* zurückgegebenen Spalten ist gleich der Anzahl der Ausdrücke. *row-value-expression* kann als Operand bei mehreren *predicates* verwendet werden (*quantified*, DISTINCT, *comparison* und IN).

147 Suchbedingungen

- search-condition 1196
- predicate 1196



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 und 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



Ein *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 anderen Arten 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'
    ...
```

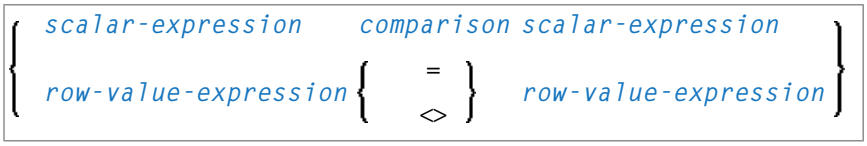
Anmerkung: Das Prozentzeichen (%) kann zu Konflikten mit Natural-Terminalkommandos führen. In diesem Fall müssen Sie als Steuerzeichen für Terminalkommandos ein anderes Zeichen als % definieren (siehe *Ändern des Terminalkommando-Steuerzeichens*).

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 nachfolgend beschrieben.

- comparison-predicate
- BETWEEN-predicate
- DISTINCT-predicate
- LIKE-predicate
- NULL-predicate

- IN-predicate
- quantified-predicate
- EXISTS-predicate
- XMLEXISTS-predicate

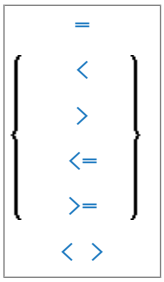
comparison-predicate



Ein *comparison-predicate* vergleicht zwei Werte oder einen Satz Werte mit einem anderen Satz Werte.

Siehe Informationen zu *scalar-expression*.

comparison



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
```

Weitere Informationen siehe ***SELECT-Ausdrücke***.

BETWEEN-predicate

```
scalar-expression [NOT] BETWEEN scalar-expression AND scalar-expression
```

Ein *BETWEEN-predicate* vergleicht einen Wert mit einem Bereich von Werten.

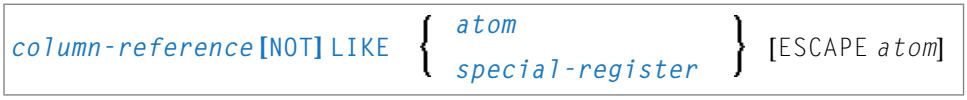
Siehe Informationen zu *scalar-expression*.

DISTINCT-predicate

```
{ scalar-expression IS [NOT] DISTINCT FROM scalar-expression          }
{ row-value-expression IS [NOT] DISTINCT FROM row-value-expression }
```

Ein *DISTINCT-predicate* vergleicht einen Wert mit einem anderen Wert oder einen Satz Werte mit einem anderen Satz Werte.

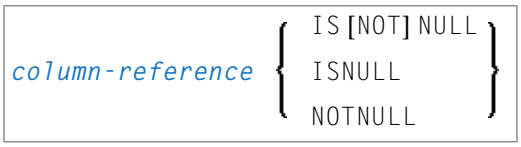
LIKE-predicate



Ein *LIKE-predicate* sucht nach Zeichenketten, die ein bestimmtes Muster haben.

Siehe Informationen zu *column-reference*, *atom* und *special-register*.

NULL-predicate

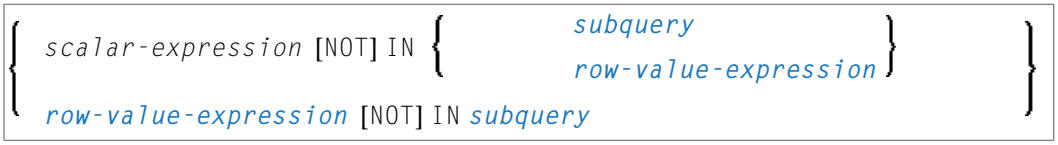


Ein *NULL-predicate* prüft auf Nullwerte.

Wenn die Compiler-Option DB2ARRY auf ON gesetzt ist, kann ein Natural-Array oder ein feststehender Indexbereich eines Arrays als *atom* angegeben werden. Der Natural SQL Compiler zerlegt dann das Array oder den feststehenden Indexbereich in eine Liste mit skalaren *host-variables*.

Siehe Informationen zu *column-reference*.

IN-predicate

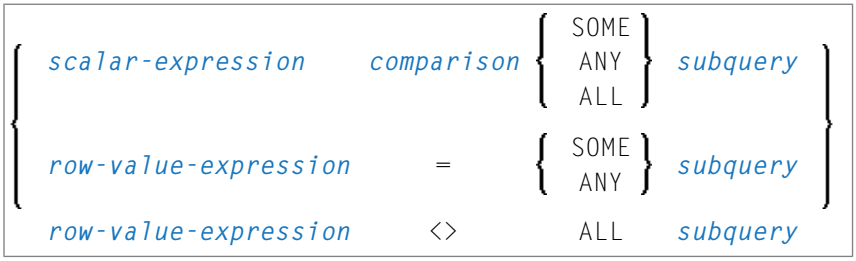


Ein *IN-predicate* vergleicht einen Wert oder einen Satz Werte mit einer Sammlung von Werten.

Siehe Informationen zu *scalar-expression*, *atom* und *special-register*.

Siehe Informationen zu *subquery*.

quantified-predicate



Ein *quantified-predicate* vergleicht einen Wert oder einen Satz Werte mit einer Sammlung von Werten.

Siehe Informationen zu *scalar-expression*, *comparison* und *subquery*.

EXISTS-predicate



Ein *EXISTS-predicate* prüft, ob bestimmte Zeilen vorhanden sind.

Die Bedingung des *EXISTS-predicate* kann nur erfüllt werden, wenn das Ergebnis der ausgewerteten *subquery* nicht leer ist, d.h. wenn mindestens ein Datensatz (eine Zeile) in der FROM-Tabelle der *subquery* die WHERE-Klausel-Suchbedingung dieser *subquery* erfüllt.

Beispiel für EXISTS:

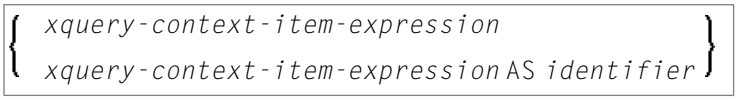
```
DEFINE DATA LOCAL
1 #NAME    (A20)
END-DEFINE
...
SELECT NAME
  INTO #NAME
  FROM SQL-PERSONNEL
  WHERE EXISTS
    ( SELECT *
      FROM SQL-EMPLOYEES
      WHERE PERSNR > 1000
        AND NAME < 'L' )
        ...
END-SELECT
...
```

Siehe Informationen zu *subquery*.

XMLEXISTS-predicate



xquery-argument



Das *XMLEXISTS-predicate* gehört zum Natural **SQL Extended Set**.

Das *XMLEXISTS-predicate* prüft, ob ein XPATH-Ausdruck eine Folge von einem oder mehreren *items* zurückgibt. Weitere Informationen siehe IBM *Db2 XML Guide*.

148 SELECT-Ausdrücke

- selection 1204
- table-expression 1205

```
SELECT selection table-expression
```

Ein *select-expression* gibt eine Ergebnistabelle an. Es wird bei den folgenden Statements benutzt: INSERT | SELECT | UPDATE

selection



selection gibt an, welche Spalten der Resultset-Tabellen (Ergebnismengen) ausgewählt werden sollen.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
ALL | DISTINCT	**Eliminierung doppelt vorkommender Zeilen:** Doppelt vorkommende Zeilen werden nicht automatisch aus dem Ergebnis eines *select-expression* entfernt. Wenn Sie dies wünschen, müssen Sie das Schlüsselwort DISTINCT angeben. Die Alternative zu DISTINCT ist ALL. Wenn Sie nichts angeben, gilt ALL.

scalar-expression	**Skalar-Ausdruck:**
	
	Anstelle einfacher Spaltennamen kann *selection* auch allgemeine *scalar-expressions* enthalten, die Skalar-Operatoren und Skalar-Funktionen enthalten, die berechnete Werte liefern. Siehe ***Skalar-Ausdrücke***.
	
	Beispiel:
	
	`SELECT NAME, 65 - AGE`
	`  FROM SQL-PERSONNEL`
	`  ...`

AS	Das optionale Schlüsselwort AS leitet einen Korrelationsnamen (*correlation-name*) für eine Spalte ein.
correlation-name	**Korrelationsname:** Es besteht die Möglichkeit, einem *scalar-expression* einen *correlation-name* als Alias-Namen für eine Ergebnisspalte zuzuweisen.

Syntax-Element	Beschreibung
	Der *correlation-name* braucht nicht eindeutig sein. Wenn für eine Ergebnisspalte kein *correlation-name* angegeben wird, wird der entsprechende *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.
unpack-row	Siehe *unpack-row* weiter unten.

*	**Stern-Notation (*) :**
	
	Alle Spalten der Ergebnistabelle werden ausgewählt.
	
	Beispiel:
	
	`SELECT *`
	`  FROM SQL-PERSONNEL, SQL-AUTOMOBILES`
	`  ...`

unpack-row

```
UNPACK (scalar-expression) .* AS ({field-name data-type} , ...)
```

Mit *unpack-row* geben Sie eine Spalte nicht entpackter Binärwerte an, die zurückgegeben werden, wenn die SQL UNPACK-Funktion aufgerufen wird. Die Anzahl der Feldnamen (*field-names*) und Datentypen (*data-types*) muss mit der Anzahl der von der SQL UNPACK-Funktion zurückgegeben Felder übereinstimmen.

table-expression

```
from-clause [where-clause]
[group-by-clause] [having-clause]
[order-by-clause] [fetch-first-clause]
```

Ein *table-expression* gibt an, von wo und nach welchen Kriterien Zeilen ausgewählt werden sollen.

Folgende Themen werden behandelt:

- FROM-Klausel
- Table Reference
- WHERE-Klausel
- GROUP BY-Klausel
- HAVING-Klausel

- ORDER BY-Klausel
- INPUT SEQUENCE
- ORDER OF table-designator
- FETCH FIRST-Klausel
- Beispiele für Tabellenausdrücke

FROM-Klausel

```
FROM table-reference,...
```

Diese Klausel gibt an, von welchen Tabellen das Resultset (Ergebnismenge) erstellt werden soll.

Table Reference

```
{ table-name [period-specification] [correlation-clause]
  [TABLE] subquery correlation-clause
  joined-table
  TABLE (function-name (scalar-expression,...)) correlation-clause
  data-change-table-reference [correlation-clause]
  xmltable-function correlation-clause }
```

Die in der FROM-Klausel angegebenen Tabellen müssen die in der Auswahlliste verwendeten Spaltenfelder enthalten.

Sie können entweder eine einzelne Tabelle angeben oder eine Zwischentabelle erzeugen, die das Ergebnis einer *subquery* oder einer Join-Operation ist (siehe unten).

Da in einer FROM-Klausel mehrere Tabellen (d.h. DDMs) angesprochen werden können und eine *table-expression* mehrere FROM-Klauseln enthalten kann, wenn *subqueries* angegeben sind, bestimmt die Datenbanknummer (DBID) des ersten DDMs in der ersten FROM-Klausel die zugrunde liegende Datenbank.

TABLE *function-name*-Klausel

Die TABLE *function-name*-Klausel gehört zum **SQL Extended Set** und erfordert eine *correlation-clause* mit einer *column-name*-Liste.

period-specification

Die *period-specification*-Klausel gehört zum **SQL Extended Set**.



Mit der optionalen Angabe einer *period-specification* geben Sie an, dass für die Zeittabelle (*table-name*) eine Zeitraumangabe gilt. Derselbe Zeitraumname (SYSTEM_TIME oder BUSINESS_TIME) darf nicht mehr als einmal für dieselbe Tabelle angegeben werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FOR SYSTEM_TIME	Gibt an, dass der SYSTEM_TIME-Zeitraum für die Zeitraumangabe benutzt wird. SYSTEM_TIME muss ein Zeitraum sein, der in der Tabelle definiert ist, und die Tabelle muss eine vom System verwaltete Zeittabelle sein, die bei der Systemdatenversionierung definiert wird.
FOR BUSINESS_TIME	Gibt an, dass der BUSINESS_TIME-Zeitraum für die Zeitraumangabe benutzt wird. BUSINESS_TIME muss ein Zeitraum sein, der in der Tabelle definiert ist
expr, *expr1*, *expr2*	Geben Sie Ausdrücke an, die einen Wert eines eingebauten Datentyps zurückgeben, der mit dem Datentyp der Spalten des angegebenen Zeitraums vergleichbar sein und der keine TIME ZONE enthalten darf.
AS OF *expr*	Gibt an, dass die Tabelle jede Zeile enthält, für die der Startwert für den angegebenen Zeitraum kleiner als der oder gleich dem Ausdruck *expr* und der Endewert für den angegebenen Zeitraum größer als der Ausdruck *expr* ist.
FROM *expr1* TO *expr2*	Gibt an, dass die Tabelle Zeilen enthält, die für den mit *expr1* bis *expr2* angegebenen Zeitraum vorhanden sind. Eine Zeile wird in die Tabelle aufgenommen, wenn der Startwert für den Zeitraum in der Zeile kleiner als *expr2* und der Endewert für den Zeitraum in der Zeile größer als *expr1* ist.
BETWEEN *expr1* AND *expr2*	Gibt an, dass die Tabelle eine Zeile enthält, in welcher sich der angegebene Zeitraum zu einem beliebigen Zeitpunkt zwischen *expr1* und *expr2* überlappt. Eine Zeile wird in die Tabelle aufgenommen, wenn der Startwert für den Zeitraum in der Zeile kleiner als oder gleich *expr2* und der Endewert für den Zeitraum in der Zeile größer als *expr1* ist.

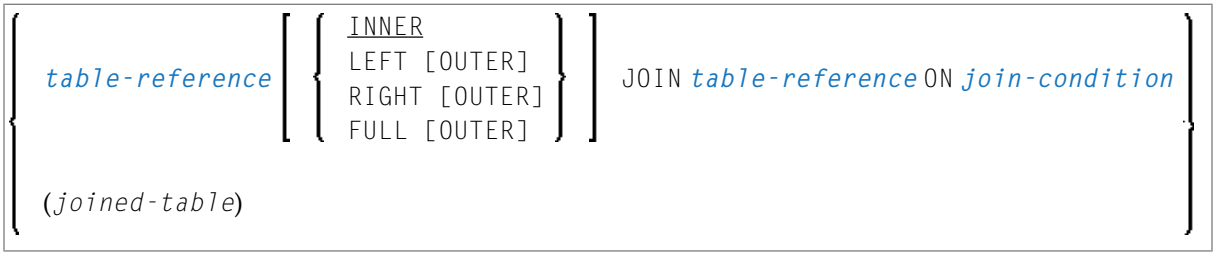
Optional kann eine *correlation-clause* zu einem *table-name* zugeordnet werden. Bei einer *subquery* muss eine *correlation-clause* zugeordnet werden.

correlation-clause

```
[AS] correlation-name [(column-name,...)]
```

Eine *correlation-clause* besteht aus dem optionalen Schlüsselwort AS und einem *correlation-name*. Optional kann danach noch eine einfache Liste mit Spaltennamen angegeben werden. *column-name* gehört zum **SQL Extended Set**.

joined-table



Eine *joined-table* gibt eine Zwischentabelle an, die das Ergebnis einer Verbundoperation (Join) ist.

Bei der Verbundoperation kann es sich um einen INNER, LEFT OUTER, RIGHT OUTER oder FULL OUTER JOIN handeln. Falls Sie nichts angeben, gilt INNER.

Es ist möglich, mehrere Joins zu schachteln, d.h. die Tabellen, die die Zwischentabelle bilden, können ihrerseits Zwischentabellen einer Verbundoperation oder einer *subquery* sein, wobei letztere wiederum ebenfalls eine *joined-table* oder eine weitere *subquery* in der FROM-Klausel haben kann.

join-condition

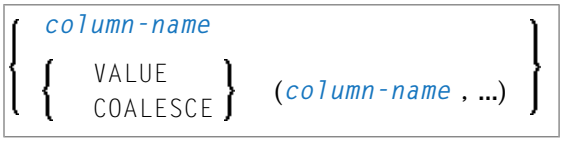
Bei INNER-, LEFT OUTER- und RIGHT OUTER-Verbundoperationen:

```
search-condition
```

Bei FULL OUTER-Verbundoperationen:

```
full-join-expression = full-join-expression [AND ...]
```

full-join-expression

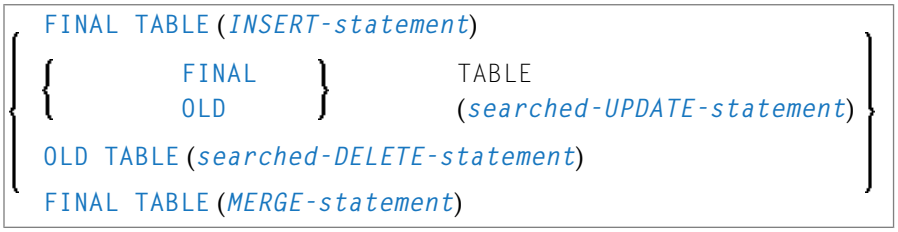


Innerhalb eines *join-expression* sind nur *column-names* und die *scalar-function* VALUE (oder ihr Synonym COALESCE) erlaubt.

Weitere Informationen siehe *column-name*.

data-change-table-reference

Die *data-change-table-reference*-Klausel gehört zum **SQL Extended Set**



Eine *data-change-table-reference* gibt eine Zwischentabelle an, die auf den Zeilen basiert, die durch das in der FROM-Klausel angegebene SQL-Datenänderungs-Statement verändert werden. Eine *data-change-table-reference* kann nur als die einzige Tabellenreferenz in der FROM-Klausel angegeben werden.

Syntax-Element-Beschreibung:

Syntax-Element	Beschreibung
FINAL TABLE	Gibt an, dass die Zeilen der Zwischentabelle den Satz Zeilen, die vom SQL-Datenänderungs-Statement geändert werden, so darstellt wie sie beim Beenden des SQL-Datenänderungs-Statement erscheinen.
OLD TABLE	Gibt an, dass die Zeilen der Zwischentabelle den Satz Zeilen, die vom SQL-Datenänderungs-Statement geändert werden, so darstellt, wie sie vor der Anwendung des SQL-Datenänderungs-Statement vorhanden sind.

xmltable-function

Die *xmltable-function*-Klausel gehört zum **SQL Extended Set**

Eine *xmltable-function* gibt den Aufruf der eingebauten XMLTABLE-Funktion an.

WHERE-Klausel



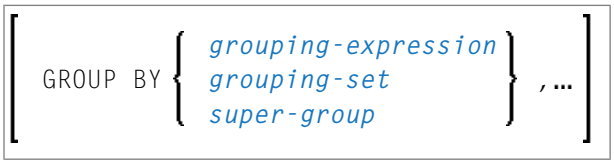
In der WHERE-Klausel geben Sie ein Auswahlkriterium (*search-condition*) an, nach dem die Zeilen ausgewählt werden sollen.

Beispiel:

```
DEFINE DATA LOCAL
01 NAME   (A20)
01 AGE    (I2)
END-DEFINE
...
SELECT *
  INTO NAME, AGE
  FROM SQL-PERSONNEL
  WHERE AGE = 32
END-SELECT
...
```

Weitere Informationen siehe *Suchbedingungen*.

GROUP BY-Klausel



Die GROUP BY-Klausel gibt eine Gruppierung der Ergebnistabelle an. Es ergibt sich eine Menge von Gruppen mit Zeilen. Innerhalb einer Gruppe mit mehr als einer Zeile sind alle Werte, die die Gruppe definieren, gleich.

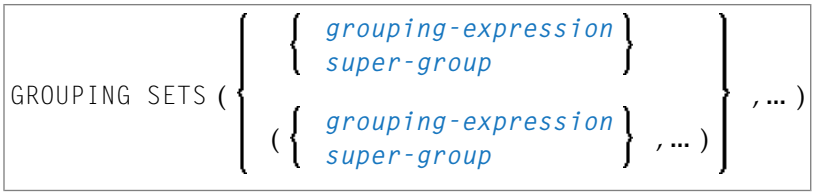
- grouping-expression
- grouping-set

- super-group

grouping-expression

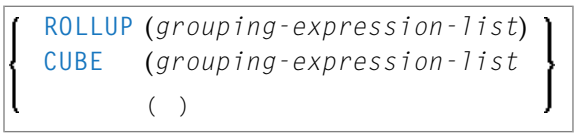
Ein *grouping-expression* ist ein *scalar-expression*, der die Gruppierung eines *result-set* definiert.

grouping-set



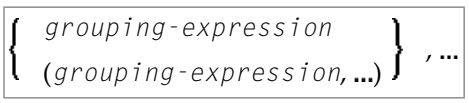
Ein *grouping-set* wird verwendet, um mehrere Gruppierungsklauseln in einem einzigen Statement anzugeben. Ein *grouping-set* kombiniert zwei oder mehr Zeilengruppen zu einem einzigen Resultset (Ergebnismenge). Es ist das gleiche wie eine UNION-Verkettung mehrerer *select-expressions* mit einer GROUP BY-Klausel, wobei jeder Ausdruck einem *grouping-set* entspricht. Ein *grouping-set* ist ein einzelnes Element oder eine Liste mit Elementen, die durch Klammern voneinander abgegrenzt sind. Ein Element ist entweder ein *grouping-expression* oder eine *super-group*. Ein *grouping-set* hat den Vorteil, dass die Gruppen in einem einzigen Durchgang über die Basistabelle berechnet werden.

super-group



Eine *super-group* ist eine komplexerer *grouping-set*.

Eine *grouping-expression-list* definiert die Anzahl der Elemente, die in einer ROLLUP- oder CUBE-Operation verwendet werden. Elemente mit mehreren *grouping-expressions* werden durch Klammern voneinander abgegrenzt.



Gesamtresultat „grand total“ ( ):

ROLLUP und CUBE liefern eine Zeile, die die Gesamtaggregation (Gesamtresultat) ist. Dies kann durch leere Klammern ( ) in der GROUPING SETS-Klausel angegeben werden.

ROLLUP

Eine ROLLUP-Gruppierung ist wie eine Reihe von *grouping-sets*. Zusätzlich zu den regulären gruppierten Zeilen erzeugt eine ROLLUP-Gruppierung einen Resultset (Ergebnismenge), die Zwischensummenzeilen enthält. Zwischensummenzeilen sind „super-aggregierte" Zeilen, die zusätzliche Gesamtsummen enthalten. Die aggregierten Werte werden mit denselben Spaltenfunktionen abgerufen, die zum Abrufen der regulären gruppierten Zeilen verwendet werden.

Allgemein geben Sie eine ROLLUP-Gruppierung mit *n* Elementen wie folgt an:

```
GROUP BY ROLLUP (c1, c2, ..., cn-1, cn)
```

Was folgender Angabe entspricht:

```
GROUP BY GROUPING SETS ((c1, c2, ..., cn-1, cn),
                        (c1, c2, ..., cn-1),
                        ...
                        (c1, c2),
                        (c1),
                        ( ))
```

CUBE

Eine CUBE-Gruppierung ist wie eine Reihe von *grouping-sets*. Zusätzlich zu den ROLLUP-Aggregationszeilen erzeugt CUBE einen Resultset (Ergebnismenge), die Kreuztabellierungszeilen enthält. Kreuztabellierungszeilen sind zusätzliche „super-aggregierte" Zeilen. Die *grouping-expression-list* eines CUBE berechnet zusammen mit dem Gesamtresultat **„grand total"** alle Permutationen. Als Ergebnis werden die *n* Elemente eines CUBE in 2**n *grouping-sets* umgesetzt. Zum Beispiel:

```
GROUP BY CUBE (a, b, c)
```

Was folgender Angabe entspricht:

```
GROUP BY GROUPING SETS ((a, b, c),
                        (a, b),
                        (a, c),
                        (b, c),
                        (a),
                        (b),
```

```
                          (c),

                          ())
```

HAVING-Klausel

```
[HAVING search-condition]
```

Wenn Sie eine `HAVING`-Klausel verwenden, sollten Sie auch eine `GROUP BY`-Klausel verwenden.

So wie die `WHERE`-Klausel Zeilen aus einer Ergebnistabelle aussortiert, sortiert die `HAVING`-Klausel Gruppen aus, und zwar auf Grundlage einer Suchbedingung (*`search-condition`*). Skalar-Ausdrücke in einer `HAVING`-Klausel dürfen pro Gruppe nur einen Wert enthalten.

Weitere Informationen siehe ***Scalar Expressions*** und ***Search Conditions***.

Beispiel:

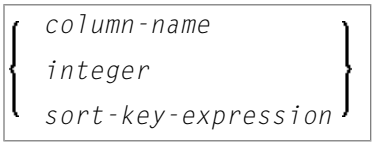
```
DEFINE DATA LOCAL
1 #NAME     (A20)
1 #AVGAGE   (I2)
1 #NUMBER   (I2)
END-DEFINE
...
SELECT NAME, AVG(AGE), COUNT(*)
  INTO #NAME, #AVGAGE, #NUMBER
  FROM SQL-PERSONNEL
  GROUP BY NAME
  HAVING COUNT(*) > 1
  ...
```

ORDER BY-Klausel

```
                  ┌                    ┐
                  │ sort-key [ ASC  ]  │
ORDER BY          │          [ DESC ]  │
                  │                    │
                  │ INPUT SEQUENCE     │
                  │                    │
                  │ ORDER OF table-designator
                  └                    ┘
```

Die *`order-by`*-Klausel gibt die Reihenfolge für die Ergebnistabellenzeilen an.

sort-key



Ein *sort-key-expression* ist ein *expression*, der nicht einfach ein *column-name* oder eine vorzeichenlose ganzzahlige *constant* ist.

INPUT SEQUENCE

`INPUT SEQUENCE` gehört zum **SQL Extended Set**.

`INPUT SEQUENCE` gibt an, dass die Ergebnistabelle die Eingabereihenfolge der Zeilen widerspiegelt, die in der `VALUE`-Klausel eines `INSERT`-Statements angegeben sind. `INPUT SEQUENCE` kann nur angegeben werden, wenn in der *from-clause* ein `INSERT`-Statement angegeben ist.

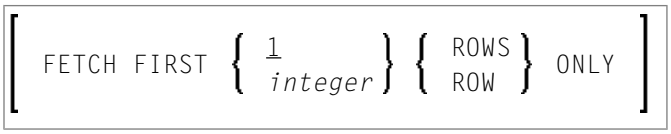
ORDER OF table-designator

`ORDER OF` *table-designator* gehört zum **SQL Extended Set**.

`ORDER OF` *table-designator* gibt an, dass die Sortierfolge der bezeichneten Tabelle nach der Tabellenkennung (*table-designator*) auf die Ergebnistabelle der Abfrage angewendet werden soll. Der *table-designator* identifiziert eine Basistabelle, eine Datensicht (view) oder einen geschachtelten Tabellenausdruck (*table-expression*) einer Untervorauswahl (*subselect*) eindeutig.

FETCH FIRST-Klausel

Die *fetch-first*-Klausel gehört zum **SQL Extended Set**.



Die *fetch-first*-Klausel begrenzt die Anzahl der Zeilen, die abgerufen werden können. Sie verbessert die Performance von Abfragen, wenn nur eine begrenzte Anzahl an Zeilen benötigt wird.

Beispiele für Tabellenausdrücke

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
...
```

149 Flexible SQL

- Flexible SQL benutzen 1218
- Textvariablen in flexibler SQL angeben 1219

Eine weitere Möglichkeit, SQL-Statements einzusetzen, ist die Verwendung so genannter „flexibler SQL“. Diese gestattet Ihnen die Verwendung beliebiger SQL-Syntax.

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 aus einer Zeichenkette bestehen, die über mehrere Zeilen gehen und ganze oder teilweise Kommentarzeilen enthalten kann (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 eines *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 erkannt, 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 flexibler SQL angeben

Innerhalb der flexiblen SQL können Sie auch so genannte „Textvariablen“ angeben.

```
<<:T:host-variable [LINDICATOR:host-variable]>>
```

Die Syntax-Elemente sind im Folgenden beschrieben.

:T:	Eine Textvariable ist eine *host-variable* mit dem Präfix :T:. Sie muss alphanumerisches Format haben. 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. Nach dem Ersetzen werden nachfolgende Leerzeichen aus der eingefügten Textzeichenkette entfernt. 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 *host-variables* enthalten. Ein Statement, das eine Textvariable enthält, wird immer im dynamischen SQL-Modus ausgeführt.

LINDICATOR	**LINDICATOR-Option:** Nach der Textvariablen können Sie das Schlüsselwort LINDICATOR sowie eine Längenindikator-Variable (d.h. eine *host-variable* mit vorangestelltem Doppelpunkt) angeben. Die Längenindikator-Variable muss Format/Länge I2 haben. Wenn Sie keine LINDICATOR-Variable angeben, wird der gesamte Inhalt der Textvariablen in die SQL-Zeichenkette eingefügt. Wenn Sie eine LINDICATOR-Variable angeben, werden nur die ersten *n* Zeichen (wobei *n* 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. Siehe auch allgemeine Informationen zu *host-variable*.

Beispiel mit einer Textvariablen:

```
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'
```

XV Referenzierte Beispielprogramme

150 Referenzierte Beispielprogramme

- ASSIGN 1226
- AT BREAK 1227
- AT END OF DATA 1229
- AT END OF PAGE 1230
- AT START OF DATA 1231
- AT TOP OF PAGE 1232
- DEFINE SUBROUTINE 1233
- FIND 1234
- FOR 1236
- HISTOGRAM 1237
- IF 1238
- PERFORM BREAK PROCESSING 1239
- READ 1240
- REPEAT 1241
- SORT 1243
- STORE 1244
- UPDATE 1246
- Beispielprogramme für Systemvariablen 1247

Dieser Abschnitt 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.

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         NAME         LEAVE-DUE-L
                        CODE
-------------------- ---------- -------------------- -----------

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
```

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         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     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             SALARY   CURRENCY
                             POSITION                        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=O) '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.
                          MILWAUKEE                 MONTERREY
554349                    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     CNT
                       PERSONS
-------------------- ----------- -----------

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      ANNUAL     MAKE
                         OF       SALARY
                       BIRTH
-------------------- ---------- ---------- --------

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         NAME            ISN         CNT
   ID
--------- -------------------- ----------- -----------

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 RPTEX1R:

```
ENTER A PERSONNEL NUMBER:
```

RPTEX2R - REPEAT mit WHILE- und UNTIL-Option (Reporting Mode)

```
** Beispiel 'RPTEX2R': 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 RPTEX2R:

```
#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   ANNUAL     ANNUAL   #TOTAL-SALARY CURRENCY PERCENT
   ID       SALARY     SALARY                  CODE     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=O) /
        '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
```

Stichwortverzeichnis

A

ACCEPT
    statement, 27
ADD
    statement, 33
application-independent variable
    define, 253
arithmatic operations
    overview of statements, 16
ASSIGN
    statement, 39
AT BREAK
    statement, 41
AT END OF DATA
    statement, 51
AT END OF PAGE
    statement, 57
AT START OF DATA
    statement, 65
AT TOP OF PAGE
    statement, 71

B

BACKOUT TRANSACTION
    statement, 77
BEFORE BREAK PROCESSING
    statement, 81

C

CALL
    statement, 87
CALL FILE
    statement, 115
CALL LOOP
    statement, 121
CALLDBPROC
    SQL statement, 125
CALLNAT
    statement, 131
CLOSE CONVERSATION
    statement, 139
CLOSE PC FILE
    statement, 145
CLOSE PRINTER
    statement, 149
CLOSE WORK FILE
    statement, 153
COMMIT
    SQL statement, 157
common set
    SQL statements, 1163
component based programming
    overview of statements, 20
COMPOSE
    statement, 159
COMPRESS
    statement, 189
COMPUTE
    statement, 201
constant
    SQL statements, 1166
context variable for Natural RPC
    define, 257
CREATE OBJECT
    statement, 211

D

data movement operations
    overview of statements, 16
data type
    SQL statements, 1176
database access and update
    overview of statements, 14
DECIDE FOR
    statement, 215
DECIDE ON
    statement, 221
DEFINE CLASS
    statement, 227
DEFINE DATA
    statement
        array dimension definition, 279
        array-init-definition, 287
        example, 295
        initial value definition, 283
        parameters for field/variable, 293
        redefinition, 275
        rules, 233
        variable definition, 265
        view definition, 269
DEFINE DATA CONTEXT
    statement, 257
DEFINE DATA GLOBAL
    statement, 237

DEFINE DATA INDEPENDENT
statement, 253
DEFINE DATA LOCAL
statement, 247
DEFINE DATA OBJECT
statement, 261
DEFINE DATA PARAMETER
statement, 241
DEFINE FUNCTION
statement, 307
DEFINE PRINTER
statement, 315
DEFINE PROTOTYPE
statement, 329
DEFINE SUBROUTINE
statement, 337
DEFINE WINDOW
statement, 347
DEFINE WORK FILE
statement, 357
DELETE
SQL statement, 371
statement, 367
DISPLAY
statement, 377
DIVIDE
statement, 401
DML statements
overview, 14
DO
statement, 409
DOEND
statement, 409
DOWNLOAD PC FILE
statement, 413
dynamic variable
overview of statements for memory management, 20

E

EJECT
statement, 421
END
statement, 427
END TRANSACTION
statement, 431
ESCAPE
statement, 437
EXAMINE
statement, 443
for Unicode graphemes, 455
EXAMINE TRANSLATE
statement, 453
example program
referenced in documentation, 1225
EXPAND
statement, 467
extended set
SQL statements, 1163

F

FETCH
statement, 475
FIND
statement, 481
flexible SQL, 1217
FOR
statement, 523
FORMAT
statement, 529

G

GET
statement, 535
GET SAME
statement, 541
GET TRANSACTION DATA
statement, 545
global data
define, 237

H

HISTOGRAM
statement, 549

I

IF
statement, 563
IF SELECTION
statement, 567
IGNORE
statement, 571
INCLUDE
statement, 573
INPUT
statement
syntax 1 - dynamic screen layout specification, 589
syntax 2 - using predefined map layout, 603
INSERT
SQL statement, 615
INTERFACE
statement, 625
internet
overview of statements, 21
internet and parsing
overview of statements, 21

L

LIMIT
statement, 633
local data
define, 247
logical condition processing
overview of statements, 18
LOOP
statement, 637
loop execution
overview of statements, 16

M

MERGE SQL

SQL statement, 641
METHOD
statement, 651
MOVE
statement, 657
MOVE ALL
statement, 672
MOVE INDEXED
statement, 687
MULTIPLY
statement, 689

N

name
SQL statements, 1167
NaturalX
define data object, 261
NEWPAGE
statement, 695

O

OBTAIN
statement, 701
ON ERROR
statement, 711
OPEN CONVERSATION
statement, 717
OPTIONS
statement, 721
output report
overview of statements, 17

P

parameter
SQL statements, 1170
parameter data
define, 241
PARSE JSON
statement, 725
PARSE XML
statement, 737
PASSW
statement, 751
PC file control
overview of statements, 19
PERFORM
statement, 755
PERFORM BREAK PROCESSING
statement, 763
PRINT
statement, 767
PROCESS
statement, 777
PROCESS COMMAND
statement, 781
PROCESS PAGE
statement, 799
PROCESS SQL
SQL statement, 813
program execution
overview of statements, 18
program termination
overview of statements, 19
PROPERTY
statement, 817

R

READ
statement, 823
READ RESULT SET
SQL statement, 857
READ WORK FILE
statement, 863
READLOB
statement, 849
REDEFINE
statement, 877
REDUCE
statement, 883
REINPUT
statement, 889
REJECT
statement, 27, 903
RELEASE
statement, 905
REPEAT
statement, 909
reporting mode
overview of statements, 21
REQUEST DOCUMENT
statement, 915
RESET
statement, 933
RESIZE
statement, 939
RETRY
statement, 945
ROLLBACK
SQL statement, 949
routine execution
overview of statements, 18
RUN
statement, 953

S

scalar expression
SQL statements, 1181
screen generation for interactive processing
overview of statements, 17
search condition
SQL statements, 1195
SELECT
SQL statement, 961
select expression
SQL statements, 1203
SEND METHOD
statement, 983
SEPARATE
statement, 995
session termination
overview of statements, 19
SET CONTROL
statement, 1009

SET GLOBALS
statement, 1013
SET KEY
statement, 1017
SET TIME
statement, 1029
SET WINDOW
statement, 1033
SKIP
statement, 1037
SORT
statement, 1041
SQL
basic syntactical items, 1165
SQL statements
overview, 15
STACK
statement, 1053
statements, xxi
STOP
statement, 1059
STORE
statement, 1065
SUBTRACT
statement, 1073
SUSPEND IDENTICAL SUPPRESS
statement, 1079
syntax
statements, 7

T

TERMINATE
statement, 1085

U

Unicode graphemes
examine, 455
UPDATE
SQL statement, 1095
statement, 1089
UPDATELOB
statement, 1103
UPLOAD PC FILE
statement, 1111
user-defined functions
overview of statements, 19

V

variable
context variable for Natural RPC, 257
define application-independent variable, 253
view concept
SQL statements, 1179

W

work file control
overview of statements, 19
WRITE
statement, 1117
WRITE TITLE
statement, 1135
WRITE TRAILER
statement, 1143
WRITE WORK FILE
statement, 1153

X

X-array
overview of statements for memory management, 20