software AG

Adabas

Utilities

Version 8.1.4

June 2014

Adabas

This document applies to Adabas Version 8.1.4.

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

Copyright © 1971-2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. 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 AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

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

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: ADAMF-UTILITIES-814-20140626

Table of Contents

Utilities xxi
1 Conventions 1
Control Statement Syntax 2
Syntax Conventions 3
Control Statement Rules 5
Parameter Values 5
I ADAACK Utility: Check Address Converter 9
2 Functional Overview 11
3 ACCHECK: Check Address Converter Against Data Storage 13
Optional Parameters 14
Examples 15
4 JCL/JCS Requirements and Examples 17
BS2000 18
z/OS 19
z/VM 20
VSE 21
II ADACDC Utility: Changed-Data Capture 23
5 Functional Overview 25
Phases of Operation and Resulting Files 26
Primary Input Data 29
Primary Output File 29
Transaction File 30
6 Running the Utility 33
Optional Parameters 34
Using ADACDC With ISNREUSE 36
7 Operating System Considerations 39
z/OS 40
VSE 40
BS2000 41
8 The ADACDC User Exit 43
Installing the Exit 44
User Exit Interface 44
User Exit Calls 46
9 Examples 49
10 JCL/JCS Requirements and Examples 51
BS2000 52
z/OS 53
z/VM 54
VSE 55
III ADACMP Utility: Compress-Decompress Data 57
11 Functional Overview 59
COMPRESS Function Overview 60
DECOMPRESS Function Overview 61

12 Input Data Requirements 63
Input Data Structure 64
Multiple-Value Field Count 64
Periodic Group Count 66
Variable-Length Field Size 69
13 Processing 71
Segmented Record Considerations 72
Data Verification 78
Data Compression 78
Representation of LOB Values and Value References in Uncompressed Data 80
Identifying MU and PE Occurrences Greater Than 191 in Compressed Records 81
Restart Considerations 81
User Exit 6 81
14 COMPRESS: Compress an Adabas File 83
Essential Parameters and Subparameters 84
Optional Parameters and Subparameters 85
ADACMP COMPRESS Examples 92
15 DECOMPRESS: Decompress an Adabas File 95
Optional Parameters and Subparameters 96
Decompressing Multiclient Files 101
ADACMP DECOMPRESS Examples 101
16 Field Definition Statements 103
FNDEF: Field and Group Definition 105
FNDEF: Periodic Group Definition 121
COLDE: Collation Descriptor Definition 124
HYPDE: Hyperdescriptor Definition 127
PHONDE: Phonetic Descriptor 130
SUBDE: Subdescriptor Definition 132
SUBFN: Subfield Definition 134
SUPDE: Superdescriptor Definition 135
SUPFN: Superfield Definition 143
17 JCL/JCS Requirements and Examples 145
User Exits with ADACMP 146
BS2000 147
z/OS 150
z/VM 155
VSE 157
18 COMPRESS Function Output 161
Compressed Data Records 162
Rejected Data Records 162
ADACMP Report 168
19 DECOMPRESS Function Output 171
Rejected Data Records 172

IV ADACNV Utility: Database Conversion 177
20 Functional Overview 179
Database Status 215
Procedure 181
21 CONVERT: Convert Database to Higher Version 183
Optional Parameters 184
Conversion Considerations 185
Example 186
22 REVERT: Revert Database to Lower Version 187
Essential Parameter and Subparameter 188
Optional Parameter 188
Reversion Considerations 230
Example 190
23 JCL/JCS Requirements and Examples 191
BS2000 192
z/OS 195
z/VM 196
VSE 197
V ADADBS Utility: Database Services 199
24 Functional Overview 201
Syntax Checking with the TEST Parameter 202
25 ADD: Add Data Set 203
Associator or Data Storage Data Set 204
Essential Parameter and Subparameter 205
Optional Parameters 278
Examples 205
26 ALLOCATE: Allocate File Extent 207
Essential Parameters 208
Optional Parameters 209
Example 209
27 CHANGE: Change Standard Length or Format of a Field 211
Essential Parameters 212
Optional Parameters 213
Example 213
28 CVOLSER: Print Adabas Extents on Given Volume 215
Essential Parameter 216
Optional Parameters 286
Example 216
29 DEALLOCATE: Deallocate File Extent 217
Essential Parameters 218
Optional Parameters 287
Example 219
30 DECREASE: Decrease Last Associator or Data Storage Data Set Size 221
Essential Parameter 222
Optional Parameters 222

Example 289
Procedure 289
31 DELCP: Delete Checkpoint Records 225
Essential Parameter 226
Optional Parameters 226
Example 289
32 DELETE: Delete File 229
Essential Parameter 290
Optional Parameters 230
Examples 231
33 DSREUSE: Reuse Data Storage Blocks 233
Essential Parameters 234
Optional Parameters 234
Example 291
34 ENCODEF: Change File Encoding 237
Essential Parameter 238
Optional Parameters 239
Example 239
35 INCREASE: Increase Last Associator or Data Storage Data Set Size 241
Essential Parameter 292
Optional Parameters 242
Example 243
General Procedure 243
Operating-System-Specific Procedures 243
36 ISNREUSE: Reuse ISNs 249
Essential Parameters 250
Optional Parameters 297
Example 251
37 MODFCB: Modify File Parameters 253
Essential Parameter 254
Optional Parameters 254
Example 300
38 MUPEX: Set Maximum Count for MU and PE Fields 257
Syntax 258
Essential Parameters 258
Optional Parameters 259
Example 259
39 NEWFIELD: Add New Field 261
Essential Parameter 262
Optional Parameters 263
Example 264
40 ONLINVERT: Start Online Invert Process 265
Essential Parameters 266
Optional Parameters 267
Example 268

41 ONLREORFASSO: Start Online Reorder Associator for Files 269
Essential Parameters 270
Optional Parameters 271
Example 271
42 ONLREORFDATA: Start Online Reorder Data for Files 273
Essential Parameters 274
Optional Parameters 274
Example 330
43 ONLREORFILE: Start Online Reorder Associator and Data for Files 277
Essential Parameters 330
Optional Parameters 279
Example 280
44 OPERCOM: Issue Adabas Operator Commands 281
Using OPERCOM Commands in Cluster Environments 282
Optional Parameters 282
Operator Commands 283
45 PRIORITY: Change User Priority 299
Essential Parameter 300
Optional Parameters 300
Example 301
46 RECORDSPANNING: Enable or Disable Record Spanning 303
Syntax 304
Essential Parameters 304
Optional Parameters 304
Example 305
47 RECOVER: Recover Space 307
Optional Parameters 308
48 REFRESH: Set File to Empty Status 309
Essential Parameter 310
Optional Parameters 310
Example 311
49 REFRESHSTATS: Reset Statistical Values 313
Optional Parameters 387
Example 315
50 RELEASE: Release Descriptor 317
Essential Parameters 318
Optional Parameters 318
Example 319
51 RENAME: Rename File or Database 321
Essential Parameter 393
Optional Parameters 322
Examples 323
52 RENUMBER: Change File Number 325
Essential Parameter 326
Optional Parameter 326

Example 327
53 REPLICATION: Activating or Deactivating Replication 329
Essential Parameter 330
Optional Parameter 330
Examples 332
54 ADADBS REPTOR: Activate, Deactivate, Open, or Close Event Replicator Resources 333
Essential Parameters 334
Optional Parameters 334
Examples 336
55 RESETDIB: Reset Entries in Active Utility List 339
Essential Parameters 399
Optional Parameters 340
Examples 341
56 RESETPPT: Reset PPT Blocks 343
Syntax 344
Essential Parameters 344
Optional Parameters 345
57 SPANCOUNT: Count Spanned Records 347
Syntax 348
Essential Parameters 348
Optional Parameters 348
Example 349
58 TRANSACTIONS: Suspend and Resume Update Transaction Processing 351
Essential Parameters 353
Optional Parameters 353
Example 354
59 UNCOUPLE: Uncouple Files 355
Essential Parameter 356
Optional Parameters 356
Example 357
60 JCL/JCS Requirements and Examples 359
Collation with User Exit 360
BS2000 360
z/OS 361
z/VM 362
VSE 363
VI ADADCK Utility: Check Data Storage and DSST 365
61 Functional Overview 367
62 DSCHECK: Check Data Storage 369
Optional Parameters and Subparameters 370
Examples 371
63 JCL/JCS Requirements and Examples 373
BS2000 374
z/OS 375

z/VM ... 376
VSE ... 377
VII ADADEF Utility: Define a Database ... 379
64 Functional Overview ... 381
Database Components ... 382
Checkpoint File ... 382
65 DEFINE: Defining a Database and Checkpoint File ... 383
Essential Parameters ... 386
Optional Parameters ... 387
Examples ... 391
66 MODIFY: Change Field Encodings ... 393
Optional Parameters ... 394
Examples ... 396
67 NEWWORK: Defining a Work File ... 397
Essential Parameter ... 398
Optional Parameters ... 398
Example ... 399
68 JCL/JCS Requirements and Examples ... 401
BS2000 ... 402
z/OS ... 403
z/VM ... 405
VSE ... 407
VIII ADAFRM Utility: Format Adabas Database Components ... 409
69 Functional Overview ... 411
Statement Restrictions ... 412
Formatting Operation ... 412
70 Formatting Database Components ... 413
Formatting Modes ... 414
Syntax ... 414
Essential Parameter ... 416
Optional Parameters ... 416
Examples ... 417
71 JCL/JCS Requirements and Examples ... 419
BS2000 ... 420
z/OS ... 422
z/VM ... 423
VSE ... 425
IX ADAICK Utility: Check Index and Address Converter ... 427
72 Functional Overview ... 429
73 ACCHECK: Check Address Converter ... 431
Essential Parameter ... 432
Optional Parameters ... 432
Sample Output ... 432
74 ASSOPRINT: Print/Dump Associator Blocks ... 435
Essential Parameter ... 523

Optional Parameter 436
75 BATCH: Set Printout Width to 132 Characters Per Line 437
Optional Parameter 438
76 DATAPRINT: Print/Dump Data Storage Blocks 439
Essential Parameter 440
Optional Parameter 440
77 DSCHECK: Print/Dump Content of Data Storage Record 441
Essential Parameter 524
Optional Parameters 442
Sample Output 443
78 DUMP: Activate Dump Print Format 445
Optional Parameter 446
79 FCBPRINT: Print/Dump File Control Block 447
Essential Parameter 448
Optional Parameters 448
Output Considerations 448
80 FDTPRINT: Print/ Dump Field Definition Table 451
Essential Parameter 452
Optional Parameters 452
81 GCBPRINT: Print/Dump General Control Blocks (GCBs) 453
Optional Parameter 454
82 ICHECK: Check Index Against Address Converter 455
Essential Parameter 456
Optional Parameters 456
83 INT: Activate Interpreted Print Format 457
Optional Parameter 458
84 NIPRINT: Print/Dump Normal Index 459
Essential Parameter 460
Optional Parameter 460
85 NOBATCH: Set Print Width to 80 Characters Per Line 461
Optional Parameter 462
86 NODUMP: Suppress Dump Print Format 463
Optional Parameter 464
87 NOINT: Suppress Interpreted Format 465
Optional Parameter 466
88 PPTPRINT: Print/Dump Parallel Participant Table 467
Optional Parameters 468
Example Output 468
89 UIPRINT: Print/Dump Upper Index 471
Essential Parameter 472
Optional Parameters 536
90 Examples 473
91 JCL/JCS Requirements and Examples 475
Collation with User Exit 476
BS2000 476

z/OS 478
z/VM 479
VSE 480
X ADAINV Utility: Inverted List Management 481
92 Functional Overview 483
93 COUPLE: Define File-Coupling Descriptors 485
Essential Parameters 486
Optional Parameters 487
Example 488
Temporary Space for File Coupling 488
Associator Coupling Lists 489
Space for Coupling Lists 490
Space Allocation 491
94 INVERT: Create Descriptors 493
Essential Parameters 494
Optional Parameters and Subparameters 495
Space Allocation for the INVERT Function 497
Examples 497
95 JCL/JCS Requirements and Examples 499
Collation with User Exit 500
BS2000 500
z/OS 503
z/VM 504
VSE 506
XI ADALOD Utility: File Loader 509
96 Functional Overview 511
97 LOAD: Load a File 513
Essential Parameters 515
Optional Parameters and Subparameters 517
Examples 531
LOAD Data and Space Requirements 533
Loading Expanded Files 536
Loading Multiclient Files 538
98 UPDATE: Add/Delete Records 541
Essential Parameters 543
Optional Parameters and Subparameters 544
Examples 549
Formats for Specifying ISNs 550
UPDATE Data and Space Requirements 552
Mass Updates of Expanded Files 554
99 Loader Storage Requirements and Use 555
100 TEMP Data Set Space Usage 557
Sequential TEMP Data Set 680
101 ADALOD Space/Statistics Report 561
102 JCL/JCS Requirements and Examples 563

Collation with User Exit 564
BS2000 564
z/OS 568
z/VM 570
VSE 572
XII ADAMER Utility: ADAM Estimation 575
103 Functional Overview 577
104 Estimate ADAM Access Requirements 579
Essential Parameters 580
Optional Parameters 580
Examples 582
105 ADAMER Output Report Description 583
106 JCL/JCS Requirements and Examples 585
BS2000 586
z/OS 587
z/VM 588
VSE 589
XIII ADAORD Utility: Reordering Functions 591
107 Functional Overview 593
Reorder Functions 594
Restructure Functions 595
Store Function 595
Space Allocation 596
Adabas 8 Considerations 596
108 REORASSO: Reorder Associator 597
Optional Parameters and Subparameters 599
Examples 603
109 REORDATA: Reorder Data Storage 605
Optional Parameters and Their Subparameters 606
Examples 609
110 REORDB: Reorder Database 611
Optional Parameters and Subparameters 614
Examples 620
111 REORFASSO: Reorder Associator for a Single File 623
Essential Parameter 625
Optional Parameters 625
Examples 780
112 REORFDATA: Reorder Data Storage for a Single File 631
Essential Parameter 632
Optional Parameters 633
Examples 636
113 REORFILE: Reorder File 637
Essential Parameter 639
Optional Parameters 640
Examples 645

114 RESTRUCTUREDB: Restructure Database 647
Optional Parameters and Subparameters 649
Examples 799
115 RESTRUCTUREF: Restructure Single Files 653
Essential Parameter 655
Optional Parameters 655
Examples 658
116 STORE: Store Files 659
Optional Parameters and Subparameters 662
Examples 668
117 JCL/JCS Requirements and Examples 671
BS2000 672
z/OS 677
z/VM 680
VSE 683
XIV ADAPLP Utility: Print Data Protection Records from PLOG/Work 687
118 Functional Overview 689
119 ADAPLP Syntax and Examples 691
Optional Parameters and Subparameters 693
Examples 696
120 JCL/JCS Requirements and Examples 697
BS2000 698
z/OS 701
z/VM 704
VSE 706
XV ADAPRI Utility: Print Selected Adabas Blocks 711
121 Functional Overview 713
122 ADAPRI Syntax and Examples 715
Essential Parameters 716
Optional Parameters 716
Examples 872
123 JCL/JCS Requirements and Examples 719
BS2000 720
z/OS 721
z/VM 722
VSE 723
XVI ADARAI Utility: Adabas Recovery Aid 725
124 Function Overview 727
Concepts and Components 728
125 CHKDB: Check the Database Recovery Status 731
126 DISABLE: Disable Recovery Logging 733
127 LIST: Display Current RLOG Generations 735
Additional LIST Information on BS2000 736
Syntax 737
Optional Parameters 737

Examples 739
128 PREPARE: Initialize and Start the RLOG 749
Syntax 750
Essential Parameter 751
Optional Parameters 751
Examples 751
129 RECOVER: Build a Recovery Job Stream 753
Recovery Processing 754
Optimized Recovery Processing 756
Requirements 756
Restrictions 757
Input Needed for Recovery 926
Output from the Recovery Operation 758
Executing the RECOVER Function 759
File-Level Recovery 760
Syntax 761
Optional Parameters and Subparameters 761
Examples 764
Skeleton Job Control 764
User Exit to Change JCL 949
Prerecovery Checking 767
Restarting the RECOVER Function or Recovery Job Stream 768
130 REMOVE: Remove the Recovery Aid 769
Example 770
131 JCL/JCS Requirements and Examples 771
BS2000 772
z/OS 788
z/VM 793
VSE 795
XVII ADAREP Utility: Database Status Report 797
132 Functional Overview 799
133 Report Syntax 801
Optional Parameters 802
Examples 805
134 Processing Save Tape Input 807
Supplying Protection Log Input 808
Checking Input Tapes 809
Concurrent Parameters 809
Reports for Delta Save Tapes 809
Report Layout 1020
135 Report Description 811
General Database Information 812
File Information 823
Checkpoint Information 835
136 JCL/JCS Requirements and Examples 841

BS2000 ... 842
z/OS ... 843
z/VM ... 844
VSE ... 845
XVIII ADARES Utility: Database Recovery ... 849
137 Functional Overview ... 851
Information about using ADARES ... 852
138 BACKOUT Functions ... 857
139 BACKOUT: Back Out Updates Using the Sequential Protection Log (SIBA) ... 859
Essential Parameters ... 861
Optional Parameters and Subparameters ... 861
Examples ... 865
140 BACKOUT DPLOG or MPLOG: Back Out Updates Using the Dual or Multiple Protection Log ... 867
Executing the Function ... 868
Syntax ... 869
Essential Parameter ... 870
Optional Parameters ... 870
Example ... 874
141 CLCOPY: Copy Dual Command Log ... 875
Optional Parameters ... 876
Examples ... 877
142 COPY: Copy a Sequential Protection Log or Save Tape ... 879
Optional Parameters ... 881
Examples ... 882
143 MERGE CLOG: Merge Nucleus Cluster Command Logs ... 883
Essential Parameter ... 884
144 PLCOPY: Copy Protection Log to Sequential Data Set ... 885
Optional Parameters ... 886
Examples ... 888
145 REGENERATE: Regenerate Updates ... 889
Syntax ... 890
Essential Parameters ... 891
Optional Parameters and Subparameters ... 891
Examples ... 896
146 REPAIR: Repair Data Storage Blocks ... 899
Syntax ... 900
Essential Parameter ... 900
Optional Parameters ... 901
Examples ... 901
147 Multithreaded Processing Statistics ... 903
148 JCL/JCS Requirements and Examples ... 905
BS2000 ... 906
z/OS ... 913

z/VM 920
VSE 925
XIX ADASAV Utility: Save/Restore Database or Files 931
149 Functional Overview 933
RESTONL and RESTORE Functions 934
Adabas Release Support 935
150 RESTONL: Restore Database from Online Source 937
Conditions 938
Result 939
Syntax 939
Optional Parameters and Subparameters 939
Examples 941
151 RESTONL FILES: Restore Files to Original RABNs from Online Source 943
Conditions 944
Result 945
Syntax 945
Optional Parameters and Subparameters 946
Examples 949
152 RESTONL FMOVE: Restore Files to Any RABNs from Online Source 951
Conditions 952
Result 953
Syntax 953
Optional Parameters 954
Examples 961
153 RESTONL GCB: Restore Database Incremental from Online Source 963
Conditions 964
Result 965
Syntax 965
Optional Parameters and Subparameters 966
Examples 968
154 RESTORE: Restore Database from Offline Source 969
Conditions 970
Result 970
Syntax 971
Optional Parameters 971
Examples 973
155 RESTORE FILES: Restore Files to Original RABNs from Offline Source 975
Conditions 976
Result 977
Syntax 977
Optional Parameters 978
Examples 980
156 RESTORE FMOVE: Restore Files to Any RABNs from Offline Source 981
Conditions 982
Result 983

Syntax 983
Optional Parameters 984
Examples 990
157 RESTORE GCB: Restore Database Incremental from Offline Source 991
Conditions 992
Result 993
Syntax 993
Optional Parameters 993
Examples 995
158 RESTPLOG: Restore Protection Log Only 997
Essential Parameters 998
Optional Parameters 999
Example 999
159 SAVE: Save Database 1001
Syntax 1003
Optional Parameters 1003
Example 1005
160 SAVE FILES: Save Specified Files 1007
Syntax 1009
Optional Parameters 1010
Examples 1011
161 JCL/JCS Requirements and Examples 1013
BS2000 1014
z/OS 1020
z/VM 1025
VSE 1029
XX ADASEL Utility: Select Protection Data 1035
162 Functional Overview 1037
163 ADASEL Syntax 1039
SELECT Statement 1041
Additional Selection Criteria 1042
date-time 1043
WITH Clause 1043
IF Statement 1044
value-criterion 1046
output-instruction 1051
164 Overriding ADASEL Defaults with Global Parameters 1059
165 JCL/JCS Requirements and Examples 1061
BS2000 1062
z/OS 1063
z/VM 1064
VSE 1065
XXI ADAULD Utility: Unload Files 1067
166 Functional Overview 1069
167 UNLOAD FILE: Unload Specified File 1071

Essential Parameter 1072
Optional Parameters and Subparameters 1072
Examples 1078
168 ADAULD Input Processing 1081
Processing a Save Tape as Input 1082
169 ADAULD Output Processing 1085
170 ADAULD User Exit 9 1087
171 JCL/JCS Requirements and Examples 1089
BS2000 1090
z/OS 1092
z/VM 1094
VSE 1096
XXII ADAVAL Utility: Validate the Database 1099
172 Functional Overview 1101
173 VALIDATE: Validate Data Storage and Associator 1103
Essential Parameters 1104
Optional Parameters 1105
174 Example of ADAVAL Output 1107
175 JCL/JCS Requirements and Examples 1109
ASSO-, DATA-, and Work Data Sets 1110
Collation with User Exit 1110
Sorting Large Files 1111
BS2000 1111
z/OS 1112
z/VM 1113
VSE 1114
XXIII ADAWRK Utility: Work Area Recovery Reports 1117
176 Functional Overview 1119
177 Utility Syntax 1121
178 Report Descriptions 1125
Environment Report 1126
Summary Report 1126
File Report 1128
Transaction Report 1131
Checkpoint Record Reporting 1150
179 JCL/JCS Requirements and Examples 1151
z/OS 1152
VSE 1154
z/VM 1155
BS2000 1156
XXIV ADAZAP Utility: Display or Modify Asso, Data, and Work Data Sets 1159
180 Functional Overview 1161
181 ADAZAP Syntax 1163
Essential Parameters 1164
Optional Parameters 1164

Examples 1165
182 JCL/JCS Requirements and Examples 1167
BS2000 1168
z/OS 1169
z/VM 1170
VSE 1171
XXV AppendicesAppendicesAppendix 1173
A Appendices 1175
Sequential File Table 1176
Operating System Dependencies 1178
B Appendices 1189
Adabas Libraries (ADAVvLIB) 1190
Adabas Files (ADAVvFIL) 1190
C Appendix 1193
Index 1195

Utilities

Each Adabas utility is described in a separate part. For a single-function utility, the part begins with a syntax diagram showing the utility statement and all possible parameters. Parts for utilities with multiple functions begin with a brief overview of the functions, followed by the individual function syntax diagrams and descriptions.

Each function description contains

- syntax diagram with all parameters;
- individual parameter descriptions describing coding rules, restrictions, and defaults; and
- utility function examples.

Following the function descriptions are job control examples for the BS2000, z/OS, z/VM, and VSE operating systems.

Note: Data set names starting with DD are referred to in the Adabas ocumentation with a slash separating the DD from the remainder of the data set name to accommodate VSE data set names that do not contain the DD prefix. The slash is not part of the data set name.

This documentation is organized in the following parts:

ADAACK Utility: Check Address Converter	Describes the ADAACK utility, which allows you to check the address converter for a specific file or range of files or for a specific ISN or range of ISNs.
ADACDC Utility: Changed-Data Capture	Describes the ADACDC utility, which allows you to produce a file containing the delta of all changes made to the database over the period covered by the input protection logs. You can also use this utility to produce a file that lists each individual insert and delete transaction made to the database over the period covered by the input protection logs (without producing a delta of these changes).
ADACMP Utility: Compress-Decompress Data	Describes the ADACMP utility, which allows you to edit and compress data records that are to be loaded into the database. This utility also allows you to decompress individual files in the Adabas database.
ADACNV Utility: Database Conversion	Describes the ADACNV utility, which allows you to convert an Adabas database to a higher version of Adabas or to revert an Adabas database to a lower version of Adabas.
ADADBS Utility: Database Services	Describes the ADADBS utility, which allows you to perform many database definition and maintenance functions.
ADADCK Utility: Check Data Storage and DSST	Describes the ADADCK utility, which allows you to check Data Storage and the Data Storage space table (DSST) of specified files in the database.

ADADEF Utility: Define a Database	Describes the ADADEF utility, which allows you to define a database and checkpoint file or define a new Work file. It also allows you to modify file encodings for the database.
ADAFRM Utility: Format Adabas Data Sets	Describes the ADAFRM utility, which allows you to format Adabas database components.
ADAICK Utility: Check Index and Address Converter	Describes the ADAICK utility, which allows you to check the physical structure of the Associator.
ADAINV Utility: Inverted List Management	Describes the ADAINV utility, which allows you to create descriptors in a file and identify the descriptors used to couple two files.
ADALOD Utility: File Loader	Describes the ADALOD utility, which allows you to load a file into a database and to add or delete a large number of records (ISNs) to or from an existing file.
ADAMER Utility: ADAM Estimation	Describes the ADAMER utility, which allows you to produce statistics that indicate the number of Data Storage accesses required to find and read a record when using an ADAM descriptor.
ADAORD Utility: Reordering Functions	Describes the ADAORD utility, which allows you to reorder the Associator or Data Storage for a database or specified files. This utility also allows you to restructure a database or file.
ADAPLP Utility: Protection Log/Work Print	Describes the ADAPLP utility, which allows you to print data protection records contained on the Adabas Work data set or the Adabas data protection log.
ADAPRI Utility: Print Selected Adabas Blocks	Describes the ADAPRI utility, which allows you to print the contents of a block (or range of blocks) contained in the Associator, Data Storage, Work, temp, sort, multiple data set command log, multiple data set protection log, or the recovery log data set.
ADARAI Utility: Adabas Recovery Aid	Describes the ADARAI utility, which allows you to manage recovery logging.
ADAREP Utility: Database Status Report	Describes the ADAREP utility, which allows you to produce the database status report.
ADARES Utility: Database Recovery	Describes the ADARES utility, which allows you to perform database recovery functions.
ADASAV Utility: Save/Restore Database or Files	Describes the ADASAV utility, which allows you to save or restore a database or specific database files.
ADASEL Utility: Select Protection Data	Describes the ADASEL utility, which allows you to select and decompress information in the Adabas sequential (SIBA) or dual/multiple (PLOG) protection log and write it to a print data set (DDDRUCK/ DRUCK) or a user-specified output data set.
ADAULD Utility: Unload Files	Describes the ADAULD utility, which allows you to unload an Adabas file.
ADAVAL Utility: Validate the Database	Describes the ADAVAL utility, which allows you to validate any or all files within an Adabas database except the checkpoint and security files.
ADAWRK Utility: Work Area Recovery Reports	Describes the ADAWRK utility, which allows you to produce reports from records in the autorestart area of Work part 1. This information

	can be used when the database autostart fails and the database will not come up. It can help you determine how to handle database recovery.
ADAZAP Utility: Display or Modify Asso, Data, and Work Data Sets	Describes the ADAZAP utility, which allows you to display (in hexadecimal format) and optionally change the contents of the Associator, Data Storage, or Work data sets.
ADABAS Sequential Files	Describes the sequential files used by the Adabas utilities as well as characteristics of file and device definitions by operating system..
Libraries and File Procedures for VSE Examples	Lists the Adabas libraries and files (ADAVvFIL) that should be cataloged into an accessible procedure library for the VSE examples.
Adabas Personnel Demo File	Provides the FDT for the Personnel demo file distributed with Adabas.

1 Conventions

- Control Statement Syntax 2
- Syntax Conventions 3
- Control Statement Rules 5
- Parameter Values 5

This document covers the following topics:

- Control Statement Syntax
- Syntax Conventions
- Control Statement Rules
- Parameter Values

Notation *vrs*, *vr*, or *v*: When used in this documentation, the notation *vrs* or *vr* stands for the relevant version of a product. For further information on product versions, see *version* in the *Glossary*.

Control Statement Syntax

Utility control statements have the following format:

where

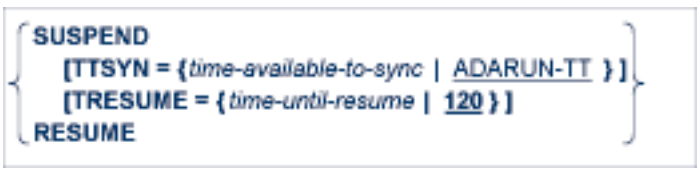
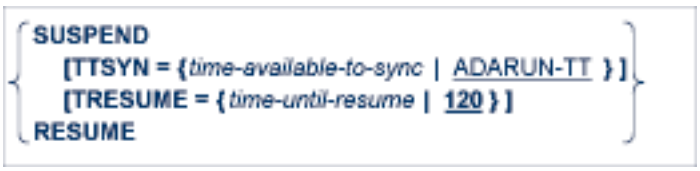
utility	is the name of the utility to be executed. Examples of utility names include: ADAORD ADADBS ADAINV
function	is the name of the specific utility operation to be executed. For example: ADAORD REORDATA ADADBS ADD ADAINV COUPLE Most single-function utilities (ADASEL, ADAULD, etc.) whose function is implicit have either no function value or an optional one.
parameter-list	is a list of parameters following the function. Parameters in the list are almost always keywords with the format: `parameter=value` A parameter may have one or more operands, and keyword parameters may be specified in any order. Most parameters require that you select or otherwise specify an operand value. Some operands are positional (value1 , value2 ,..., valuex), meaning that the values must be in a certain order as described in the text. All parameters must be separated by commas.

	In the statement syntax descriptions in this documentation, parameters are listed vertically (stacked) or are separated by vertical bars (|). Each list shows all possible parameters, from which one or more can (or must) be specified. Although parameters in the list must be separated by commas, these commas are omitted in the syntax statements when the parameters are stacked.

Syntax Conventions

The following table describes the conventions used in syntax diagrams of Adabas statements.

Convention	Description	Example
uppercase, bold	Syntax elements appearing in uppercase and bold font are Adabas keywords. When specified, these keywords must be entered exactly as shown.	**ADADBS CHANGE FILE =** *file-number* The syntax elements ADADBS, CHANGE, and FILE are Adabas keywords.
lowercase, italic, normal font	Syntax elements appearing in lowercase and normal, italic font identify items that you must supply.	**ADADBS CHANGE FILE =** *file-number* The syntax element *file-number* identifies and describes the kind of value you must supply. In this instance, you must supply the number of the file affected by the ADADBS CHANGE operation.
mixed case, normal font	Syntax elements appearing in mixed case and normal font (not bold or italic) identify items established by other Adabas control statements. This notation is usually used to identify how default values are determined for some parameters in Adabas syntax.	**[SORTDEV = {** *device-type* | ADARUN-device **}]** The syntax element "ADARUN-device" indicates that the device type identified by the ADARUN DEVICE parameter will be used if a different device type is not specified. The literal "ADARUN-device" should *not* be specified for the SORTDEV parameter.
underlining	Underlining is used for two purposes: 1. To identify default values, wherever appropriate. Otherwise, the defaults are explained in the accompanying parameter descriptions.	**[LRECL = {***record-buffer-length* | **4000 }]** In the example above, 4000 is the default that will be used for the LRECL parameter if no other record buffer length is specified. **DEVICE**

Convention	Description	Example
	2. To identify the short form of a keyword.	In the example above, the short version of the DEVICE parameter is DE.
vertical bars (|)	Vertical bars are used to separate mutually exclusive choices. **Note:** In more complex syntax involving the use of large brackets or braces, mutually exclusive choices are stacked instead.	 In the example above, you must select RESTRUCTUREF or REF for this ADAORD function. There are no defaults.
brackets ([ ])	Brackets are used to identify optional elements. When multiple elements are stacked or separated by vertical bars within brackets, only one of the elements may be supplied.	 In this example, the SORTSEQ parameter and the MU, NU, and STARTISN subparameters are optional. **Note:** Note that the mutually exclusive choices for the SORTSEQ parameter are stacked.
braces ({ })	Braces are used to identify required elements. When multiple elements are stacked or separated by vertical bars within braces, one and only one of the elements must be supplied.	 In this example, either the SUSPEND or RESUME parameter is required.
indentation	Indentation is used to identify subparameters of a parameter.	 In this example, TTSYN and TRESUME are subparameters of the SUSPEND parameter.
ellipsis (...)	Ellipses are used to identify elements that can be repeated. If the term preceding the ellipsis is an expression enclosed in square brackets or braces, the ellipsis applies to the entire bracketed expression.	 In this example, the FIELD parameter can be repeated. In addition, more than one option can be associated with a field.

Convention	Description	Example
other punctuation and symbols	All other punctuation and symbols must be entered exactly as shown.	[FIELD = 'field-name [, option]... '] ... In this example, the single quotation marks must be specified around the field definitions and their associated options. In addition, options must be separated by commas.

Control Statement Rules

The following rules apply for the construction of utility control statements:

1. Each control statement must contain a utility name in positions one through six.
2. The utility function name follows the utility name, separated by at least one space.
3. Keyword parameter entries and multiple values within keyword entries must be separated by commas.
4. No space is permitted before or after an equals symbol (=).
5. The comma following the last parameter entry of a statement is optional.
6. Control statement processing ends with position 72 or when a space is encountered after the beginning of the parameter list. Entries made in positions 73-80 are not processed.
7. A statement that contains an asterisk (*) in position one is read as a comment and is not processed.
8. Control statements are continued by specifying the extra parameters on a new statement following (and separated by at least one space from) the utility name in positions one through 6.

Parameter Values

Variable values actually specified following the equals symbol (=) in parameters (represented by italicized labels in the preceding examples and elsewhere in this documentation) have the following syntax:

```
parameter = value
parameter = value-list
parameter = value-range
```

where *value* is as described in the following sections. Parameters *value-list* and *value-range* are variations of *value*, and are allowed either in place of or with *value*, depending on the individual parameter rules as described in the text.

value

The *value* parameter may consist of a number or a string of alphanumeric or hexadecimal characters. In some optional keyword parameters, a default value is assumed if the parameter is not specified.

Alphanumeric Values

Alphanumeric values are specified in one of the following ways:

If the value comprises . . .	Apostrophes around it are . . .
only upper- or lowercase letters, numeric digits and minus (-)	optional
any other characters including an apostrophe itself (which must be entered twice)	required

Numeric Values

Numeric values are specified as follows:

If the value represents . . .	Specify . . .
a number of either blocks or cylinders	the letter B must immediately follow the value if blocks are being specified; otherwise, cylinders are assumed: SIZE=200B (200 blocks) SIZE=200 (200 cylinders)
an Adabas file	a one- to four-digit number (leading zeros permitted): FILE=3 FILE=03 FILE=162
a device type	a four-digit number corresponding to the model number of the device type to be used: DEVICE=3380
a field name or descriptor	a two-character field name corresponding to the field name or descriptor: FIELD1=NA

Hexadecimal values are accepted if this is specified in the parameter description. Hexadecimal values must be within apostrophes following the indicator X:

```
X'0002DC9F'
```

value-list

value,... (numeric values)

```
BITRANGE=2,10,2
```

or

'value,...' (alphanumeric values)

```
UQDE='AA,AC,AE'
```

value-range

value - value, ...

```
ISN=600-900,1000-1200
```

Individual values within a value list or value range may be positional if they relate to values specified on corresponding parameters. For example:

```
ADADBS  UNCOUPLE  FILES=13,20,PASSWORD='PW13,PW20'
```

-instructs the ADADBS UNCOUPLE function to uncouple files 13 and 20, which are password-protected.

The passwords (specified by the PASSWORD parameter) must be in the same order as their corresponding files in the FILES parameter.

If file 13 is *not* password-protected, either the PASSWORD parameter must be specified with a placeholder comma as shown below

```
... PASSWORD=',PW20'
```

-to position the password "PW20" to the corresponding position of file 20 in the FILES value list, or FILES must specify file 20 first.

I ADAACK Utility: Check Address Converter

This chapter covers the following topics:

Functional Overview
ACCHECK: Check Address Converter Against Data Storage
JCL/JCS Requirements and Examples

2 Functional Overview

ADAACK checks the address converter for a specific file or range of files or for a specific ISN or range of ISNs. If spanned records are in use, ADAACK assumes any ISNs passed to it are primary ISNs, and performs its processing accordingly. The ranges can encompass all files or all ISNs. ADAACK is used in conjunction with ADAICK.

If the file being checked has spanned records enabled, the secondary address converter, used to map the secondary ISNs to the RABNs of the secondary records, will automatically be checked as well. For more information about spanned records, read *Spanned Records*, in *Adabas Concepts and Facilities Manual*.

ADAACK checks each address converter element to determine whether the Data Storage RABN is within the used portion of the Data Storage extents specified in the file control block (FCB).

ADAACK checks the ISN for each record in each Data Storage block (within the specified ISN range) to ensure that the address converter element for that ISN contains the correct Data Storage RABN. This is done in the following way:

1. Main memory is allocated for the specified range of ISNs (number of ISNs, times 4). If no range is specified, the entire range (MINISN through TOPISN) is checked.

   The address converter is read from the database into this area in memory.
2. Each used Data Storage block (according to the Data Storage extents in the FCB) is read and checked against the address converter in memory. Each ISN in the address converter must have exactly one associated Data Storage record.
3. The address converter in memory is checked for ISNs that did not occur in Data Storage.

For large files, ADAACK may run for a long time. ADAACK prints a message line after every 20 Data Storage blocks processed.

Run time is not affected by the ISN range, since all used Data Storage blocks are read.

Notes:

1. ADAACK does not require the Adabas nucleus to be active.
2. A pending autorestart condition is ignored.
3. If the nucleus is active, ADAACK synchronizes its operation with the active nucleus unless the NOOPEN parameter is specified.
4. This utility should be used only for diagnostic purposes.

ADAACK returns a condition code 8 if any errors occur.

3 ACCHECK: Check Address Converter Against Data Storage

- Optional Parameters 14
- Examples 15

```
ADAACK  ACCHECK  [FILE={ file | file - file |  all-files  } ]
                 [ISN= { isn - isn |  all-isn  } ]
                 [NOOPEN]
                 [NOUSERABEND]
```

This chapter describes the syntax and parameters of the ACCHECK function.

Optional Parameters

FILE: Files to be Checked

The file, single range of files, or all files to be checked. By default, all files in the database are checked.

ISN: ISN Range to be Checked

A range of ISNs or all ISNs to be checked. By default, the entire range MINISN through TOPISN is checked.

If spanned records are in use, ADAACK assumes that any ISNs passed to it are primary ISNs, and performs its processing accordingly. If an ISN is the primary ISN of a spanned Data Storage record, Adabas will automatically check the appropriate segments records for the spanned record in the secondary address converter (AC2).

When printing error information about a particular ISN, the ADAACK utility will now indicate whether the problem is with a primary or secondary ISN, if the record is spanned.

NOOPEN: Prevent Open Resynchronization

When starting, ADAACK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAACK from issuing the open call.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

Examples

Example 1:

```
ADAACK ACCHECK
```

Check all files in the database.

Example 2:

```
ADAACK ACCHECK FILE=12, ISN=1-8000
```

Check ISNs 1 through 8000 for file 12.

Example 3:

```
ADAACK ACCHECK FILE=8-10
```

Check all ISNs in files 8 through 10.

4 JCL/JCS Requirements and Examples

- BS2000 18
- z/OS 19
- z/VM 20
- VSE 21

This section describes the job control information required to run ADAACK with BS2000, z/OS , z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAACK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAACK messages	SYSLST DDDRUCK		*Messages and Codes*

ADAACK JCL Examples (BS2000)

In SDF Format:

```
/.ADAACK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A A C K ADDRESS CONVERTER CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.ACK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAACK,DB=yyyyy,IDTNAME=ADABAS5B
ADAACK ACCHECK FILE=ffff
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAACK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A A C K ADDRESS CONVERTER CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.ACK.DATA
```

```
/FILE ADAvrs.MOD ,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAACK,DB=yyyyy,IDTNAME=ADABAS5B
ADAACK ACCHECK FILE=ffff
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADAACK messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations*
ADAACK parameters	DDKARTE	reader	

ADAACK JCL Example (z/OS)

```
//ADAACK    JOB
//*
//*   ADAACK:
//*      ADDRESS CONVERTER CHECK
//*
//ACK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD        <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1  <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1  <=== DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAACK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAACK ACCHECK FILE=ffff
/*
```

Refer to ADAACK in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADAACK messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAACK parameters	DDKARTE	disk/ terminal/ reader	

ADAACK JCL Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAACK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAACK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNACK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAACK.CONTROL,MODE=A
ADARUN
```

Contents of RUNACK CONTROL A1:

```
ADARUN PROG=ADAACK,DEVICE=dddd,DB=yyyyy
```

Contents of ADAACK CONTROL A1:

```
ADAACK ACCHECK FILE=ffff
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
ADAACK messages		printer	SYS009	*Messages and Codes*
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADARUN parameters	CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAACK parameters		reader	SYSIPT	

* *Any programmer logical unit may be used.*

ADAACK JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

```
* $$ JOB JNM=ADAACK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       ADDRESS CONVERTER CHECK
// JOB ADAACK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAACK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAACK ACCHECK FILE=27
/*
/&
* $$ EOJ
```

Refer to member ADAACK.X for this example.

II ADACDC Utility: Changed-Data Capture

This chapter covers the following topics:

Functional Overview
Running the Utility
Operating System Considerations
The ADACDC User Exit
Examples
JCL/JCS Requirements and Examples

5 Functional Overview

- Phases of Operation and Resulting Files 26
- Primary Input Data 29
- Primary Output File 29
- Transaction File 30

The ADACDC utility takes, as input, one or more sequential protection logs and produces ultimately, as output, a *primary output file* containing the delta of all changes made to the database over the period covered by the input protection logs. *Delta of changes* means that the last change to each ISN in a file that was altered during this period appears on the primary output file.

Notes:

1. If the ADACDC ISN parameter is specified for the run, the delta is not produced. Instead, each individual delete and insert transaction is written to the primary output file and no delta of changes is produced.
2. Spanned records are not supported by the ADACDC utility at this time. However, when the `IGNORESPANNED` parameter is specified in an ADACDC run, ADACDC processing ignores any spanned records, issues a warning message, and continues its processing. A return code of "4" is returned.

The data in the primary output file output from an ADACDC run may be used on a regular basis as input for data warehousing population procedures so that the changes to a database are applied to the data warehouse database rather than a copy of the entire database. This affords more frequent and less time-consuming updates to the data warehouse, ensuring greater accuracy of the information stored there.

In order to run the ADACDC utility:

- an external sorter must be available and installed as the standard sorter in the operating system. See **Operating System Considerations** for more information.
- the external sorter must have access to the database's Associator containing the FDTs of the files for which records are to be processed.

ADACDC uses this sort package to produce its output in ISN sequence, so all changes are written to the primary output file in ISN sequence.

Phases of Operation and Resulting Files

ADACDC processes sequential protection logs in two phases. You can execute phase 1 and phase 2 separately, or both at once (the default):

- If phase 2 is being run separately or both phases are being completed together, the data is decompressed and written to the *primary output file*.
- If only phase 1 is being executed, the data is written to an *extract file* . This extract file may then be processed multiple times by a phase 2 operation to decompress the records and write to primary output files.

The extract file contains data records in compressed format whereas the primary output file contains records in decompressed format. Refer to the section *ADACMP (Compress - Decompress)* in the *Adabas Utilities* documentation for more information about these formats.

The primary output file and the extract file are standard operating-system-dependent files that can handle variable length records.

Phase 1 and the Extract File

During phase 1, updates from the protection logs are analyzed and prefixed with a standard structure called the CDCE. The format of each record on the file is a constant CDCE prefix followed by the compressed record information.

Usually, these records are passed to an external sort routine to establish the most recent update for each ISN on a file. Only the last change for a given file and ISN combination is written to the extract file. However, if the ISN parameter is specified for the ADACDC run, the updates are still sorted in ISN order, but they are not summarized. Instead, every change transaction for an ISN is recorded in the extract file.

The extract file created when phase 1 is run separately makes it possible to process the PLOG data once and then optionally produce multiple primary output files from it based, for example, on file selection criteria. The option is useful if different file changes are required for different purposes.

When the phase 1 process is being run, the extract file is opened for output. As records are output from the sort processing, the updates for each file and ISN combination is written to the extract file if:

- the update was performed by an ET user and belongs to a completed transaction; or
- the update was performed by an EXU user and belongs to a completed command; or
- NOET is specified.

All other updates for the file and ISN combination for that period are discarded if there are no controlled utility operations against that file (see ***Checkpoints Written to the Primary Output File***).

Note: It is possible to have duplicate file and ISN combinations on the file if the ADACDC user exit (described later) adds records with file and ISN combinations that already exist. A record added or modified by the user exit is so marked in the CDCE structure.

Phase 2 or Both and the Primary Output File

The primary output file is used when both stages of ADACDC are run together, or for phase 2 processing only.

- If both phases are run together, the primary output file is opened and created directly using the output from the sort processing. In this case, processing occurs as for the extract file in phase 1 processing.
- If only phase 2 is run, the primary output file is created using input from the extract file.

The format of each record on the file is a constant CDCO prefix followed by the decompressed record information. If for some reason the record cannot be decompressed, a warning message is issued and the compressed record is written to the primary output file. A flag in the CDCO structure informs a user program when decompression for the record has failed.

Note: It is possible to have duplicate file and ISN combinations on the file if the ADACDC user exit (described later) adds records with file and ISN combinations that already exist. A record added or modified by the user exit is so marked in the CDCO structure.

Checkpoints Written to the Primary Output File

The primary objective of the ADACDC utility is to provide an output file containing the most recent summarized changes for each ISN in a file that has been modified for the period concerned. If the ISN parameter is specified for the run, the primary objective of the ADACDC utility is to provide an output file containing the changes for each ISN in a file that has been modified for the period concerned.

Apart from simple changes to a file, some utility operations executed against a file may fundamentally affect its contents. For example, if the file is deleted, simply providing the last updates for ISNs in the file does not accurately reflect the state of the file since all ISNs have been deleted.

For this reason, the following checkpoints are recorded and written to the primary output file as appropriate with the associated indication in the output record:

```
ADASAV RESTORE FILE          File created
ADAORD STORE FILE            File created
ADALOD LOAD FILE             File created
ADALOD UPDATE FILE           File updated
ADADBS DELETE FILE           File deleted
ADADBD REFRESH FILE          File deleted
```

Because these operations can fundamentally impact a file and its appearance, the checkpoint is written to the primary output file when it occurs relative to the other updates.

ADACDC retains the last change to all ISNs before each of the above checkpoints. This means that a file and ISN combination could appear multiple times on the primary output file if one or more

checkpoints were written to it. This is useful for the many data warehouse packages that may wish to complete their view of a file and maintain a copy of it prior to deletion, recreation, or mass update.

Primary Input Data

The primary input data comprises sequential protection logs produced either by the database directly or by the ADARES PLCOPY of nonsequential protection logs.

Note: Spanned records are not supported by the ADACDC utility at this time. However, when the `IGNORESPANNED` parameter is specified in an ADACDC run, ADACDC processing ignores any spanned records, issues a warning message, and continues its processing. A return code of "4" is returned..

ADACDC processes this data to ensure that:

- when a new PLOG block is read and the PLOG number is the same, the PLOG block number is 1 greater than the previous PLOG block number.
- when the PLOG number itself changes, the new PLOG number is higher than the previous PLOG number and the new PLOG block number is 1.

Note: When the PLOG number changes and the difference between the PLOG numbers is greater than 1, a warning message is issued and processing continues as this can legitimately happen if online saves are used.

If any of these checks fail, the utility execution terminates.

Primary Output File

The primary output file is a sequential file comprising all database records that were added, updated, or deleted during the period covered by the input protection logs.

If a record was changed several times, only its last change appears in the output file; ADACDC employs a sort process to identify multiple changes to the same record. However, if the ISN parameter is specified for the ADACDC run, all changes for an ISN appear in the primary output file; ADACDC still employs the sort process to put the primary output file in ISN sequence.

Each primary output file record comprises a fixed-length record prefix followed by the database record in decompressed form. The decompressed data corresponds in format to the output of the **ADACMP DECOMPRESS** function.

The primary output record prefix is described by the CDCO DSECT. It has the following structure:

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>0-1</td><td>record length (binary)</td></tr>
<tr><td>2-3</td><td>set to zeros</td></tr>
<tr><td>4-7</td><td>constant 'CDCO'</td></tr>
<tr><td>8-9</td><td>database ID</td></tr>
<tr><td>10-11</td><td>file number</td></tr>
<tr><td>12-15</td><td>ISN of the updated record</td></tr>
<tr><td>16-19</td><td>length of the decompressed data in bytes</td></tr>
<tr><td>20-47</td><td>28-byte communication ID of the last user who updated the record</td></tr>
<tr><td>48</td><td>change indicator:<table><tr><td>X'04'</td><td>record added</td></tr><tr><td>X'08'</td><td>record updated</td></tr><tr><td>X'0C</td><td>record deleted</td></tr><tr><td>X'10'</td><td>file created</td></tr><tr><td>X'14'</td><td>file updated</td></tr><tr><td>X'18'</td><td>file deleted or refreshed</td></tr></table></td></tr>
<tr><td>49</td><td>flags (independent bit settings):<table><tr><td>X'80'</td><td>record added by user exit</td></tr><tr><td>X'40'</td><td>record modified by user exit</td></tr><tr><td>X'20'</td><td>record still compressed; decompression failed</td></tr></table></td></tr>
<tr><td>50</td><td>database version indicator</td></tr>
<tr><td>51</td><td>reserved for future use</td></tr>
<tr><td>52-59</td><td>4-byte STCK, followed by a 4-byte hexadecimal counter. Users can sort on this 8-byte field to put the primary output file records back into PLOG sequence, when necessary. Read Using ADACDC With ISNREUSE, elsewhere in this section, for more information about when this might be necessary.</td></tr>
<tr><td>60-67</td><td>reserved for future use</td></tr>
<tr><td>68-...</td><td>decompressed record data</td></tr>
</table>

Transaction File

To maintain input data checking over multiple runs of the utility, ADACDC stores information on the transaction file in a transaction control record containing the last database ID, the PLOG number, and the PLOG block number processed. This information is used to verify the latest input (unless the RESETTXF option is specified - see section ***RESETTXF : Reset Input Transaction File*** in ***ADACDC Optional Parameters***, elsewhere in this section.

ADACDC actually recognizes two different transaction files: input and output. Both transaction files are standard operating-system-dependent files that can handle variable length records.

Input Transaction File Processing

During the input processing stage, ADACDC processes the input transaction file to the sort program.

Following the control record on the input transaction file, zero or more records may be found. These are database updates related to transactions not completed during the last run of the utility. These records are processed again as part of the input as their transactions will normally have been completed in the next sequential protection logs provided to the utility. This is the reason the sequence of protection logs is so important: updates may remain outstanding forever if the correct sequence is not maintained.

The transaction file also records whether the NOET option was specified during the last phase 1 run of the utility. When ADACDC detects that this option has changed from one utility execution to the next, it uses the information from the control record on the input transaction file; however, all transactional information in the other records is ignored. This is due to the fact that changing this option may cause inconsistent data to be written to the primary output file or extract file, as appropriate. ADACDC issues a warning when this happens.

Output Transaction File Processing

Once output processing from the sort program starts, the input transaction file is closed and the output transaction file is opened. The control record is written to the output transaction file followed by any updates that relate to incomplete transactions or, in the case where the NOET option is in effect or an EXU user is in control, to incomplete commands. The output transaction file is closed once processing is complete.

Using a Single Transaction File

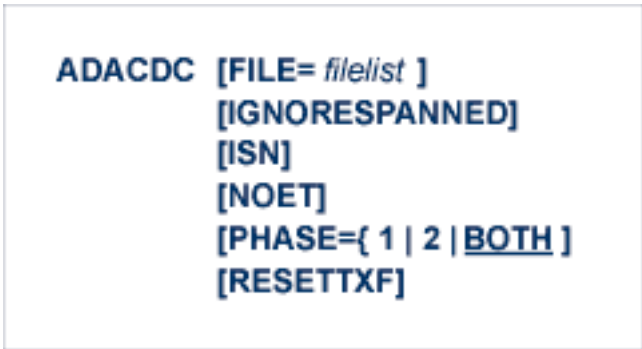
It is possible to use the same file as both the input and output transaction file; however, if the utility fails while writing to the output transaction file (that is, at any time during the output processing of the sort utility), the input transaction file will no longer exist and therefore, rerunning the utility will yield a different result.

For this reason, the transaction file must be backed up prior to the utility run so that it can be restored in the event of a failure.

Alternatively, you could use a facility on your operating system (if available) that produces a new version of a file whenever a program updates the file.

6 Running the Utility

- Optional Parameters 34
- Using ADACDC With ISNREUSE 36



The first time you run the ADACDC utility, use the following syntax and either do not specify or dummy the input transaction file (CDCTXI) to create a valid transaction file for input to subsequent ADACDC runs:

```
ADACDC RESETTXF,PHASE=BOTH
```

The RESETTXF option ignores the absent or dummied input transaction file, reads the primary input data, and produces primary output using the input data.

After the input transaction file has been created during the first run, only the utility name ADACDC is required to run this utility; the PHASE parameter defaults to BOTH. Parameter options are explained in the following sections.

Optional Parameters

FILE: Files Processed

Use the FILE parameter to limit the file(s) processed by the utility and written to the output file:

- For phase 1 operation, only records relating to the files specified are written to the extract file.
- For phase 2 and BOTH operations, only records relating to the files specified are written to the primary output file.

Note: Clearly, files required for phase 2 processing must have been specified on the previous phase 1 operation that created the input extract file.

When this parameter is not specified, all files are processed by the utility.

IGNORESPANNED: Ignore Spanned Records

Spanned records are not supported by the ADACDC utility at this time. However, when the `IGNORESPANNED` parameter is specified in an ADACDC run, ADACDC processing ignores any spanned records, issues a warning message, and continues its processing. A return code of "4" is returned.

ISN: Record Delete and Insert Transactions Separately

Ordinarily, ADACDC processing consolidates all delete and insert transactions to the same ISN, creating a single update transaction for the ISN. However, if you specify the ISN parameter, each delete and insert transaction is recorded in the primary output file (CDCOUT) individually. So, when you use the ADACDC utility with the ISN parameter, the number of records produced in the primary output file will increase, possibly dramatically.

NOET: Bypass ET Processing

ADACDC normally accepts for processing only those records that are part of completed transactions or, in the case of EXU users, records that are part of completed commands.

Use the NOET option to bypass this transaction processing when PHASE=1 or PHASE=BOTH. NOET has no effect when PHASE=2 because the input is the extract file from phase 1 which has already processed the protection log (PLOG) input with or without the NOET option.

When NOET is specified, any update made to the database is processed and written to the extract file (PHASE=1) or primary output file (PHASE=BOTH) as soon as it is encountered on the PLOG.

Caution: Specifying this option may result in updates recorded on the primary output file that are related to transactions that were not complete at the end of the input PLOG.

PHASE: Execution Phase

The PHASE parameter determines the input the utility requires and the output it produces:

- PHASE=1 reads the sequential PLOG input and produces an interim extract file for later processing by a phase 2 step.
- PHASE=2 reads an extract file produced by a previously executed phase 1 step and produces a primary output file containing the delta of changes made to the file.
- PHASE=BOTH (the default) reads the sequential PLOG input and produces the primary output file containing the delta of changes directly without reading or writing an extract file.

Refer to the section ***Phases of Operation and Resulting Files***, elsewhere in this section, for more information.

RESETTXF: Reset Input Transaction File

ADACDC checks the primary input data to the utility to ensure that the PLOGs are read in sequence, by PLOG block and PLOG number. If these checks fail, the utility execution terminates.

To maintain the checks over multiple runs of the utility, ADACDC maintains input and output transaction files. These files also track record updates related to incomplete transactions or, in the case of EXU users, incomplete commands from one utility execution to the next. Normally, such incomplete transactions or commands are completed in the next sequential PLOGs provided to the utility.

However, if the need arises to process PLOGs out of sequence and the information in the transaction file can be safely removed, the RESETTXF option can be used to reset the transaction file so that the checks are bypassed and all outstanding transaction or command data is ignored for a given run. ADACDC ignores information on the input transaction file and writes the output transaction file at end of job.

Caution: If the sequence of PLOGs is interrupted, record updates related to incomplete transactions recorded in the transaction file may remain outstanding indefinitely.

Using ADACDC With ISNREUSE

Normal ADACDC processing produces a primary output file in ISN sequence. Ordinarily, this processing works fine. However, if the database file was created with the ADADBS or ADALOD ISNREUSE option specified, errors (response 98) can occur. These errors can occur because an ISN might have been reused, so multiple transactions may reside in the PLOG for the same unique descriptor key (UQ) with different ISNs.

To resolve these problems, the following steps should be taken:

1 Run the ADACDC utility with the ISN parameter specified. This will give you a granular list of changes in the primary output file, instead of attempting to summarize the changes by ISN. The data in the primary output file after this run will still be in ISN sequence.

2 Sort the primary output file (CDCOUT) in PLOG sequence prior to applying its data to your data warehouse or other application. This sort should be performed on the CDCOUT data at offset 52 for 8 bytes.

Once the data is sorted in PLOG sequence, the data can be applied to your data warehouse or other application.

3 Sort the transaction file (CDCTXI/CDCTXO) back into PLOG sequence prior to running any additional ADACDC jobs. If you neglect to do this, future runs of ADACDC may be compromised.

To get the transaction file (CDCTXI/CDCTXO) in the correct sequence, two sorts are actually required, in the following sequence:

1. First run a sort that puts the CDCE records in PLOG sequence. This sort should be run on offset 68 for 4 bytes and then on offset 16 for 4 bytes. For example:

```
SORT FIELDS=(68,4,BI,A,16,4,BI,A),RECORD TYPE=V,LENGTH=32756
```

2. The second sort should sort the transaction file back into CDCC, CDCE, CDCX order. This sort should be run on offset 4 for 4 bytes.

When these two sorts have been run, the transaction file should be ready to be processed by future ADACDC jobs.

7 Operating System Considerations

- z/OS 40
- VSE 40
- BS2000 41

For its sort requirements, the ADACDC utility uses a standard sort function installed in the operating system. The following additional considerations should be taken into account for each operating system.

Note: Regardless of platform, special sorts of the primary output file and the transaction file are required if ADACDC is to be run for a file for which ISNs can be reused (ADADBS or ADALOD ISNREUSE is specified). For complete information, read ***Using ADACDC With ISNREUSE***, elsewhere in this chapter.

z/OS

No additional job steps are required by ADACDC when the sort function is invoked. However, depending on the amount of data to be sorted, the ADACDC job step may require additional sort-related DD statements for work files or for other sort-specific facilities. Refer to the sort documentation for more details.

Note: A sort package generally supplies summary information when a SYSOUT DD statement is specified.

When ADACDC invokes sort, it expects by default to transfer control to a load module named 'SORT'. If the sort module has a different name, you must reassemble and link the Adabas options module ADAOPD, specifying the name of the external sort program as follows:

1. Modify the OPDOS member, specifying the name of the sort program in parameter SORTPGM=.
2. Modify and run member ASMLOPD to assemble and link the module ADAOPD.

VSE

Whenever an external sort may be called, an ADACDC utility job must reserve space in the partition area. The EXEC statement must therefore specify the SIZE parameter as either

```
// EXEC ADARUN,SIZE=(ADARUN,128K)
```

or

```
// EXEC ADARUN,SIZE=(AUTO,128K)
```

No additional job steps are required by ADACDC when the sort function is invoked. However, depending on the amount of data to be sorted, the ADACDC job step may require additional sort-related DLBL statements for work files or for other sort-specific facilities. Refer to the sort documentation for more details.

When ADACDC invokes sort, it expects by default to transfer control to a load module named 'SORT'. If the sort module has a different name, the Adabas options module ADAOPD must first be reassembled and relinked with the correct name of the sort module in parameter SORTPGM. See *Modify, Assemble, and Link the Adabas Options Table* in the section *VSE Systems Installation* of the Adabas Installation documentation for VSE.

BS2000

The Fujitsu Technology Solutions external sort may be called for large sort operations. The following job cards are required.

```
/SET-FILE-LINK BLSLIBnn,$.SORTLIB
/SET-FILE-LINK SORTWK1,#SORTWK,BUF-LEN=STD(2),OPEN-MODE=OUTIN
/CREATE-FILE #SORTWK,PUB(SPACE=(&PRIM,&SEC))
/START-PROGRAM .....,RUN-MODE=ADVANCED,ALT-LIBRARY=YES
```

where

nn	is a value between 00 and 99
#SORTWK	was created with the BS2000 command
&PRIM	is the number of primary PAM pages to allocate
&SEC	is the number of secondary PAM pages to allocate

Note: The size of the SORTWK1 file depends on the amount of data to be sorted.

8 The ADACDC User Exit

- Installing the Exit 44
- User Exit Interface 44
- User Exit Calls 46

ADACDC calls a user exit at various points in its processing, providing you with the opportunity to intercede in that processing.

Installing the Exit

To install the user exit

1 Compile the user exit you wish ADACDC to use as module name ADACDCUX.

2 Make the module available to the ADACDC utility.

A sample user exit called ADACDCUX is provided on the source data set. The only function of the sample is to show you how to add, delete, and update records using the user exit interface.

User Exit Interface

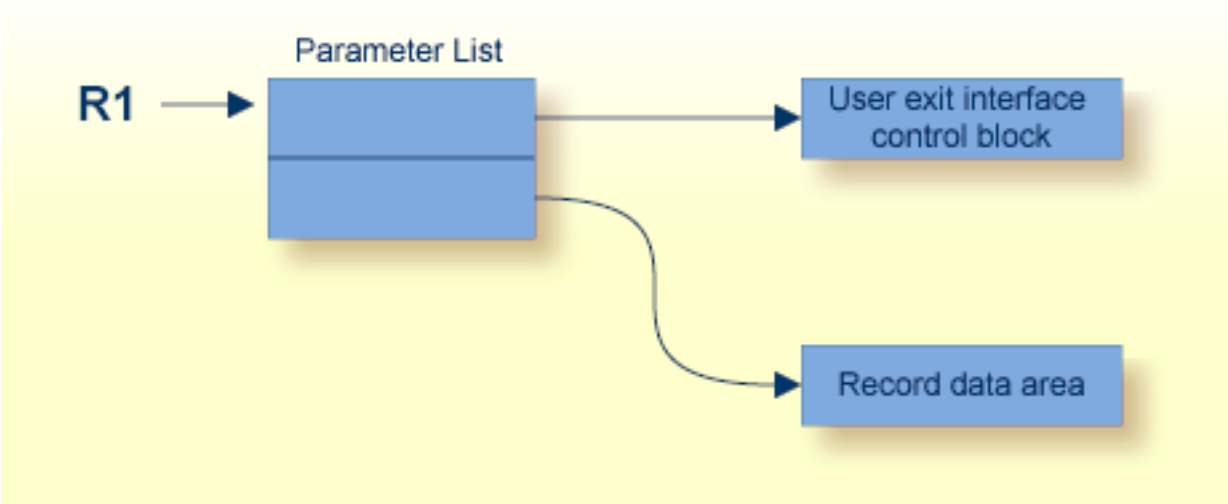
The user exit is called with the following registers set:

R1	user parameter list
R13	standard 72-byte register save area
R14	return address
R15	entry point

The user parameter list contains two pointers:

- the first to the ADACDC user exit parameter list mapped by the CDCU DSECT; and
- the second to the record area for the user exit where the record being processed is passed as appropriate.

The action to be performed is indicated in the CDCUFUNC field whereas the action the user exit directs ADACDC to take on return is indicated using the CDCURESP field.



ADACDC User Exit

The structure of the ADACDC user exit interface control block (CDCU DSECT) is as follows:

Bytes	Description	
0-3	constant 'CDCU'	
4-7	available for use by user exit	
8-11	length of record in second parameter	
12	function identifier:	
	X'00'	initialization
	X'04'	before pass to SORT input
	X'08'	before write to extract file
	X'0C'	before write to primary output file
	X'10'	termination
13	response code from user exit:	
	X'00'	normal processing
	X'04'	ignore this record
	X'08'	record has been updated
	X'0C'	insert new record
14-31	reserved for future use	

User Exit Calls

The following subsections describe the calls made to the user exit and their purpose.

Initialization Call (CDCUFUNC=CDCUINIT)

During initialization, ADACDC calls the user exit so that it can set up any areas it requires for future processing. The CDCUUSER field is provided in the CDCU for anchoring a user control block, if appropriate.

The record area pointer points to data that has no relevance for this call.

Termination Call (CDCUFUNC=CDCUTERM)

During termination, ADACDC calls the user exit so that it can close any open files or clean up any areas still outstanding after ADACDC execution. For example, if an anchor pointer was set in CDCUUSER, this area could be freed and the CDCUUSER field set to nulls.

The record area pointer points to data that has no relevance for this call.

SORT Input Call (CDCUFUNC=CDCUINPT)

ADACDC calls the user exit before a record is passed to the SORT routine as input.

The record area pointer points to the compressed data record to be returned prefixed by the CDCE control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer is passed to the SORT routine. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next input record.

Extract Output Call (CDCUFUNC=CDCUWRTE)

ADACDC calls the user exit before a record is written to the extract file during phase 1 processing. This exit point is *only* called during phase 1 processing and has no relevance in other cases.

The record area pointer points to compressed the data record to be written prefixed by the CDCE control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer on return is written to the extract file. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next record to be written to the extract file.

Primary Output Call (CDCUFUNC=CDCUWRTO)

ADACDC calls the user exit before a record is written to the primary output file. This exit point is *not* called during phase 1 processing and has no relevance in this case.

The record area pointer points to the decompressed data record to be written prefixed by the CDCO control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer on return is written to the primary output file. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next record to be written to the output file.

Updating or Adding Records

Consider the following points when updating or adding records from the exit:

- The CDCELEN/CDCERECL fields must correctly reflect the length of data following the CDCEDATA field for the input and write extract exit points.
- The CDCOLEN/CDCORECL fields must correctly reflect the length of data following the CDCODATA field for the input and write extract exit points.
- For the input exit point, records added should have a unique ISN if no ISN update is to be replaced.
- For the input exit point where an ISN is to be replaced, the last occurrence of the ISN should be updated or the replacement record for the ISN should be added after all occurrences for the ISN have been seen.

- When adding records at the extract or primary output exit points, be aware that if file and ISN combinations are duplicated, the duplicated information is written to the primary output file which may confuse processing routines for that file.

9 Examples

```
ADACDC RESETTXF,PHASE=BOTH
```

Ignoring any information on the input transaction file, reads the primary input data and produces primary output using the input data.

Use this syntax and either remove or dummy the input transaction file (CDCTXI) the first time you run the utility to create a valid transaction file for input to subsequent runs.

```
ADACDC PHASE=1
ADACDC FILE=20
ADACDC FILE=40-50
```

Reads the primary input data and processes data only for files 20 and 40 to 50 inclusive. The latest updates for each ISN on those files are written to the extract file.

```
ADACDC PHASE=2
ADACDC FILE=44-47
```

Reads a previously created extract file and writes all records for files 44, 45, 46, and 47 to the primary output file.

10 JCL/JCS Requirements and Examples

- BS2000 52
- z/OS 53
- z/VM 54
- VSE 55

This section describes the job control information required to run ADACDC with BS2000, z/OS, z/VM, and VSE and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log	DDSIIN/ DDSIINnn	tape/ disk	sequential log (not required when PHASE=2)
Extract file	CDCEXT	tape/ disk	output or input extract file (not required when PHASE=BOTH)
Input transaction file	CDCTXI	tape/ disk	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	tape/ disk	not required when PHASE=2
Primary output file	CDCOUT	tape/ disk	not required when PHASE=1
ADARUN parameters	SYSDTA/ DDCARD	disk/ terminal/ reader	*Operations*
ADACDC parameters	SYSDTA/ DDKARTE	disk/ terminal/ reader	*Utilities*
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADACDC messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADACDC JCL Example (BS2000)

```
/.ADACDC LOGON
 /REMA  ADACDC: CAPTURE DELTA CHANGES
 /REMA
 /ASS-SYSOUT EXAMPLE.ADACDC.SYSOUT
 /MODIFY-TEST-OPTION DUMP=YES
 /DELETE-FILE EXAMPLE.OUTPUT.TRANS.FILE
 /SET-JOB-STEP
 /DELETE-FILE EXAMPLE.OUTPUT.PRIMARY.FILE
 /SET-JOB-STEP
 /CREATE-FILE EXAMPLE.OUTPUT.TRANS.FILE,PUB(SPACE=(48,48))
 /CREATE-FILE EXAMPLE.OUTPUT.PRIMARY.FILE,PUB(SPACE=(48,48))
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK  DDASSOR1,EXAMPLE.DByyyyy.ASSOR1
 /SET-FILE-LINK  DDSIIN,EXAMPLE.DByyyyy.PLOG000
 /SET-FILE-LINK  DDSIIN01,EXAMPLE.DByyyyy.PLOG001
 /SET-FILE-LINK  DDSIIN02,EXAMPLE.DByyyyy.PLOG002
 /SET-FILE-LINK  DDSIIN03,EXAMPLE.DByyyyy.PLOG003
 /SET-FILE-LINK  CDCTXI,EXAMPLE.INPUT.TRANS.FILE
 /SET-FILE-LINK  CDCTXO,EXAMPLE.OUTPUT.TRANS.FILE
```

```
     /SET-FILE-LINK  CDCOUT,EXAMPLE.OUTPUT.PRIMARY.FILE
     /SET-FILE-LINK  DDDRUCK,EXAMPLE.ADACDC.DRUCK
     /SET-FILE-LINK  DDPRINT,EXAMPLE.ADACDC.PRINT
     /SET-FILE-LINK  DDLIB,ADABAS.ADAvrs.MOD
     /START-PROGRAM *M(ADABAS.ADAvrs.MOD,ADARUN)
     ADARUN
PROG=ADACDC,MODE=MULTI,IDTNAME=xxxxxxxx,DEVICE=dddd,DBID=yyyyy
     ADACDC
PHASE=BOTH,FILE=1,10,20-30
     /LOGOFF SYS-OUTPUT=DEL
NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log	DDSIIN	tape/ disk	sequential log (not required when PHASE=2)
Input transaction file	CDCTXI	tape/ disk	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	tape/ disk	not required when PHASE=2
Extract file	CDCEXT	tape/ disk	output or input extract file (not required when PHASE=BOTH)
Primary output file	CDCOUT	tape/ disk	not required when PHASE=1
ADARUN parameters	DDCARD	reader	*Operations*
ADACDC parameters	DDKARTE	reader	*Utilities*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACDC messages	DDDRUCK	printer	*Messages and Codes*

ADACDC JCL Example (z/OS)

Refer to ADACDC in the JOBS data set for this example.

```
//ADACDC    JOB
//*
//*    ADACDC: CAPTURE DELTA CHANGES
//*
//CDC      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD          <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1    <=== ASSO
//DDSIIN    DD   DSN=EXAMPLE.DByyyyy.PLOG(-3),DISP=SHR  <== PLOG TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(-2),DISP=SHR  <== PLOG TAPE
```

```
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(-1),DISP=SHR  <== PLOG TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(0),DISP=SHR   <== PLOG TAPE
//CDCTXI    DD   DSN=EXAMPLE.input.trans.file,DISP=SHR
//CDCTXO    DD   DSN=EXAMPLE.output.trans.file,DISP=OLD
//CDCOUT    DD   DSN=EXAMPLE.output.primary.file,DISP=OLD
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADACDC,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACDC PHASE=BOTH,FILE=1,10,20-30
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log (PLOG)	DDSIIN	disk/ tape	sequential log (not required when PHASE=2)
Input transaction file	CDCTXI	disk/ tape	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	disk/ tape	not required when PHASE=2
Extract file	CDCEXT	disk/ tape	output or input extract file (not required when PHASE=BOTH)
Primary output file	CDCOUT	disk/ tape	not required when PHASE=1
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADACDC parameters	DDKARTE	reader	*Utilities*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACDC messages	DDDRUCK	printer	*Messages and Codes*

ADACDC JCL Example (z/VM)

```
/*                                                  */
/*       ADACDC JCL Example (z/VM)                  */
/*                                                  */
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
/*                                                  */
DATADEF DDSIIN,DSN=ADACDC.PLOG,MODE=A
DATADEF CDCTXI,DSN=ADACDC.INNPUT,MODE=A
DATADEF CDCTXO,DSN=ADACDC.OUTPUT,MODE=A
```

```
DATADEF CDCOUT,DSN=ADACDC.PRIMARY,MODE=A
/*                                                  */
DATADEF DDPRINT,DSN=ADACDC,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADACDC.DDDRUCK,MODE=A
/*                                                  */
DATADEF DDCARD,DSN=RUNCDC.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADACDC.CONTROL,MODE=A

EXECOS ADARUN
RCODE = RC
EXIT RCODE
```

Contents of RUNCDC CONTROL A1:

```
ADARUN PROG=ADACDC,DEVICE=dddd,DB=yyyyy
```

Contents of ADACDC CONTROL A1:

```
ADACDC PHASE=BOTH,FILE=1,10,20-30
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	required to read the GCB and FDT entries
Protection log	SIIN	tape disk	SYS010 *	sequential log (not required when PHASE=2)
Input transaction	CDCTXI	tape disk	SYS015 *	not required when RESETTXF is specified or when PHASE=2
Output transaction	CDCTXO	tape disk	SYS016 *	not required when PHASE=2
Extract	CDCEXT	tape disk	SYS017 *	output or input extract file (not required when PHASE=BOTH)
Primary output	CDCOUT	tape disk	SYS018 *	not required when PHASE=1
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations*
ADACDC parameters	-	reader	SYSIPT	*Utilities*
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADACDC messages	-	printer	SYS009	*Messages and Codes*

** Any programmer logical unit may be used.*

ADACDC JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to member ADACDC.X for this example.

```
* $$ JOB JNM=ADACDC,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADACDC
*       CAPTURE DELTA CHANGES
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL SIIN,'EXAMPLE.DByyy.PLOG'
// MTC REW,SYS010
// DLBL CDCTXI,'EXAMPLE.INPUT.TRANS.FILE',,SD
// EXTENT SYS015
// ASSGN SYS015,DISK,VOL=DISK01,SHR
// DLBL CDCTXO,'EXAMPLE.OUTPUT.TRANS.FILE',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK02
// DLBL CDCOUT,'EXAMPLE.OUTPUT.TRANS.FILE',,SD
// EXTENT SYS018,,,,sssss,nnnnn
// ASSGN SYS018,DISK,VOL=DISK04
// EXEC ADARUN,SIZE=(ADARUN,128K)
ADARUN DBID=yyyyy,DEVICE=dddd,PROG=ADACDC,SVC=xxx,MODE=MULTI
/*
ADACDC PHASE=BOTH,FILE=1,10,20-30
/*
/&
* $$ EOJ
```

III ADACMP Utility: Compress-Decompress Data

This chapter covers the following topics:

Functional Overview

Input Data Requirements

Processing (including, among other things, *Segmented Record Considerations*, *Restart Considerations*, and *User Exit 6* explanations)

COMPRESS: Compress an Adabas File

DECOMPRESS: Decompress an Adabas File

Field Definition Statements

JCL/JCS Requirements and Examples

COMPRESS Function Output

DECOMPRESS Function Output

11 Functional Overview

- COMPRESS Function Overview 60
- DECOMPRESS Function Overview 61

This chapter provides an overview of the ADACMP utility.

COMPRESS Function Overview

The COMPRESS function edits and compresses data records that are to be loaded into the database:

Input can be data records from:

- a physical sequential data set (fixed- or variable-length records) supplied by the user; or
- an existing Adabas file (that is, from ADACMP DECOMPRESS or ADAULD UNLOAD).

The logical structure and characteristics of the input data are described with ***field definition statements***:

- The FNDEF statement is used to define a field (or group of fields).
- The SUBFN and SUPFN statements are used to define a subfield and a superfield, respectively.
- The COLDE, HYPDE, PHONDE, SUBDE, and SUPDE statements are used to define a collation descriptor, hyperdescriptor, phonetic descriptor, subdescriptor and superdescriptor, respectively.

The field definitions provided are used to create the Adabas field definition table (FDT) for the file. Alternatively, you can use an existing Adabas FDT instead of providing field definitions (read about the **FDT parameter**).

If the fields in the input record are to be processed in an order that is different from their position in the input record, or if one or more fields are to be skipped, the **FORMAT parameter** may be used to indicate the order and location of the input fields.

The ADACMP COMPRESS function processes the input data as follows:

- Checks numeric data for validity.
- Removes trailing blanks from alphanumeric fields.
- Removes leading zeros from numeric fields.
- Packs numeric unpacked fields.

Fields defined with the **fixed (FI) option** are not compressed.

A user exit can be used to further edit the input data. For more information, read ***User Exit 6***, elsewhere in this section.

The output of the ADACMP COMPRESS function that is used as input to the ADALOD utility includes the FDT, compressed records, and on the utility report, the Data Storage space requirement (for the ADALOC DSSIZE parameter) and the Temp and Sort data set size estimates (TEMPSIZE and SORTSIZE).

The ADACMP COMPRESS function report also indicates:

- the number of records processed;
- the number of records rejected; and
- the compression rate percentage.

A data set containing rejected records is also produced.

DECOMPRESS Function Overview

The DECOMPRESS function decompresses individual files:

Input data can be decompressed from data records in existing Adabas files:

- unloaded using the ADAULD (file unload) utility; or
- directly (without separate file unloading).

The INFILE parameter of ADACMP DECOMPRESS is used for Adabas files that are directly decompressed. As part of the decompression process, the target file is unloaded without FDT information, which can save time when decompressing larger files.

The output of the ADACMP DECOMPRESS function includes ISNs if the ISN parameter is specified. The DECOMPRESS output may be used as input to a non-Adabas program or as input to the COMPRESS function, once any desired changes to the data structure or field definitions for the file are completed.

12 Input Data Requirements

- Input Data Structure 64
- Multiple-Value Field Count 64
- Periodic Group Count 66
- Variable-Length Field Size 69

This chapter describes the input data requirements of the ADACMP utility.

Input Data Structure

ADACMP input data must be in a sequential data set or file. Indexed sequential and VSAM input cannot be used.

The records may be fixed, variable, or of undefined length. The maximum input record length permitted depends on the operating system. The maximum compressed record length is restricted by the Data Storage block size in use and the maximum compressed record length set for the file (see the **MAXRECL parameter** of the ADALOD utility). The input records can be in either blocked or unblocked format.

The fields in each record must be structured according to the field definition statements provided (or the specified FDT if an existing Adabas FDT is being used). If a user exit routine is used, the structure following user exit processing must agree with the field definitions. Any trailing information contained in an input record for which a corresponding field definition statement is not present is ignored and is not included in the ADACMP output.

Data sets that contain no records are also permitted.

The input data set can be omitted if the parameter NUMREC=0 is supplied.

Multiple-Value Field Count

The number of values in each record's multiple-value field must either be specified in the field definition statement, or the value count must precede the values in each input record. When specified in the field definition statement, the minimum multiple-value count is 1, and the maximum is 65,534 or 191, depending on the setting of the **MUPEX** and **MUPECOUNT** parameters of ADACMP COMPRESS. When the minimum count is specified in the input record, zero (0) can be specified to indicate a multiple-value field containing no values.

If the number of values is the same for each record, this number may be specified with the field definition statement for the multiple-value field (in the **occurrences** specification). In this case, the count byte in the input record must be omitted. If the record definitions are from an existing FDT or if the input data is from an earlier DECOMPRESS operation, the data already contains the length values; the count must not be specified in the **field definition statements**.

The count you specify may be changed by ADACMP if the **NU** option is specified for the field. ADACMP suppresses null values, and changes the count field accordingly. This is true whether you specify the value count before each series of values, or in the field definition statement. Refer to the section ***MU: Multiple-Value Field***.

Example 1: Multiple-Value Field Count with Varying Number of Occurrences

Field Definition:

```
ADACMP FNDEF='01,MF,5,A,MU,NU'
```

Each record contains a different number of values for MF, and the count comes before each series of occurrences.

	Before ADACMP	After ADACMP
Input Record 1 (3 values)	MF count=3 AAAA BBBB CCCC	MF count=3 AAAA BBBB CCCC
Input Record 2 (2 values)	MF count=2 AAAA BBBB	MF count=2 AAAA BBBB
Input Record 3 (3 values)	MF count=3 AAAA bbbb CCCC	MF count=2 AAAA CCCC
Input Record 4 (no values)	MF count=0	MF count=0
Input Record 5 (1 value)	MF count=1 bbbb	MF count=0

Example 2: Multiple-Value Field Count with Same Number of Occurrences

Field Definition:

```
ADACMP FNDEF='01,MF,4,A,MU(3),NU'
```

Each record contains 3 values for MF, as specified in the field definition statement.

	Before ADACMP	After ADACMP
Input Record 1	AAAA BBBB CCCC	MF count=3 AAAA BBBB CCCC
Input Record 2	AAAA BBBB bbbb	MF count=2 AAAA BBBB
Input Record 3	AAAA bbbb CCCC	MF count=2 AAAA CCCC

	Before ADACMP	After ADACMP
Input Record 4	bbbb bbbb bbbb	MF count=0

Periodic Group Count

Each periodic group must specify a count of field iterations (occurrences) in the record. The count is specified either within the field definition statement for all records, or as a one- or two-byte binary value (depending on the value of the **MUPECOUNT** parameter) before each occurrence group in every record. If the count is in the field definition statement, the count byte must be omitted from the input records. When specified in the field definition statement, the minimum count allowed is 1, and the maximum is 65,534 or 191, depending on the setting of the **MUPEX** and **MUPECOUNT** parameters of ADACMP COMPRESS.. When the minimum count is specified in the record, the value can be zero (0) for a periodic group with no occurrences.

The occurrence count provided may be modified by ADACMP if all the fields contained in the periodic group are defined with the **NU** option. If all the fields within a given occurrence contain null values and there are no following occurrences that contain non-null values, the occurrence will be suppressed and the periodic group occurrence count will be adjusted accordingly.

Example 1: Periodic Group Count with Varying Number of Occurrences

Field Definitions:

```
ADACMP COMPRESS
ADACMP FNDEF='01,GA,PE'
ADACMP FNDEF='02,A1,4,A,NU'
ADACMP FNDEF='02,A2,4,A,NU'
```

The input records contain a variable number of occurrences for GA (up to 191 occurrences are permitted as the **MUPEX** parameter is not specified causing the **MUPECOUNT** parameter to default to "1"). The count of occurrences comes before each occurrence group in the input records.

	Before ADACMP	After ADACMP
Input Record 1	GA count=2	GA count=2
	GA (1st occurrence) A1=AAAA A2=BBBB	A1=AAAA A2=BBBB
	GA (2nd occurrence) A1=CCCC A2=DDDD	A1=CCCC A2=DDDD
Input Record 2	GA count=1	GA count=0

	Before ADACMP	After ADACMP
	GA (1st occurrence) A1=bbbb A2=bbbb	suppressed suppressed
Input Record 3	GA count=3	GA count=3
	GA (1st occurrence) A1=AAAA A2=bbbb	A1=AAAA A2=suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	A1=BBBB A2=suppressed
	GA (3rd occurrence) A1=CCCC A2=bbbb	A1=CCCC A2=suppressed
Input Record 4	GA count=0	GA count=0

Example 2: Periodic Group Count with Same Number of Occurrences

Field Definitions:

```
ADACMP FNDEF='01,GA,PE(3)'
ADACMP FNDEF='02,A1,4,A,NU'
ADACMP FNDEF='02,A2,4,A,NU'
```

All input records contain 3 occurrences for GA, as specified in the field definition statement.

	Before ADACMP	After ADACMP
Input Record 1		GA count=3
	GA (1st occurrence) A1=AAAA A2=bbbb	A1=AAAA A2 suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	A1=BBBB A2 suppressed
	GA (3rd occurrence) A1=CCCC A2=bbbb	A1=CCCC A2 suppressed
Input Record 2		GA count=2 (see note)
	GA (1st occurrence) A1=bbbb A2=bbbb	A1=suppressed A2=suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	A1=BBBB A2=suppressed

	Before ADACMP	After ADACMP
	GA (3rd occurrence) A1=bbbb A2=bbbb	A1=suppressed A2=suppressed
Input Record 3	All occurrences contain null values	GA count=0 All occurrences are suppressed

Note: The first occurrence is included in the count since occurrences follow that contain non-null values. The third occurrence is not included in the count since there are no non-null values in the occurrences that follow.

Example 3: Adding a Field to a PE-Group

In the PE named AW, the field AY should be added:

Old FDT	New FDT
01 AA,8,A,DE,UQ	01 AA,8,A,DE,UQ
01 AW,PE	01 AW,PE
02 AX,8,U,NU	02 AX,8,U,NU
02 AT,8,U,NU	02 AT,8,U,NU
01,AZ,3,A,DE,MU,NU	02 AY,8,U,NU
	01,AZ,3,A,DE,MU,NU

Note: All of the currently existing fields in the PE must be specified.

1. Determine the maximum occurrence of the PE (for example, a result of 2).
2. Decompress the file with the format parameter.
3. Decompress INFILE=xx,FORMAT='AA,AX1-2,AT1-2,AZ'
4. Compress again:

```
ADACMP COMPRESS FILE=32
ADACMP FORMAT='AA,AX1-2,AT1-2,AZ'
ADACMP FNDEF='01,AA,8,A,DE,UQ'
ADACMP FNDEF='01,AW,PE(2)'
ADACMP FNDEF='02,AX,8,U,NU'
ADACMP FNDEF='02,AT,8,U,NU'
ADACMP FNDEF='02,AY,8,U,NU'
ADACMP FNDEF='01,AZ,3,A,DE,MU,NU'
```

Variable-Length Field Size

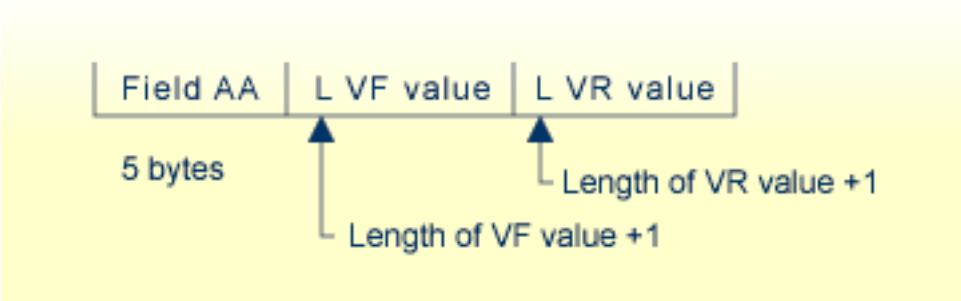
Each value of a variable-length field (length parameter not specified in the field definition) must be preceded by a one-byte binary count indicating the value length (including the length byte itself). An LA field specified with variable length (i.e., length 0), must be preceded by a two-byte inclusive length. An LB field specified with variable length must be preceded by a four-byte inclusive length.

Example of Variable-Length Field Size

Field Definitions:

```
ADACMP FNDEF='01,AA,5,A,DE'
ADACMP FNDEF='01,VF,0,A'
ADACMP FNDEF='01,VR,0,A'
```

Input record:



13 Processing

- Segmented Record Considerations 72
- Data Verification 78
- Data Compression 78
- Representation of LOB Values and Value References in Uncompressed Data 80
- Identifying MU and PE Occurrences Greater Than 191 in Compressed Records 81
- Restart Considerations 81
- User Exit 6 81

This chapter describes ADACMP utility processing.

Segmented Record Considerations

If a decompressed record from ADACMP is too long to fit into the longest record length allowed for a sequential data set (32KB or less), ADACMP can segment it into multiple physical records. A single logical decompressed record can span one or more physical decompressed records.

In addition, the ADACMP utility allows you to create special headers, ADAH and ADAC, in the decompressed output. These special headers are used only with ADACMP processing. They identify the position of the payload data in the logical record as well as the relation between the physical record and the other physical records in the same logical record. When created, an ADAH header is used for the first physical record of a logical record; ADAC headers are used for the second and subsequent physical records that comprise the logical record. If a decompressed logical record does not need to be segmented (if it fits into one physical record), only an ADAH header is created; there is no need for ADAC headers.

Do not confuse the record segmentation that occurs with ADACMP decompression logic and *record spanning*. Spanned records also consist of multiple physical records (one primary record and multiple secondary records), but they are compressed records. In addition, each spanned record is automatically assigned a standard spanned record header that is not the same as the ADAH and ADAC headers you can create for decompressed records using ADACMP; segmented records produced by ADACMP do not contain the standard spanned record header. For complete information about spanned records, read *Spanned Records* in *Adabas Concepts and Facilities Manual*.

This section covers the following topics:

- Creating and Supporting ADACMP Headers
- ADAH and ADAC Header Descriptions

Creating and Supporting ADACMP Headers

The HEADER parameter of ADACMP DECOMPRESS controls whether the decompression logic produces the headers in its *output*. The HEADER parameter of ADACMP COMPRESS controls whether the compression logic will accept the ADACMP headers as part of the uncompressed *input*.

ADAH and ADAC Header Descriptions

This section covers the following topics:

- ADAH Headers
- ADAC Headers
- Example

ADAH Headers

When ADACMP headers are used, the first physical record of a logical record begins with an ADAH header containing the following information:

- The characters "ADAH".
- The length of the ADAH header.
- A continuation indicator that indicates whether this is the last physical record in the logical record or whether another physical record for the same logical record will follow this one.
- The total length of the record (with the headers). The value may be zero if the total record length is not known when the first physical record is written.
- The length of the payload data (a segment of the logical record) in this physical record. This refers to the length of the payload data; it does not include the length of the ADAH header. The length must be less than or equal to the the lenght of the physical record minus the length of the header. If it is less than this value, any extra data in the physical record (not covered by the payload data length) is ignored.

The payload data follows the ADAH header.

The ADAH DSECT can be found in the ADAH member of the distributed Adabas 8 SRCE library.

ADAC Headers

When ADACMP decompressed record segmenting occurs and when ADACMP headers are requested, the second and every subsequent physical record for a logical record begins with an ADAC header containing the following information:

- The characters "ADAC".
- The length of the ADAC header.
- A continuation indicator that indicates whether this is the last physical record in the logical record or whether another physical record for the same logical record will follow this one.
- The sequence number of this secondary record in the logical record. The second physical record of a logical record is the first secondary record and therefore has a sequence number of "1". The sequence numbers are in ascending order, without gaps.

- The offset within the logical record of the payload data (segment) contained in this physical record. This offset is the sum of the payload data lengths of each prior physical record in the logical record.
- The length of the payload data (segment) in this physical record. This refers to the length of the payload data; it does not include the length of the ADAC header. The length must be less than or equal to the the lenght of the physical record minus the length of the header. If it is less than this value, any extra data in the physical record (not covered by the payload data length) is ignored.

The payload data follows the ADAC header.

The ADAC DSECT can be found in the ADAC member of the distributed Adabas 8 SRCE library.

Example

The following table depicts three logical records spanning seven physical records of uncompressed data.

Note: DSECTs for the ADAH and ADAC headers can be found in members ADAH and ADAC in the distributed Adabas 8 SRCE library.

Logical Record	Physical Record Headers	Header Fields		Description
		Field	Value	
1	ADAH	ADAHEYE	ADAH	ADAH header eyecatcher
		ADAHLEN	32	ADAH header length
		ADAHIND	C	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Must contain binary zeros.
		ADAHTLEN	50000	Total length of logical record. This value may be zero if the total length is not known when the first segment is written.
		Reserved	0	Reserved
		ADAHSLEN	27962	Length of this segment (length of the payload data). The sum of the values of ADAHLEN and ADAHSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADAHDATA	'Record 1 - payload data part 1'	Payload data
	ADAC	ADACEYE	ADAC	ADAC header eyecatcher

Logical Record	Physical Record Headers	Header Fields		Description
		Field	Value	
		ADACLEN	32	ADAC header length
		ADACIND	E	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADACSEQ	1	Continuation record sequence number within the logical record (the first ADAC record has a sequence number of "1").
		ADACOFFS	27962	Segment offset within the logical record (the first payload data byte is at offset "0").
		Reserved	0	Reserved
		ADACSLEN	22038	Length of this segment (length of the payload data). The sum of the values of ADACLEN and ADACSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADACDATA	'Record 1 - payload data part 2'	Continuation record payload data
2	ADAH	ADAHEYE	ADAH	ADAH header eyecatcher
		ADAHLEN	32	ADAH header length
		ADAHIND	E	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Must contain binary zeros.
		ADAHTLEN	25000	Total length of logical record. This value may be zero if the total length is not known when the first segment is written.
		Reserved	0	Reserved
		ADAHSLEN	25000	Length of this segment (length of the payload data). The sum of the values of ADAHLEN and ADAHSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADAHDATA	'Record 2 - payload data'	Payload data
3	ADAH	ADAHEYE	ADAH	ADAH header eyecatcher
		ADAHLEN	32	ADAH header length

Logical Record	Physical Record Headers	Header Fields		Description
		Field	Value	
		ADAHIND	C	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Must contain binary zeros.
		ADAHTLEN	0	Total length of logical record. This value may be zero if the total length is not known when the first segment is written (as is the case for logical record 3).
		Reserved	0	Reserved
		ADAHSLEN	27962	Length of this segment (length of the payload data). The sum of the values of ADAHLEN and ADAHSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADAHDATA	'Record 3 - payload data part 1'	Payload data
	ADAC	ADACEYE	ADAC	ADAC header eyecatcher
		ADACLEN	32	ADAC header length
		ADACIND	C	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADACSEQ	1	Continuation record sequence number within the logical record (the first ADAC record has a sequence number of "1").
		ADACOFFS	27962	Segment offset within the logical record (the first payload data byte is at offset "0").
		Reserved	0	Reserved
		ADACSLEN	27962	Length of this segment (length of the payload data). The sum of the values of ADACLEN and ADACSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADACDATA	'Record 3 - payload data part 2'	Continuation record payload data
	ADAC	ADACEYE	ADAC	ADAC header eyecatcher
		ADACLEN	32	ADAC header length

Logical Record	Physical Record Headers	Header Fields		Description
		Field	Value	
		ADACIND	C	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADACSEQ	2	Continuation record sequence number within the logical record (the first ADAC record has a sequence number of "1").
		ADACOFFS	55924	Segment offset within the logical record (the first payload data byte is at offset "0").
		Reserved	0	Reserved
		ADACSLEN	27962	Length of this segment (length of the payload data). The sum of the values of ADACLEN and ADACSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADACDATA	'Record 3 - payload data part 3'	Continuation record payload data
	ADAC	ADACEYE	ADAC	ADAC header eyecatcher
		ADACLEN	32	ADAC header length
		ADACIND	E	Continuation indicator. Valid values are: C: Continuation record segment to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADACSEQ	3	Continuation record sequence number within the logical record (the first ADAC record has a sequence number of "1").
		ADACOFFS	83886	Segment offset within the logical record (the first payload data byte is at offset "0").
		Reserved	0	Reserved
		ADACSLEN	16114	Length of this segment (length of the payload data). The sum of the values of ADACLEN and ADACSLEN is the minimum length of the physical record. The physical record can be longer than this; in this case, the excess data has no meaning and is ignored.
		ADACDATA	'Record 3 - payload data part 4'	Continuation record payload data

Data Verification

ADACMP checks each field defined with format P (packed) or U (unpacked) to ensure that the field value is numeric and in the correct format. If a value is empty, the null characters must correspond to the format specified for the field (see *Representing SQL Null Values* in the *Field Definition Statements* section.

Alphanumeric (A)	blanks (hex '40')
Binary (B)	binary zeros (hex '00')
Fixed (F)	binary zeros (hex '00')
Floating Point (G)	binary zeros (hex '00')
Packed (P)	decimal packed zeros with sign (hex '00' followed by '0F', '0C', or '0D' in the rightmost, low-order byte)
Unpacked (U)	decimal unpacked zeros with sign (hex 'F0' followed by 'C0' or 'D0' in the rightmost, low-order byte)

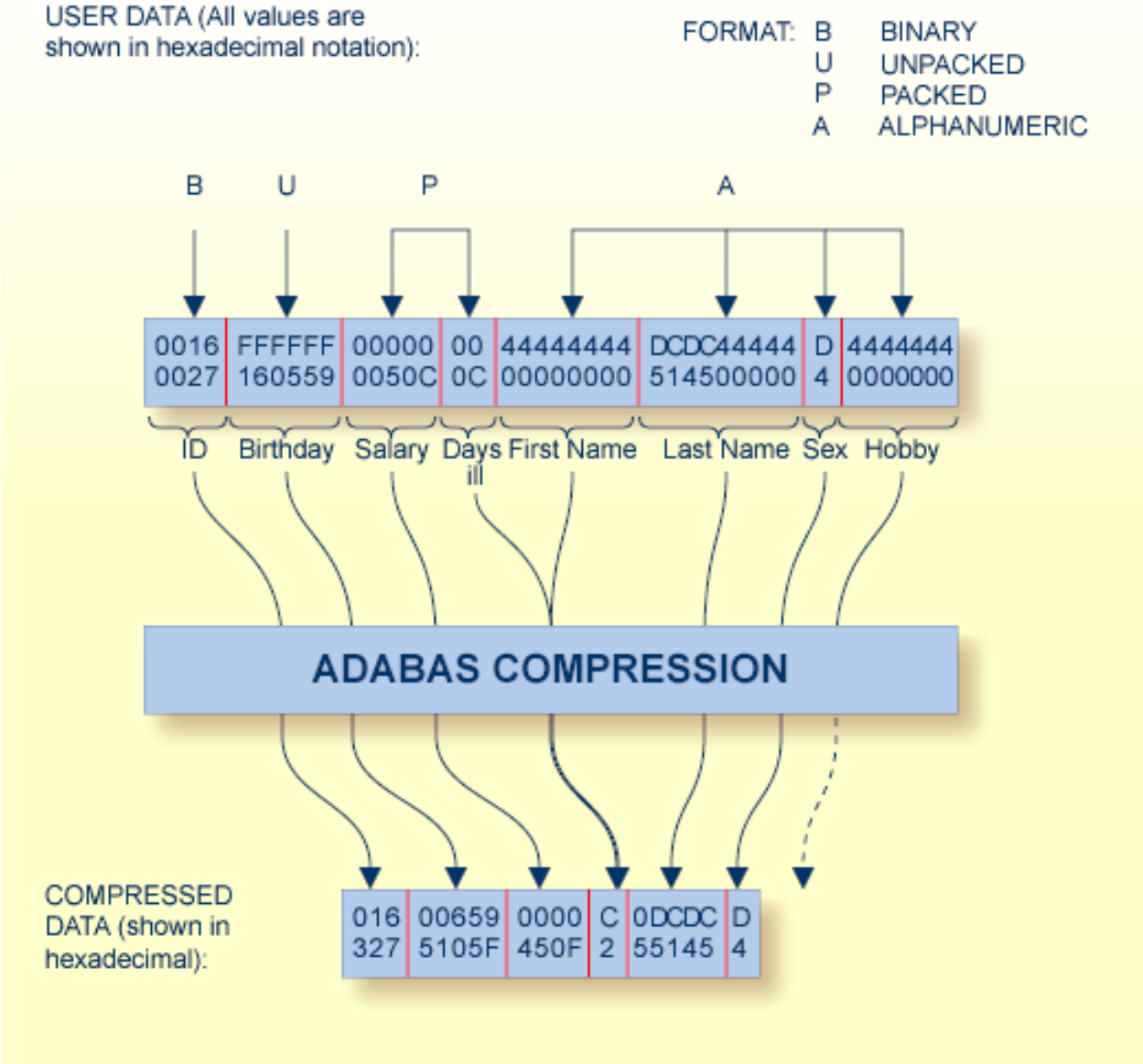
Any record that contains invalid data is written to the ADACMP error (DDFEHL) data set and is not written to the compressed data set.

Data Compression

The value for each field is compressed (unless the **FI option** is specified) as follows:

- Trailing blanks are removed for fields defined with A format.
- Leading zeros are removed for numeric fields (fields defined with B, F, P or U format).
- If the field is defined with U (unpacked) format, the value is converted to packed (P) format.
- Trailing zeros in floating-point (G format) fields are removed.
- If the field is defined with the NU option and the value is a null value, a one-byte indicator is stored. Hexadecimal 'C1' indicates one empty field follows, 'C2' indicates that two empty fields follow, and so on, up to a maximum of 63 before the indicator byte is repeated. For SQL null value (NC option field) compression, see *Representing SQL Null Values* in the *Field Definition Statements* section.
- Empty fields located at the end of the record are not stored, and therefore not compressed.

Example of Data Compression



ADACMP Compression

The graphic shows how the following field definitions and corresponding values would be processed by ADACMP:

```
FNDEF='01,ID,4,B,DE'
FNDEF='01,BD,6,U,DE,NU'
FNDEF='01,SA,5,P'
FNDEF='01,DI,2,P,NU'
FNDEF='01,FN,9,A,NU'
FNDEF='01,LN,10,A,NU'
```

```
FNDEF='01,SE,1,A,FI'
FNDEF='01,HO,7,A,NU'
```

Representation of LOB Values and Value References in Uncompressed Data

This section describes how large object (LB) field values, LB field value references, and logical records that are longer than 32 KB must be represented in the input data set for the ADACMP COMPRESS function and how these items are represented in the output data set of the ADACMP DECOMPRESS function.

- Large Object (LB) Field Values
- Large Object (LB) Field Value References

Large Object (LB) Field Values

If ADACMP is run without the FORMAT parameter, each large object (LB) field value in the uncompressed data is preceded by a 4-byte length field. The length value includes the length of the LB field value proper plus four bytes for the length field itself. An empty LB field value for a field defined *without* the NB option consists of the length field with a value of 5 and a single blank; for a field defined *with* the NB option, an empty LB field value consists only of the length field with a value of 4.

If ADACMP COMPRESS is used to define an FDT with LB fields, each LB field value in the uncompressed input must be less than or equal to 253 bytes.

Large Object (LB) Field Value References

When the ADACMP DECOMPRESS function is run with LOBVALUES=NO to decompress only the records from the *base file* of a *LOB file group*, omitting all LB field values stored in the associated *LOB file*, each reference in a base file record to a LB field value in the LOB file is represented in the uncompressed output as follows:

- The four-byte length field for the LB field value contains X'FFFFFFFF' (high value) to indicate the presence of the reference to an LB field value.
- The indicator is followed by a two-byte inclusive length field for the LB field value reference. The length value includes the length of the LB field value reference proper plus two bytes for the length field itself.
- The length field is followed by the LB field value reference proper.

The same structure is expected by the ADACMP COMPRESS function with LOBVALUES=NO in the place of an LB field value that is stored in the *LOB file* associated with the *base file* that is being compressed.

LB field value references that are input to ADACMP COMPRESS must originate from ADACMP DECOMPRESS. There is no sensible way to introduce new LB field value references using COMPRESS, as they would not properly refer to existing LB field values in a *LOB file*.

Identifying MU and PE Occurrences Greater Than 191 in Compressed Records

MU and PE occurrences greater than 191 are indicated in compressed records by a x'C0' byte at the beginning of the occurrence count. This byte is set by the ADACMP utility or the nucleus when the records are compressed. The x'C0' indicator byte is followed by a byte indicating the number of count bytes used for the MU or PE occurrence count that follows. For example, consider the following indicator:

```
X'C0020204'
```

In this example, x'C0' indicates this is an extended count; x'02' indicates that there are two count bytes, and x'0204' indicates that there are 516 occurrences of the field.

Restart Considerations

ADACMP has no restart capability. An interrupted ADACMP execution must be reexecuted from the beginning.

User Exit 6

A user-written routine called user exit 6 can be used for editing during ADACMP COMPRESS processing. The routine may be written in Assembler or COBOL. It must be assembled or compiled and then linked into the Adabas load library (or any library concatenated with it).

User exit 6 is invoked by specifying:

```
ADARUN UEX6=program
```

where *program* is the routine name in the load library.

For specific information about the user exit 6 structure and parameters, read *User Exits and Hyperexits* in *Adabas User, Hyperdescriptor, and Collation Descriptor Exits Manual*.

14 COMPRESS: Compress an Adabas File

- Essential Parameters and Subparameters 84
- Optional Parameters and Subparameters 85
- ADACMP COMPRESS Examples 92

```
ADACMP  COMPRESS  {field-definition-statements | FDT = file-number}
                  [CODE = cipher-code]
                  [DATADEVICE = { device-type | ADARUN-device }]
                  [DEVICE = { device-type-list | ADARUN-device }]
                  [FACODE = file-alpha-EBCDIC-key]
                  [FILE = {file-number | 0 }]
                  [FWCODE = file-wide-key]
                  [FUWCODE = {wide-key | UWCODE-definition }]
                  [FORMAT = format]
                  [HEADER = { YES | NO }]
                  [LOBDEVICE = { device-type-list | DEVICE-parmvalue }]
                  [LOBVALUES = { YES | NO }]
                  [MAXLOGRECLEN = buffer-size]
                  [MUPECOUNT = 1 | 2]
                  [MUPEX]
                  [NOUSERABEND]
                  [NUMREC = number-of-records]
                  [PASSWORD = "password"]
                  [RECFM = { F | FB | V | VB | U }]
                        [,LRECL = record-length }]
                  [SPAN]
                  [ { USERISN | MINISN = { start-isn | 1 } } ]
                  [UACODE = userdata-alpha-key]
                  [UWCODE = {userdata-wide-key | FWCODE-definition }]
                  [UARC = {userdata-architecture-key | 2 }]
```

This chapter describes the syntax and parameters of the ADACMP COMPRESS function.

Essential Parameters and Subparameters

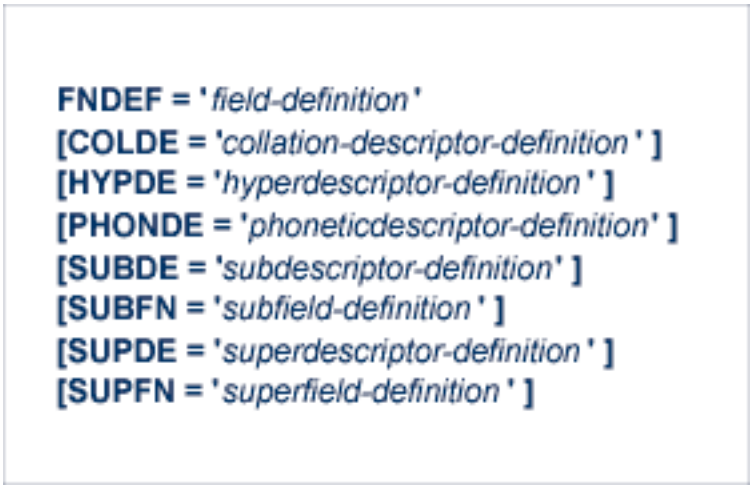
field-definition-statements

Field definition statements, when provided as input to ADACMP, are used to:

- provide the length and format of each field contained in the input record. This enables ADACMP to determine the correct field length and format during editing and compression.
- create the Field Definition Table (FDT) for the file. This table is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

Either an FDT parameter or field definition statements must be supplied for ADACMP COMPRESS. If both are supplied, the field definition statements are ignored.

The field definition statements that can be included in this syntax:



For complete information on field definition statements, including their syntax, read ***Field Definition Statements***, elsewhere in this section.

FDT: Use Existing Adabas Field Definition Table

Specifies an existing Adabas FDT to be used. The FDT may be that of an existing file or a file that has been deleted with the KEEPFDT option of the ADADBS utility.

Either an FDT parameter or field definition statements must be supplied for ADACMP COMPRESS. If both are supplied, the field definition statements are ignored.

If the FDT parameter is specified, the input data must be consistent with the structure as defined in the specified FDT, unless the FORMAT parameter is used. When the FDT defines multiple-value fields or periodic groups, length values must be defined or already included in the FDT. Read sections ***Multiple-Value Field Count*** and ***Periodic Group Field Count***.

If the FDT parameter is used, any field definitions specified will be ignored.

Optional Parameters and Subparameters

CODE: Cipher Code

If the data is to be loaded into the database in ciphered form, the cipher code must be specified with this parameter. See the *Adabas Security* documentation for additional information on the use of ciphering.

Note: You cannot specify the CODE parameter in a ADACMP COMPRESS function if the file contains LB fields.

DATADEVICE: Device Type

The DATADEVICE parameter specifies the Data Storage device type to be used for the segmentation of spanned records. If the SPAN parameter is specified, ADACMP will break long, spanned, compressed records into segments that are just a bit smaller than the Data Storage block size implied by the DATADEVICE parameter.

If the SPAN parameter is not specified, no value for DATADEVICE is required. However, it can be specified to limit the size of compressed records. In this case, all records that exceed the given storage device block size will be written to the DDFEHL error data set.

DEVICE: Device Type

If the DEVICE parameter is specified, ADACMP calculates and displays a report of this run's space requirements for each specified device type. This report includes an indication of whether or not the `MUPEX` parameter has been set for a file.

If no device types are listed on the DEVICE parameter, the ADARUN device type is used as the default.

FACODE: Alphanumeric Field Encoding

FACODE must be specified if you want to define UES file encoding for alphanumeric fields in the file. The alphanumeric encoding must belong to the EBCDIC encoding family; that is, the space character is X'40'.

FILE: File Number

If the FDT contains a hyperdescriptor, this parameter must be specified. The specified file number becomes input for the related hyperexit. For more information about hyperexits, refer to the *Adabas DBA Reference* documentation.

User exit 6 is always supplied with this file number. If FILE is not specified, a value of zero is assumed.

FORMAT: Input Record Format Definition

Use this parameter to provide a format definition that indicates the location, format, and length of fields in the input record. The format provided must follow the rules for format buffer entries for update commands as described in the *Adabas Command Reference Guide* documentation.

Conversion rules are those described for Adabas update commands in the *Adabas Command Reference Guide* documentation. For conversion of SQL null (NC option) field values, see ***NC: SQL Null Value Option***. If a field is omitted in the FORMAT parameter, that field is assigned no value.

If the FORMAT parameter is omitted, the input record is processed in the order of the field definition statements provided or, if the FDT parameter is used, according to an existing Adabas field definition table.

If the LOBVALUES parameter is set to NO, LB fields cannot be used in the definition supplied in the FORMAT parameter.

FUWCODE: Wide-Character Field Default User Encoding

FUWCODE defines the default user encoding for wide-character fields for the file when loaded in the database. If this parameter is omitted, the encoding is taken from the UWCODE definition of the database.

FWCODE: Wide-Character Field Encoding

If fields with format W (wide-character) exist in the compressed file, you *must* specify FWCODE to define the file encoding for them.

FWCODE also determines the maximum byte length of the wide-character field.

HEADER

This optional parameter indicates whether or not the ADACMP compression logic should expect segmented ADACMP record headers in the uncompressed input records. Valid values are YES or NO; the default is NO.

HEADER=NO is the format accepted and produced by ADACMP in Adabas versions prior to Adabas 8. When it is specified, the input records must contain only the uncompressed data for the fields of the file being processed. Each data record must fit into one physical record of the sequential input data set (less than 32 KB). If the data exceeds this size, the records in error are written to the DDFEHL error data set.

HEADER=YES can only be specified if you are running Adabas 8. If HEADER=YES is specified, each input record must begin with either an ADAH or ADAC header, relating the physical segmented record with a logical record to be processed by ADACMP. Each logical record can be larger than 32 KB. The header in each physical record defines the position of the data following it within the logical record. DSECTs for the ADAH and ADAC headers can be found in members ADAH and ADAC of the distributed Adabas SRCE data set.

If HEADER=YES is specified, an error may occur while segmented uncompressed records are being assembled into a logical record. If the ADAH header is in error, the ADAH record is written and subsequent ADAC records are not written until the next ADAH record is processed. If an ADAC header is in error, the preceding ADAH header will be written without its payload data. The ADAC record in error will be written in its entirety. Subsequent ADAC records are not written until the next ADAH record is processed. For more information about rejected records and possible response codes resulting from them, read ***COMPRESS Function Output***, elsewhere in this section.

Do not confuse the HEADER parameter with the SPAN parameter. The SPAN parameter controls whether the compressed records themselves should be spanned if they exceed the Data Storage block size of the device. Spanned records contain a standard spanned record header that differs from the ADAH or ADAC headers expected by the HEADER parameter. For complete information about spanned records, read *Spanned Records* in *Adabas Concepts and Facilities Manual*.

LOBDEVICE: Device Type for LOB File

This optional parameter specifies the data storage device type that will be used for loading the *LOB file* produced by the ADACMP COMPRESS function. ADACMP will divide the LB

field values into segments based on the block size of the specified device. This parameter is only valid if the FDT includes one or more large object (LB option) fields.

If LOBDEVICE is not specified, the device specified for the ADACMP COMPRESS DEVICE parameter is used. If no device is specified for the DEVICE parameter either, the value of the ADARUN DEVICE parameter is used.

LOBVALUES: LB Field Size Indicator

This optional parameter indicates whether long LB field values (larger than 253 bytes) or short LB field values (up to 253 bytes) are expected in the ADACMP COMPRESS input data. Valid values for this parameter are "YES" and "NO"; the default is "NO".

If "YES" is specified for this parameter, the uncompressed input data may contain LB field values larger than 253 bytes. In this case, a second sequential output data set must also be supplied in the JCL for the run. This second data set is identified in the JCL using the DD control statement DDAUSB1. It is used to store the compressed LB segment records for LB field values that are larger than 253 bytes.

If "NO" is specified for this parameter, the uncompressed input data may contain only LB fields up to 253 bytes long and references to LB field values stored in a *LOB file*. In this case, you cannot specify an LB field in the ADACMP COMPRESS FORMAT parameter. During processing, ADACMP writes any short LB field values and LB field value references contained in the input to the output.

Note: An ADACMP DECOMPRESS function with LOBVALUES=NO followed by an ADACMP COMPRESS function with LOBVALUES=NO can be used to modify the FDT of the *base file* in the *LOB file group*.

LRECL: Input Record Length (VSE Only)

If RECFM=F or RECFM=FB is specified, this parameter must also be specified to provide the record length (in bytes) of the input data; otherwise, do not specify LRECL.

For z/OS, the record length is taken from the input data set label or DD statement.

For BS2000, the record length is taken from the catalog entry or /FILE statement.

MAXLOGRECLEN: Buffer Size

This optional parameter can be used to specify the size, in bytes, of a buffer used by ADACMP to assemble any segmented, uncompressed, physical records into a compressed logical record. This buffer is allocated only if HEADER=YES is also specified. Otherwise, the setting of MAXLOGRECLEN is ignored. The default value of MAXLOGRECLEN is 1,048,576 bytes (1 MB).

If the value specified by MAXLOGRECLEN is appended by the letter "K", it is multiplied by 1024. The minimum value is 32768 bytes.

Note: MAXLOGRECLEN pertains to the spanning of uncompressed input data and should not be confused with the SPAN parameter which pertains to the spanning of compressed records.

MINISN: Starting ISN

For automatic ISN assignment, MINISN defines the lowest ISN to be used. If MINISN is not specified, the default is 1. If USERISN is specified, MINISN cannot be specified.

MUPECOUNT: Specify Value Count Field Size

The `MUPECOUNT` parameter specifies the size of the value count field in the input record for the COMPRESS function . Its syntax is:

```
MUPECOUNT={1 | 2}
```

If "1" is specified, each value count field preceding the MU or PE values in the input data must be one byte with a value of no more than "191". If "2" is specified, each value count field preceding the MU or PE values in the input data must be two bytes. A value count may exceed 191 only if the `MUPEX` parameter is also specified. When `MUPEX` is specified, the maximum count is "65,534".

If the `MUPEX` parameter has been set, the default for `MUPECOUNT` is "2"; if the `MUPEX` parameter has not been set, the default for `MUPECOUNT` is "1".

Note: This option is not compatible with releases prior to Adabas 8; therefore, backward conversion to prior versions is not possible once records with more than 191 PE group occurrences have been loaded. However, ADACMP data sets created by versions of Adabas prior to Adabas 8 will load successfully using the Version 8 ADALOD utility.

For information on how to identify MU and PE occurrences greater than 191 in the compressed record, read ***Identifying MU and PE Occurrences Greater Than 191 in Compressed Records***, elsewhere in this section.

MUPEX: Enable Extended Periodic Group Count

The `MUPEX` parameter indicates whether extended MU/PE limits (greater than 191) are allowed for the file. If this option is *not* specified, the maximum number of MU fields and the maximum number of PE fields that can be specified is 191. Otherwise, the maximum is 65,534.

If you set the `MUPEX` parameter, consider setting the `SPAN` parameter as well to avoid compression errors if the compressed record size is exceeded when compressing the additional MU and PE fields.

Note: This option is not compatible with releases prior to Adabas 8; therefore, backward conversion to prior versions is not possible once records with more than 191 PE group occurrences have been loaded. However, ADACMP data sets created by versions of Adabas prior to Adabas 8 will load successfully using the Version 8 ADALOD utility.

For information on how to identify MU and PE occurrences greater than 191 in the compressed record, read *Identifying MU and PE Occurrences Greater Than 191 in Compressed Records*, elsewhere in this section.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Number of Records to Be Processed

Specifies the number of input records to be processed. If this parameter is omitted, all input records contained on the input data set are processed.

Software AG recommends using this parameter for the initial ADACMP execution if a large number of records are contained on the input data set. This avoids unneeded processing of all records when a field definition error or invalid input data results in a large number of rejected records. This parameter is also useful for creating small files for test purposes.

Setting NUMREC to zero (0) prevents the input data set from being opened.

PASSWORD: Password for FDT File

If the FDT parameter is specified and the file is password-protected, this parameter must be used to provide a valid password for that file.

RECFM: Input Record Format (VSE Only)

You *must* specify the input record format with this parameter as follows:

F	fixed length, unblocked (requires that you also specify the LRECL parameter)
FB	fixed length, blocked (requires that you also specify the LRECL parameter)
V	variable length, unblocked
VB	variable length, blocked
U	undefined

Under z/OS, the record format is taken from the input data set label or DD statement.

Under BS2000, the record format is taken from the catalog entry or FILE statement.

SPAN: Enabling Spanned Records

The `SPAN` parameter allows the record to be spanned after compression if its compressed size exceeds the Data Storage block size of the device.

UACODE: User Encoding for Input Alphanumeric Fields

UACODE defines the user encoding of the sequential input of alphanumeric fields. If you specify UACODE, you *must* also specify FACODE.

UARC: Architecture for Input Uncompressed User Data

The UARC parameter specifies the architecture of the sequential input of the uncompressed user data. The "userdata-architecture-key" is an integer which is of the sum of the following numbers:

byte order	b=0	high-order byte first
	b=1	low-order byte first
encoding family	e=0	ASCII encoding family
	e=2	EBCDIC encoding family (default)
floating-point format	f=0	IBM370 floating-point format
	f=4	VAX floating-point format
	f=8	IEEE floating-point format

The default is ARC = b + e + f = 2; that is, high-order byte first; EBCDIC encoding family; and IBM370 floating-point format (b=0; e=2; f=0).

User data from an Intel386 PC provides the example: b=1; e=0; f=8; or ARC=9.

USERISN: User ISN Assignment

The ISN for each record is to be user-defined. If this parameter is omitted, the ISN for each record is assigned by Adabas.

If USERISN is specified, you must provide the ISN to be assigned to each record as a four-byte binary number immediately preceding each data record. If the MINISN parameter is specified, USERISN cannot be specified.

If USERISN is specified with HEADER=YES, the ISN immediately follows the ADAH header as part of the logical record.

The format for fixed or undefined length input records with user-defined ISNs is:

```
userisn/data
```

The format for variable-length input records with user-defined ISNs is

```
length/xx/userisn/data
```

where

length	is a two-byte binary physical record length (length of record data, plus 8 bytes).
xx	is a two-byte field containing binary zeros.
userisn	is a four-byte binary ISN to be assigned to the record.
data	is input record data.

ISNs may be assigned in any order, must be unique (for the file), and must not exceed the MAXISN setting specified for the file (see the **ADALOD utility** documentation).

ADACMP does not check for unique ISNs or for ISNs that exceed MAXISN. These checks are performed by the ADALOD utility.

UWCODE: User Encoding for Input Wide-Character Fields

UWCODE defines the user encoding of the sequential input of wide-character fields. If you specify UWCODE, you *must* also specify FWCODE.

For user input, all wide-character fields are encoded in the same code page. It is not possible to select different encodings for different fields in the same ADACMP run.

ADACMP COMPRESS Examples

Example 1:

```
ADACMP   COMPRESS
ADACMP   FNDEF='01,AA,7,A,DE,FI'           Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'       Field AB
ADACMP   FNDEF='01,GA'                     Group GA
ADACMP   FNDEF='02,AC,15,A,NU'             Field AC
ADACMP   FNDEF='02,AD,2,P,FI'              Field AD
ADACMP   FNDEF='02,AE,5,P,NU'              Field AE
ADACMP   FNDEF='02,AF,6,W'                 Field AF
ADACMP   COLDE='7,Y1=AF'                   Collation descriptor Y1
ADACMP   SUBDE='BB=AA(1,4)'                Subdescriptor BB
ADACMP   SUPDE='CC=AA(1,4),AD(1,1)'        Superdescriptor CC
ADACMP   HYPDE='1,DD,4,A,MU=AB,AC,AD'      Hyperdescriptor DD
ADACMP   PHONDE='EE(AA)'                   Phonetic descriptor EE
ADACMP   SUBFN='FF=AA(1,2)'                Subfield FF
ADACMP   SUPFN='GG=AA(1,4),AD(1,1)'        Superfield GG
```

Field AA	is defined as level 1, 7 bytes alphanumeric, descriptor, fixed storage option.
Field AB	is defined as level 1, 15 bytes alphanumeric, descriptor, multiple value field, null value suppression.
GA	is a group containing fields AC, AD, AE, and AF.
BB	is a subdescriptor (positions 1-4 of field AA).
CC	is a superdescriptor (positions 1-4 of field AA and position 1 of field AD).

DD	is a hyperdescriptor consisting of fields AB, AC and AD. DD is assigned hyperexit 1.
EE	is a phonetic descriptor derived from field AA.
FF	is a subfield (positions 1-2 of field AA).
GG	is a superfield (positions 1-4 of AA and position 1 of AD).
Y1	is a collation descriptor for AF and is assigned to collation descriptor user exit 7 (CDX07).

Example 2:

```
ADACMP   COMPRESS
ADACMP   FORMAT='AG,6,U,AF,4X,AA,'    input record format
ADACMP   FORMAT='AB,AC'               continuation of FORMAT statement
ADACMP   FNDEF='01,AA,10,A,NU'        field definitions
ADACMP   FNDEF='01,AB,7,U,NU'
ADACMP   FNDEF='01,AF,5,P,NU'
ADACMP   FNDEF='01,AG,12,P,NU,DE'
ADACMP   FNDEF='01,AC,3,A,NU,DE'
```

The input record format is provided explicitly using the FORMAT parameter. ADACMP uses this format as the basis for processing fields from the input record. The FDT for the file corresponds to the structure specified in the FNDEF statements.

Example 3:

```
ADACMP COMPRESS
ADACMP FORMAT='AG,AF,4X,AA,AB,AC'     input record format
ADACMP FDT=8                          FDT same as file 8
```

The input record format is provided explicitly using the FORMAT parameter. The FDT to be used is the same as that currently defined for Adabas file 8.

Example 4:

```
ADACMP   COMPRESS NUMREC=2000,USERISN
ADACMP   FNDEF='01,AA,7,A,DE,FI'            Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'        Field AB
```

The number of input records to be processed is limited to 2,000. The ISN for each record is to be provided by the user.

Example 5:

```
ADACMP   COMPRESS RECFM=FB,LRECL=100
ADACMP   FNDEF='01,AA,7,A,DE,FI'            Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'        Field AB
```

A VSE input file contains fixed length (blocked) records. The record length is 100 bytes.

15 DECOMPRESS: Decompress an Adabas File

- Optional Parameters and Subparameters 96
- Decompressing Multiclient Files 101
- ADACMP DECOMPRESS Examples 101

The DECOMPRESS function decompresses data either

- from output unloaded by the ADAULD UNLOAD utility function; or
- directly from a single compressed Adabas file when the file number is specified with the INFILE parameter.

When decompressing data directly from the INFILE file, DECOMPRESS first performs an ADAULD UNLOAD/MODE=SHORT function. This can save time over separate ADAULD and ADACMP DECOMPRESS operations.

```
ADACMP  DECOMPRESS  [CODE = cipher-code ]
                    [FORMAT = output-record-format-definition ]
                    [HEADER = { YES | NO } ]
                    [INFILE = file-number ]
                       [ETID = owner-id ]
                       [LPB = { prefetch-buffer-size | based-on-ADARUN-lu } ]
                       [PASSWORD = ' password ' ]
                       [SORTSEQ = { descriptor [,NU] | ISN | physical-sequence } ]
                       [UTYPE = { EXF | EXU } ]
                    [ISN]
                    [LOBVALUES = { YES | NO } ]
                    [MAXLOGRECLEN = buffer-size ]
                    [NOUSERABEND]
                    [NUMREC = {number-of-records |  all-records  } ]
                    [TRUNCATE]
                    [UACODE = userdata-alpha-key ]
                    [UWCODE = userdata-wide-key ]
                    [UARC = {architecture-key |  2 } ]
```

This chapter describes the syntax and parameters of the ADACMP DECOMPRESS function.

Optional Parameters and Subparameters

CODE: Cipher Code

If the file to be decompressed is ciphered, the cipher code that was used when the file was compressed must be specified with this parameter. See the *Adabas Security* documentation for additional information on the use of ciphering.

ETID: Multiclient File Owner ID

ETID specifies an owner ID for a multiclient file specified by INFILE. ADACMP DECOMPRESS selectively decompresses only those records in the multiclient file assigned to the owner ID

specified by ETID. The ETID value must be the same as that assigned to the records when they were loaded into the multiclient file.

FORMAT: Output Record Format Definition

FORMAT allows decompression to a format other than that specified by the FDT. It can be used to change the FDT of an existing file and, in particular, the structure of a periodic (PE) group.

The FORMAT parameter syntax is the same as the format buffer syntax used for read commands except that text cannot be inserted (text is not compressible/decompressible); see the *Adabas Command Reference Guide* documentation for more information.

Note: The FORMAT parameter does not check whether all related data fields have been processed during decompression.

For example, if a multiple-value (MU) field defined as:

```
01,AA,8,A,MU
```

has five occurrences, and the ADACMP DECOMPRESS FORMAT parameter specifies:

```
AA1-4
```

then only the first four AA field values are decompressed; no indication is given regarding the fifth field value. This also applies to PE field occurrences and length overrides.

HEADER

This optional parameter indicates whether or not the ADACMP decompression logic should produce the ADACMP segmented record headers (ADAH and ADAC) as part of the decompressed output. Valid values are YES or NO; the default is NO.

HEADER=NO is the format accepted and produced by ADACMP in Adabas versions prior to Adabas 8. When it is specified, the decompressed output records produced by ADACMP will contain only the data for the fields of the file being processed. Each data record must fit into one physical record of the sequential input data set (less than 32 KB). If the data exceeds this size, the records in error are written to the DDFEHL error data set.

HEADER=YES can only be specified if you are running Adabas 8. If HEADER=YES is specified, each output decompressed record produced by ADACMP begins with either an ADAH or ADAC header, relating the physical record with a logical record. Each logical record can be larger than 32 KB. The header in each physical record defines the position of the data following it within the logical record. For complete information about segmented records in ADACMP, read ***Segmented Record Considerations***, elsewhere in this guide. DSECTs for the ADAH and ADAC headers can be found in members ADAH and ADAC of the distributed Adabas SRCE data set.

INFILE: Number of File to Be Decompressed

The INFILE parameter allows you to decompress a file without first unloading it with the ADAULD utility. If the INFILE parameter is not specified, the input is read from a sequential

(DD/EBAND) file. With the ETID parameter, INFILE permits selectively decompressing records from a multiclient file. When decompressing multiclient files, refer to the section ***Decompressing Multiclient Files***, elsewhere in this section.

ISN: Include ISN in Decompressed Output

The ISN of each record is to be included with each decompressed record output. If this parameter is omitted, the ISN will not be included with each record.

If ISN is specified with HEADER=YES, the ISN immediately follows the ADAH header as part of the logical record. The DSECT for the ADAH header can be found in member ADAH of the distributed Adabas SRCE data set.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer for the ADACMP DECOMPRESS INFILE function. The maximum value is 32,760 bytes. The default is calculated by Adabas, depending on the ADARUN LU value in effect for the nucleus.

LOBVALUES: LB Field Size Indicator

The LOBVALUES parameter should only be specified for files containing large object (LB) fields.

This optional parameter indicates whether long LB field values (larger than 253 bytes) or short LB field values (up to 253 bytes) are expected in the ADACMP DECOMPRESS output data. Valid values for this parameter are "YES" and "NO"; the default is "NO".

If "NO" is specified for this parameter, the uncompressed output data may contain only LB fields up to 253 bytes long and references to LB field values stored in a *LOB file*. In this case, you cannot specify an LB field in the ADACMP DECOMPRESS FORMAT parameter. During processing, ADACMP DECOMPRESS reads only the base file records as input; if the base file contains references to LB field values in a LOB file, ADACMP DECOMPRESS does not read them, but only reproduces the references in the output.

If "YES" is specified for this parameter, the uncompressed output data will contain the LB field values present for the record. In this case, the INFILE parameter must also be specified to identify the file number of the *base file* of a *LOB file group* whose data is to be decompressed. During processing, as ADACMP DECOMPRESS reads and decompresses the records from the base file, it reads any referenced LB field values from the *LOB file*.

Note: An ADACMP DECOMPRESS function with LOBVALUES=NO followed by an ADACMP COMPRESS function with LOBVALUES=NO can be used to modify the FDT of the *base file* in the *LOB file group*.

MAXLOGRECLEN: Buffer Size

This optional parameter can be used to specify the size, in bytes, of a buffer used by ADACMP to assemble logical records that span one or more physical records of uncompressed output data. This buffer is allocated only if HEADER=YES is also specified. Otherwise, the setting of MAXLOGRECLEN is ignored. The default value of MAXLOGRECLEN is 1,048,576 bytes (1 MB).

If the value specified by MAXLOGRECLEN is appended by the letter "K", it is multiplied by 1024. The minimum value is 32768 bytes.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Number of Records to Be Processed

NUMREC specifies the number of input records to be processed. If this parameter is omitted, all input records contained on the input data set are processed.

Use of NUMREC is recommended for the initial ADACMP execution if a large number of records are contained on the input data set. This avoids unneeded processing of all records when a field definition error or invalid input data causes a large number of rejected records. NUMREC is also useful for creating small files for test purposes.

PASSWORD: Password for INFILE

The PASSWORD parameter must specify the correct password if the file is to be decompressed directly from a password-protected Adabas file.

SORTSEQ: Processing Sequence for INFILE File

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence. SORTSEQ can be specified only when INFILE is also specified.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a hyperdescriptor, a phonetic descriptor, a multiple-value descriptor field, or a descriptor contained in a periodic group.

If the descriptor name refers to a field defined with the null suppression (NU) option, you must specify ",NU" after the descriptor name. In this case, records of the descriptor that contain null values are not decompressed. If NU is not specified in this case (the default), ADACMP rejects NU descriptors.

Note: Even when the descriptor field is not null-suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TRUNCATE: Truncate Excess Alphanumeric Characters

The TRUNCATE parameter enables truncation of compressed alphanumeric data during decompression. When TRUNCATE is specified and ADACMP DECOMPRESS operation finds an alphanumeric field containing more characters than the FDT description allows for the

field, the extra characters are truncated. If TRUNCATE is not specified, alphanumeric records with extra characters are written to the DDFEHL data set. Non-alphanumeric fields cannot be truncated.

UACODE: Encoding Protocol for Output Alphanumeric Fields

UACODE defines the encoding of the sequential output of alphanumeric fields. This parameter allows you to override the user encoding for alphanumeric fields passed in the header of the compressed sequential input.

UARC: Architecture for Output Uncompressed User Data

The UARC parameter specifies the architecture of the sequential output of the uncompressed user data. This parameter allows you to override the user encoding passed in the header of the compressed sequential input.

The 'userdata-architecture-key' is an integer which is the sum of the following numbers:

byte order	b=0	high-order byte first
	b=1	low-order byte first
encoding family	e=0	ASCII encoding family
	e=2	EBCDIC encoding family (default)
floating-point format	f=0	IBM370 floating-point format
	f=4	VAX floating-point format
	f=8	IEEE floating-point format

The default is ARC = b + e + f = 2; that is, high-order byte first; EBCDIC encoding family; and IBM370 floating-point format (b=0; e=2; f=0).

User data from an Intel386 PC provides the example: b=1; e=0; f=8; or ARC=9.

UTYPE: User Type

The user type to be in effect when unloading the file specified by INFILE. Allowed values are

EXF	no access/update allowed for other users of the file.
EXU	access only is allowed for other users of the file. EXU is the default.

UWCODE: Encoding Protocol for Output Wide-Character Fields

UWCODE defines the encoding of the sequential output of wide-character fields. This parameter allows you to override the user encoding for wide-character fields passed in the header of the compressed sequential input.

Decompressing Multiclient Files

ADACMP decompresses Adabas data to a sequential user file. The DECOMPRESS function can decompress records selectively if the INFILE parameter specifies a multiclient file and a valid ETID value is specified.

The DECOMPRESS function skips the owner ID, if present. The output of a DECOMPRESS operation on a multiclient file contains neither owner ID nor any ETID information.

If the INFILE parameter specifies a multiclient file for the DECOMPRESS function, you can use the ETID parameter to limit decompression to records for a specific user only. ADACMP then reads and decompresses records only for the specified user. If the ETID parameter is not specified when decompressing a multiclient file, all records in the file are decompressed.

Example:

Only records owned by USER1 from file 20 are decompressed to a sequential output file:

```
ADACMP  DECOMPRESS  INFILE=20,ETID=USER1
```

ADACMP DECOMPRESS Examples

Example 1:

The DECOMPRESS function is to be executed. The input data set to be used is the output of a previous execution of the ADAULD utility:

```
ADACMP DECOMPRESS
```

Example 2:

Adabas file 23 is to be decompressed. The ISN of each record is to be included in the decompressed output:

```
ADACMP DECOMPRESS INFILE=23,ISN
```

16 Field Definition Statements

- FNDEF: Field and Group Definition 105
- FNDEF: Periodic Group Definition 121
- COLDE: Collation Descriptor Definition 124
- HYPDE: Hyperdescriptor Definition 127
- PHONDE: Phonetic Descriptor 130
- SUBDE: Subdescriptor Definition 132
- SUBFN: Subfield Definition 134
- SUPDE: Superdescriptor Definition 135
- SUPFN: Superfield Definition 143

The field definitions provided as input to ADACMP are used to:

- provide the length and format of each field contained in the input record. This enables ADACMP to determine the correct field length and format during editing and compression.
- create the Field Definition Table (FDT) for the file. This table is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

The following syntax must be followed when entering field definitions. A minimum of one and a maximum of 926 definitions may be specified.

Statement Type	Syntax
Field and Group	**FNDEF = '***level*, *name* **[** , *length*, *format* **] [, MU [(***occurrences* **)]] [** , *option* **] ... '**
Periodic Group	**FNDEF = '***level*, *name* **[** , **PE [(** *occurrences* **)]]'**
Collation descriptor	**COLDE = '***number*, *name* **[** , **UQ [, XI]] =** *parent-field***'**
Hyperdescriptor	**HYPDE = '***number*, *name*, *length*, *format* **[{** , *option***} ...] = {** *parent-field* **} , ...'**
Phonetic descriptor	**PHONDE = '** *name* **(***field***)'**
Subdescriptor	**SUBDE = '***name* **[, UQ [XI]] =** *parent-field* **(***begin*, *end***)'**
Subfield	**SUBFN = '** *name* **=** *parent-field* **(***begin*, *end***)'**

Statement Type	Syntax
Superdescriptor	**SUPDE = '***name* **[, UQ [XI]] = { *parent-field* (*begin*, *end*) } , ...'**
Superfield	**SUPFN = '***name* **=** *parent-field* **(***begin*, *end***)[,** *parent-field* **(** *begin* **,** *end* **)]...'**

User comments may be entered to the right of each definition. At least one blank must be present between a definition and any user comments.

Each of these definition types is described in this chapter.

FNDEF: Field and Group Definition

The FNDEF parameter can be used to specify an Adabas field or group definition. The syntax used in constructing field and group definition entries is:

FNDEF = '*level*, *name* **[,** *length*, *format* **] [, MU [(***occurrences***)]] [,** *option* **] ... '**

Level number and name are required. Any number of spaces may be inserted between definition entries.

Each FNDEF parameter is described in this section.

- level
- name
- length
- format
- occurrences
- field options

The MU parameter is documented in ***field options***

level

The level number is a one- or two-digit number in the range 01-07 (the leading zero is optional) used in conjunction with field grouping. Fields assigned a level number of 02 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number.

The definition of a group enables reference to a series of fields (may also be only 1 field) by using the group name. This provides a convenient and efficient method of referencing a series of consecutive fields.

Level numbers 01-06 may be used to define a group. A group may consist of other groups. When assigning the level numbers for nested groups, no level numbers may be skipped.

In the following example, fields A1 and A2 are in group GA. Field B1 and group GC (consisting of fields C1 and C2) are in group GB:

FNDEF='01,GA'	group	
FNDEF='02,A1,...'		elementary or multiple-value field
FNDEF='02,A2,...'		elementary or multiple-value field
FNDEF='01,GB'	group	
FNDEF='02,B1,...'		elementary or multiple-value field
FNDEF='02,GC'	group (nested)	
FNDEF='03,C1,...'		elementary or multiple-value field
FNDEF='03,C2,...'		elementary or multiple-value field

name

The name to be assigned to the field (or group).

Names must be unique within a file. The name must be two characters long: the first character must be alphabetic; the second character can be either alphabetic or numeric. No special characters are permitted.

The values E0-E9 are reserved as edit masks and may not be used.

Valid Names	Invalid Names
AA	A (not two characters)
B4	E3 (edit mask)
S3	F* (special character)
WM	6M (first character not alphabetic)

length

The length of the field (expressed in bytes). The length value is used to

- indicate to ADACMP the length of the field as it appears in each input record; and
- define the standard (default) length to be used by Adabas during command processing.

The standard length specified is entered in the FDT and is used when the field is read/updated unless the user specifies a length override.

The maximum field lengths that may be specified depend on the "format" value:

Format	Maximum Length
Alphanumeric (A)	253 bytes
Binary (B)	126 bytes
Fixed Point (F)	4 bytes (always exactly 2 or 4 bytes)
Floating Point (G)	8 bytes (always exactly 4 or 8 bytes)
Packed Decimal (P)	15 bytes
Unpacked Decimal (U)	29 bytes
Wide-character (W)	253 bytes*

** Depending on the FWCODE attribute value, the maximum byte length of the W field may be less than 253. For example, if the default value of FWCODE is used (that is, Unicode), the maximum length is 252 (2 bytes per character).*

Standard length may not be specified with a group name.

Standard length does not limit the size of any given field value unless the FI option is used - see ***FI: Fixed Storage*** . A read or update command may override the standard field length, up to the maximum length permitted for that format.

If standard length is zero for a field, the field is assumed to be a variable-length field. Variable-length fields have no standard (default) length. A length override for fixed-point (F) fields can specify a length of two or four bytes only; for floating-point (G) fields, the override can specify four or eight bytes only.

If a variable-length field is referenced without a length override during an Adabas command, the value in the field will be returned preceded by a one-byte binary length field (including the length byte itself). This length value must be specified when the field is updated, and also in the input records that are to be processed by ADACMP. If the field is defined with the long alpha (LA) option, the value is preceded by a two-byte binary length field (including the two length bytes).

format

The standard format of the field (expressed as a one-character code):

A	Alphanumeric (left-justified)
B	Binary (right-justified, unsigned/positive)
F	Fixed point (right-justified, signed, two's complement notation)
G	Floating point (normalized form, signed)
P	Packed decimal (right-justified, signed)
U	Unpacked decimal (right-justified, signed)
W	Wide character (left-justified)

The standard format is used to

- indicate to ADACMP the format of the field as it appears in each input record; and
- define the standard (default) format to be used by Adabas during command processing. The standard format specified is entered in the FDT and is used when the field is read/updated unless the user specifies a format override.

Standard format must be specified for a field. It may not be specified with a group name. When the group is read (written), the fields within the group are always returned (must be provided) according to the standard format of each individual field. The format specified determines the type of compression to be performed on the field.

A fixed-point field is either two or four bytes long. A positive value is in normal form, and a negative value in two's complement form.

A field defined with floating-point format may be either four bytes (single precision) or eight bytes (double precision) long. Conversion of a value of a field defined as floating point to another format is supported.

If a binary field is to be defined as a descriptor, and the field may contain both positive and negative numbers, "F" format should be used instead of "B" format because "B" format assumes that all values are unsigned (positive).

Like an alphanumeric field, a wide-character field may be a standard length in bytes defined in the FDT, or variable length. Any non-variable format override for a wide-character field must be compatible with the user encoding; for example, a user encoding in Unicode requires an *even* length. Format conversion from numbers (U, P, B, F, G) to wide-character format is not allowed.

occurrences

The number of occurrences of MU fields that will occur in a record if the MUPEX option is specified. This is an optional parameter.

field options

Options are specified by the two-character codes. More than one code may be specified (as applicable for the field). They may be specified in any order, separated by a comma.

The available options for field and group definitions are listed in the following table. For more information about a specific option, click on its name in the table.

Note: The PE option is another Adabas field option. However, it is available only if you are specifying a periodic group. For more information, read *FNDEF: Periodic Group Definition*, elsewhere in this section.

Code	Description
DE: Descriptor	The field is to be a descriptor (key).
FI: Fixed Storage	The field is to have a fixed storage length; values are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.
LA: Long Alpha Option	This A or W-format variable-length field may contain a value up to 16,381 bytes long.
LB: Large Object Option	An alphanumeric field may contain up to 2,147,483,643 (about 2 GB) of data.
MU: Multiple-Value Field	The field may contain up to about 65,534 values in a single record.
NB: Blank Compression Option	Trailing blanks should not be removed (compressed) from the LA or LB fields.
NU: Null Value Suppression	Null values occurring in the field are to be suppressed.
NV: No Conversion	This A or W-format field is to be processed in the record buffer without being converted.
UQ: Unique Descriptor	The field is to be a unique descriptor; that is, for each record in the file, the descriptor must have a different value.
XI: Exclude Instance Number	For this field, the index (occurrence) number is excluded from the UQ option set for a PE.
NC: SQL Null Value Option	Adabas includes two data definition options, **NC** and **NN**, to provide SQL-compatible null representation for Software AG's mainframe Adabas SQL Gateway(ACE) and other Structured Query Language (SQL) database query languages. For more information about these options, read *Representing SQL Null Value*, elsewhere in this section.
NN: SQL Not Null Option	

Code	Description
	With the **NC** option, the field may contain a null value that satisfies the SQL interpretation of a field having no value; that is, the field's value is not defined (not counted). With the **NN** option, the field defined with the NC option must always have a value defined; it cannot contain an SQL null (not null).

DE: Descriptor

DE indicates that the field is to be a descriptor (key). Entries will be made in the Associator inverted list for the field, enabling the field to be used in a search expression, as a sort key in a FIND command, to control logical sequential reading, or as the basis for file coupling.

The descriptor option should be used judiciously, particularly if the file is large and the field that is being considered as a descriptor is updated frequently.

Although the definition of a descriptor field is independent of the record structure, note that if a descriptor field is not ordered first in a record and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to reorder the field first for each record of the file.

FI: Fixed Storage

FI indicates that the field is to have a fixed storage length. Values in the field are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.

The FI option is recommended for fields with a length of one or two bytes that have a low probability of containing a null value (personnel number, gender, etc.) and for fields containing values that cannot be compressed.

The FI option is not recommended for multiple-value fields, or for fields within a periodic group. Any null values for such fields are not suppressed (or compressed), which can waste disk storage space and increase processing time.

The FI option *cannot* be specified for

- U-format fields;
- NC, NN, or NU option fields;
- variable-length fields defined with a length of zero (0) in the FNDEF statement;
- a descriptor within a periodic (PE) group.

A field defined with the FI option *cannot* be updated with a value that exceeds the standard length of the field.

Example of FI usage:

	Definition	User Data	Internal Representation
Without FI Option	FNDEF='01,AA,3,P'	33104C 00003C	0433104F (4 bytes) 023F (2 bytes)
With FI Option	FNDEF='01,AA,3,P,FI'	33104C 00003C	33104F (3 bytes) 00003F (3 bytes)

LA: Long Alpha Option

The LA (long alphanumeric) option can be specified for variable-length alphanumeric and wide format fields; i.e., A or W format fields having a length of zero in the field definition (FNDEF). With the LA option, such a field can contain a value up to 16,381 bytes long.

An alpha or wide field with the LA option is compressed in the same way as an alpha or wide field without the option. The maximum length that a field with LA option can actually have is restricted by the block size where the compressed record is stored.

When a field with LA option is updated or read, its value is either specified or returned in the record buffer, preceded by a two-byte length value that is inclusive (field length, plus two).

A field with LA option:

- can also have the NU, NC/NN, NV, or MU option;
- can be a member of a PE group;
- cannot have the FI option;
- cannot be a descriptor field;
- cannot be a parent of a sub-/superfield, sub-/superdescriptor, hyperdescriptor, or phonetic descriptor; and
- cannot be specified in the search buffer, or response code 61 occurs.

For more information, read *Specifying Field Lengths of LA (Long Alpha) Fields in Format Buffers* and *Specifying Field Lengths of LA (Long Alpha) Fields in Record Buffers* in the *Adabas Command Reference Guide* .

Example of LA usage:

	Definition	User Data	Internal Representation
Without LA Option	FNDEF='01,BA,0,A'	X'06',C'HELLO' --	X'06C8C5D3D3D6' (1-byte length) --
With LA Option	FNDEF='01,BA,0,A,LA'	X'0007',C'HELLO' X'07D2',C' ... (2000 data bytes) ...'	X'06C8C5D3D3D6' (1-byte length) X'87D2 ... (2000 data bytes) ... '

LB: Large Object Field Option

The large object (LB) option can be specified for some fields to identify them as *large object* fields. LB fields can contain up to 2,147,483,643 bytes (about 2 GB) of data. At this time, you can only store and retrieve entire LB fields, you cannot store and retrieve portions of an LB field.

The format of an LB field must be "A" (alphanumeric) and its default field length must currently be defined as zero.

LB fields *cannot* be:

- Descriptors or parents of a special (phonetic, sub-, super-, or hyper-) descriptor.
- Defined with the FI or LA options.

  To assist you in determining whether to define a field as an LA or an LB field, read *Comparing LA and LB Fields*, in the *Adabas Concepts and Facilities Manual*.
- Specified in a search buffer or in format selection criteria in a format buffer.

LB fields may be:

- Defined with any of the following options: MU, NB, NC, NN, NU, or NV
- Part of a simple group or a PE group.

The presence of the NB (no blank compression) field option in the LB field definition indicates whether on not Adabas removes trailing blanks in LB fields containing characters.

LB fields containing both binary and character data are supported. An LB field defined with both the NV and NB options can store binary large object data, as Adabas will not modify binary LB fields in any way. The identical LB binary byte string that was stored is what is retrieved when the LB field is read. In addition, because LB fields containing binary values are defined with the NV and the NB options, Adabas will not convert LB field binary values according to some character code page nor will it cut off trailing blanks in LB fields containing binary values.

Note: LB fields containing binary values are not defined using format B, because format B can imply byte swapping in some environments with different byte orders. Byte swapping does not apply to binary LB fields.

The following table provides some valid example of FDT definitions for LB fields:

FDT Specification	Description
1,L1,0,A,LB,NU	Field L1 is a null-suppressed, character, large object field
1,L2,0,A,LB,NV,NB,NU,MU	Field L2 is a null-suppressed, multiple-value, binary, large object field.

Commands dealing with LB fields must always be directed to the *base file* of a *LOB file group*. User commands against *LOB files* are rejected.

For information on getting started using LB fields, read *Getting Started with Large Object (LB) Fields*, in *Adabas DBA Tasks Manual*.

MU: Multiple-Value Field

The multiple-value field option, indicating that the field may contain more than one value in a single record. If the **MUPEX** parameter is specified and the **MUPECOUNT** parameter is set to "2", the field may contain up to 65,534 values in a single record. If these parameters are not set, the field may contain up to 191 values in a single record. At least one value (even if null) must be present in each record input to ADACMP.

The values are stored according to the other options specified for the field. The first value is preceded by a count field that indicates the number of values currently present for the field. The number of values that are stored is equal to the number of values provided in the ADACMP input record, plus any values added during later updating of the field, less any values suppressed (this applies only if the field is defined with the NU option).

If the number of values contained in each record input to ADACMP is constant, the number can be specified in the MU definition statement in the form MU(*n*), where *n* is the number of values present in each input record. In the following example, three values of the multiple-value field AA are present in each input record:

```
FNDEF='01,AA,5,A,MU(3)' ↩
```

Specifying a value of zero MU(0) indicates that no values are present for the multiple-value field in the input record.

If the number of values is not constant for all input records, a one- or two-byte binary count field (depending on the **MUPECOUNT** parameter setting) must precede the first value in each input record to indicate the number of values present in that record (see also the section ***Input Data Requirements***).

If the FDT is provided (see the **FDT parameter** description), the field count must be contained as a one- or two-byte binary value in each input record (depending on the **MUPECOUNT** parameter setting).

If the input records were created using the DECOMPRESS function, all required count fields are already contained in the input record. In this case, the count must not be specified in the field definition statements.

All values provided during input or updating will be compressed (unless the **FI option** has also been specified). Care should be taken when using the FI and MU options together since a large amount of disk storage may be wasted if a large number of compressible values are present.

If the NU option is specified with the MU option, null values are both logically and physically suppressed. The positional relationship of all values (including null values) is usually maintained in MU occurrences, unless the occurrences are defined with the NU option. If a large number of null values are present in an MU field group, the NU option can reduce the disk storage requirements for the field but should not be used if the relative positions of the values must be maintained.

The NC (or NC/NN) option *cannot* be specified for an MU field.

For information on how to identify MU and PE occurrences greater than 191 in the compressed record, read ***Identifying MU and PE Occurrences Greater Than 191 in Compressed Records***, elsewhere in this section.

Example of MU usage with NU:

```
FNDEF='01,AA,5,A,MU,NU'
```

The original content where "L" is the length of the "value" is:

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

2	L value A	L value C
count	AA1	AA2

Example of MU usage without NU:

```
FNDEF='01,AA,5,A,MU'
```

The original content where "L" is the length of the "value" is

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

NB: Blank Compression

The NB option can be used with **LA** and **LB** fields to control blank compression. When specified, the NB option indicates that Adabas should *not* remove trailing blanks for the field; when not specified, Adabas removes trailing blanks when storing an alphanumeric or wide-character field value. If you specify the NB option for a field, you must also specify the NU or NC option for the field; NB processing requires the use of NC or NU as well.

Note: Fields specified without the NB option can lead to differences in the stored and retrieved lengths of the fields. The retrieved length of a non-NB field is likely to be smaller than the length specified for the field when it is stored due to blank compression. This may matter if the value is not really a character string, but rather a binary value that happens to end with the character codes for a blank. Therefore, if you want the stored and retrieved lengths of a field to be the same, use the NB option.

NU: Null Value Suppression

NU suppresses null values occurring in the field.

Normal compression (NU or FI not specified) represents a null value with two bytes (the first for the value length, and the second for the value itself, in this case a null). Null value suppression represents an empty field with a one-byte empty field indicator. The null value itself is not stored.

A series of consecutive fields containing null values and specifying the NU option is represented by a one-byte empty field (binary 11nnnnnn) indicator, where "nnnnnn" is the number of the fields' successive bytes containing null values, up to a total of 63. For this reason, fields defined with the NU option should be grouped together whenever possible.

If the NU option is specified for a descriptor, any null values for the descriptor are not stored in the inverted list. Therefore, a find command in which this descriptor is used and for which a null value is used as the search value will always result in no records selected, even though there may be records in Data Storage that contain a null value for the descriptor. If a descriptor defined with the NU option is used to control a logical sequence in a read logical sequence (L3/L6) command, those records that contain a null value for the descriptor will not be read.

Descriptors to be used as a basis for file coupling and for which a large number of null values exist should be specified with the NU option to reduce the total size of the coupling lists.

The NU option cannot be specified for fields defined with the combined NC/NN options or with the FI option.

Example of NU usage:

	Definition	User Data	Internal Representation
Normal Compression	FNDEF='01,AA,2,B'	0000	0200 (2 bytes)
With FI Option	FNDEF='01,AA,2,B,FI'	0000	0000 (2 bytes)
With NU Option	FNDEF='01,AA,2,B,NU'	0000	C1 (1 byte)*

* *C1 indicates 1 empty field.*

NV: No Conversion

The "do not convert" option for alphanumeric (A) or wide-character (W) format fields specifies that the field is to be processed in the record buffer without being converted.

Fields with the NV option are not converted to or from the user: the field has the characteristics of the file encoding; that is, the default blank

- for A fields, is always the EBCDIC blank (X'40'); and
- for W fields, is always the blank in the file encoding for W format.

The NV option is used for fields containing data that cannot be converted meaningfully or should not be converted because the application expects the data exactly as it is stored.

The field length for NV fields is byte-swapped if the user architecture is byte-swapped.

For NV fields, "A" format cannot be converted to "W" format and vice versa.

UQ: Unique Descriptor

UQ indicates that the field is to be a unique descriptor. A unique descriptor must contain a different value for each record in the file. In FNDEF statements, the UQ option can only be specified if the DE option is also specified. The UQ option can also be used in SUBDE, SUPDE, and HYPDE statements.

The UQ option *must* be specified if the field is to be used as an ADAM descriptor (see the **ADAMER utility**).

ADACMP does not check for unique values; this is done by the ADALOD utility, or by the ADAINV utility when executing the INVERT function. If a non-unique value is detected during file loading, ADALOD terminates with an error message.

Because ADAINV and ADALOD must execute separately for each file in an expanded file chain, they cannot check for uniqueness across the chain.

However, Adabas does checks the value of unique descriptors across an expanded file chain. If the value being added (N1/N2) or updated (A1) is not unique across all files within the chain, response code 198 is returned.

XI: Exclude Instance Number

By default, the occurrence number of fields within periodic groups (PE) defined as unique descriptors (UQ) is included as part of the descriptor value. This means that the same field value can occur in different periodic group occurrences in different records.

The XI option is used to exclude the occurrence number from the descriptor value for the purpose of determining the the value's uniqueness. If the XI option is set, any field value can occur at most once over all occurrences of the PE field in all records.

Representing SQL Null Value

Adabas includes two data definition options, NC and NN, to provide SQL-compatible null representation for Software AG's mainframe Adabas SQL Gateway (ACE) and other Structured Query Language (SQL) database query languages.

The NC and NN options *cannot* be applied to fields defined

- with Adabas null suppression (NU)
- with fixed-point data type (FI)
- with multiple-values (MU)
- within a periodic group (PE)
- as group fields

In addition, the NN option can only be specified for a field that specifies the NC option.

A parent field for sub-/superfields or sub-/superdescriptors can specify the NC option. However, parent fields for a single superfield or descriptor cannot use a mix of NU and NC fields. If any parent field is NC, no other parent field can be an NU field, and vice versa.

Examples:

A correct ADACMP COMPRESS FNDEF statement for defining the field AA and assigning the NC and NN option:

```
ADACMP  FNDEF='01,AA,4,A,NN,NC,DE'
```

Incorrect uses of the NC/NN option that would result in an ADACMP utility ERROR-127:

Incorrect Example	Reason
ADACMP FNDEF='01,AA,4,A,NC,NU'	NU and NC options are not compatible
ADACMP FNDEF='01,AB,4,A,NC,FI'	NC and FI options are not compatible
ADACMP FNDEF='01,PG,PE' ADACMP FNDEF='02,P1,4,A,NC'	NC option within a PE group is not allowed

This section covers the following topics:

- NC: SQL Null Value Option
- Null Indicator Value
- NN: SQL Not Null Option

NC: SQL Null Value Option

Without the NC (not counted) option, a null value is either zero or blank depending on the field's format.

With the NC option, zeros or blanks specified in the record buffer are interpreted according to the ***null indicator value***: either as true zeros or blanks (that is, as *significant* nulls) or as undefined values (that is, as true SQL or *insignificant* nulls).

If the field defined with the NC option has no value specified in the record buffer, the field value is always treated as an SQL null.

When interpreted as a true SQL null, the null value satisfies the SQL interpretation of a field having no value. This means that no field value has been entered; that is, the field's value is not defined.

The null indicator value is thus responsible for the internal Adabas representation of the null. For more information, read the next section in this guide, ***Null Indicator Value*** , and read *Search Buffers* in the *Adabas Command Reference Guide*.

The following rules apply when compressing or decompressing records containing NC fields:

1. If the FORMAT parameter is specified, ADACMP behaves in the same way the nucleus does for update-type commands. See the *Adabas Command Reference Guide* documentation.
2. If the FORMAT parameter is *not* specified:
   - For *compression*:

     Only the value of the NC field is placed in the input record; the two null value indicator bytes must be omitted. The value is compressed as if the null value indicator bytes were set to zero. It is not possible to assign a null value to an NC field using this method.

     Example:

Field Definition Table (FDT) definition	FNDEF='01,AA,4,A,NC'
Input record contents:	MIKE

- For *decompression*:

  If the value of an NC field is *not significant*, the record is written to DDFEHL (or FEHL) with response code 55.

  If the value of an NC field is *significant*, the value is decompressed as usual. There are no null indicator bytes.

  Example:

Field Definition Table (FDT) definition	FNDEF='01,AA,4,A,NC'
Output record contents	MIKE

Null Indicator Value

The *null indicator* value is always two bytes long and has fixed-point format, regardless of the data format. It is specified in the record buffer when a field value is added or changed; it is returned in the record buffer when the field value is read.

For an update (Ax) or add (Nx) command, the null indicator value must be set in the record buffer position that corresponds to the field's designation in the format buffer. The setting must be one of the following:

Hex Value	Indicates that . . .
FFFF	the field's value is set to "undefined", an insignificant null; the differences between no value, binary zeros, or blanks for the field in the record buffer are ignored; all are interpreted equally as "no value".
0000	no value, binary zeros, or blanks for the field in the record buffer are interpreted as significant null values.

For a read (L*x*) or find with read (S*x* with format buffer entry) command, your program must examine the null indicator value (if any) returned in the record buffer position corresponding to the field's position in the format buffer. The null indicator value is one of the following values, indicating the meaning of the actual value that the selected field contains:

Hex Value	Indicates that . . .
FFFF	a zero or blank in the field is not significant.
0000	a zero or blank in the field is a significant value; that is, a true zero or blank.
xxxx	the field is truncated. The null indicator value contains the length (xxxx) of the entire value as stored in the database record if the length is less than 32,768.
0001	the field is significant and the value is truncated, and the length of the value does not fit into the S element because it is greater than 32, 767.

Example:

The field definition of a null represented in a two-byte Adabas binary field AA defined with the NC option is:

01,AA,2,B,NC

For a . . .	Null Indicator Value (Record Buffer)	Data	Adabas Internal Representation
non-zero value	0 (binary value is significant)	0005	0205
blank	0 (binary null is significant)	0000 (zero)	0200
null	FFFF (binary null is not significant)	(not relevant)	C1

NN: SQL Not Null Option

The NN (*not null* or *null value not allowed*) option may only be specified when the NC option is also specified for a data field. The NN option indicates that an NC field must always have a value (including zero or blank) defined; it cannot contain "no value".

The NN option ensures that the field will not be left undefined when a record is added or updated; a significant value must always be set in the field. Otherwise, Adabas returns a response code 52.

The following example shows how an insignificant null would be handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option:

Example

An insignificant null handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option is as following:

Option	Field Definition	Null Indicator Value	Adabas Internal Representation
With NN	01,AA,2,A,NC,NN	FFFF (insignificant null)	none; response code 52 occurs
Without NN	01,AA,2,A,NC	FFFF (insignificant null)	C1

FNDEF: Periodic Group Definition

The syntax used in constructing periodic group definition entries is:

FNDEF = '*level*, *name* **[, PE [(** *occurrences* **)]]'**

Level number and name are required. Any number of spaces may be inserted between definition entries.

Each FNDEF parameter in these definitions is described in this section.

- level
- name
- PE: Periodic Group
- occurrences

level

The level number is a one- or two-digit number in the range 01-07 (the leading zero is optional) used in conjunction with field grouping. Fields assigned a level number of 02 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number.

The definition of a group enables reference to a series of fields (may also be only 1 field) by using the group name. This provides a convenient and efficient method of referencing a series of consecutive fields.

Level numbers 01-06 may be used to define a group. A group may consist of other groups. When assigning the level numbers for nested groups, no level numbers may be skipped.

In the following example, fields A1 and A2 are in group GA. Field B1 and group GC (consisting of fields C1 and C2) are in group GB:

FNDEF='01,GA'	group	
FNDEF='02,A1,...'		elementary or multiple-value field
FNDEF='02,A2,...'		elementary or multiple-value field
FNDEF='01,GB'	group	
FNDEF='02,B1,...'		elementary or multiple-value field
FNDEF='02,GC'	group (nested)	
FNDEF='03,C1,...'		elementary or multiple-value field
FNDEF='03,C2,...'		elementary or multiple-value field

name

The name to be assigned to the field (or group).

Names must be unique within a file. The name must be two characters long: the first character must be alphabetic; the second character can be either alphabetic or numeric. No special characters are permitted.

The values E0-E9 are reserved as edit masks and may not be used.

Valid Names	Invalid Names
AA	A (not two characters)
B4	E3 (edit mask)
S3	F* (special character)
WM	6M (first character not alphabetic)

PE: Periodic Group

The periodic group field option, indicating that the group field is to be followed by a periodic group definition that may occur multiple times in a given record. If the **MUPEX** parameter is specified and the **MUPECOUNT** parameter is set to "2", the periodic group may occur up to 65,534 times in a single record. If these parameters are not set, the periodic group may occur up to 191 times in a single record. At least one occurrence (even if it contains all null values) must be present in each ADACMP input record.

A periodic group:

- May comprise one or more fields. A maximum of 254 elementary fields may be specified. Descriptors and/or multiple value fields and other groups may be specified, but a periodic group may not contain another periodic group.
- If the **MUPEX** parameter is specified and the **MUPECOUNT** parameter is set to "2", a periodic group may occur from 0 to 65,534 times within a given record, although at least one occurrence

(even if it contains all null values) must be present in each ADACMP input record. If these parameters are not set, the periodic group may occur from 0 to 191 times within a given record.

- Must be defined at the 01 level. All fields in the periodic group must immediately follow and must be defined at level 02 or higher (in increments of 1 to a maximum of 7). The next 01 level definition indicates the end of the current periodic group.
- May only be specified with a group name. Length and format parameters may not be specified with the group name.

For information on how to identify MU and PE occurrences greater than 191 in the compressed record, read ***Identifying MU and PE Occurrences Greater Than 191 in Compressed Records***, elsewhere in this section.

The following are two examples of period group definitions:

Periodic Group GA:

```
FNDEF='01,GA,PE'
FNDEF='02,A1,6,A,NU'
FNDEF='02,A2,2,B,NU'
FNDEF='02,A3,4,P,NU'
```

In this example, periodic group GA consists of fields A1, A2, and A3. The number of occurrences of the periodic group in a record is defined as a one- or two-byte binary value before each occurrence group in every record (depending on the setting of the **MUPECOUNT** parameter).

Periodic Group GB:

```
FNDEF='01,GB,PE(3)'
FNDEF='02,B1,4,A,DE,NU'
FNDEF='02,B2,5,A,MU(2),NU'
FNDEF='02,B3'
FNDEF='03,B4,20,A,NU'
FNDEF='03,B5,7,U,NU'
```

In this example, periodic group GB consists of fields B1, B2, and gropu B3 (which includes fields B4 and B5). Three (3) occurrences of the periodic group can occur in a record.

occurrences

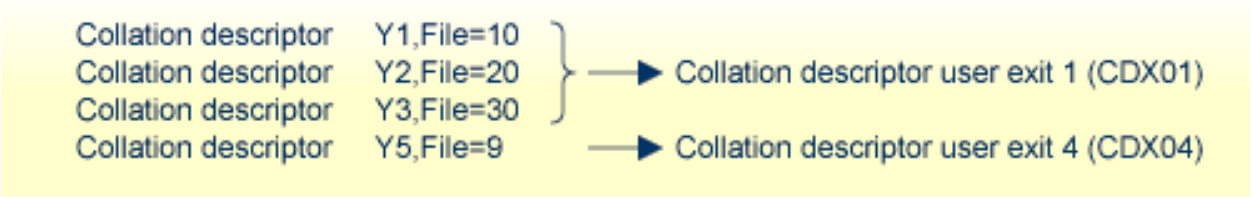
The number of occurrences of PE fields that will occur in a record if the MUPEX option is specified. This is an optional parameter.

COLDE: Collation Descriptor Definition

The collation descriptor option enables descriptor values to be sorted (collated) based on a user-supplied algorithm.

The values are based on algorithms coded in special collation descriptor user exits (CDX01 through CDX08). Each collation descriptor must be assigned to a user exit, and a single user exit may handle multiple collation descriptors.

Example:



The Collation Exit functions are called on the following events:

INITIALIZE function

- nucleus session start
- utility initialization when collation exits have been defined (ADARUN parameters)

ENCODE function

- update/insert/delete of the parent's value (Nucleus)
- Search specifying the collation descriptor with the search value (Nucleus)
- compression of a record (ADACMP)

DECODE function

- Read Index (L9) by Collation DE, only if the exit supports the DECODE function (Nucleus)

Input parameters supplied to the user exit are described in *Collation Descriptor Exits 01 - 08* in *Adabas User, Hyperdescriptor, and Collation Descriptor Exits Manual*. They include the:

- address and length of input string
- address and size of output area
- address of fullword for the returned output string length.

The user exit sets the length of the returned output string.

Read *CDXnn : Collation Descriptor User Exit* in *Adabas Operations Manual* for more information.

Notes:

1. A collation descriptor can be defined for an alphanumeric (A) or wide character (W) parent field. The format, length, and options (except UQ and XI) are taken from the parent field defined in the COLDE parameter. The unique descriptor (UQ) and exclude index (XI) options are separately defined for the collation descriptor itself.
2. A search using a collation descriptor value is performed in the same manner as for standard descriptors.

3. The user is responsible for creating correct collation descriptor values. There is no standard way to check the values of a collation descriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
4. If a file contains more than one collation descriptor, the assigned exits are called in the alphabetical order of the collation descriptor names.

Collation Descriptor Syntax

A collation descriptor is defined using the following syntax:

where:

number	is the user exit number to be assigned to the collation descriptor. The Adabas nucleus uses this number to determine the collation descriptor user exit to be called.
name	is the name to be used for the collation descriptor. The naming conventions for collation descriptors are identical to those for Adabas field names.
UQ	indicates that the unique descriptor option is to be assigned to the collation descriptor.
XI	indicates that the uniqueness of the collation descriptor is to be determined with the index (occurrence) number excluded.
parent-field	is the name of an elementary A or W field. A collation descriptor can have one parent field. The field name and address is passed to the user exit.

MU, NU, and PE options are taken from the parent field and are implicitly set in the collation descriptor.

If a parent field with the NU option is specified, no entries are made in the collation descriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other collation descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Collation Descriptor Definition Example:

Field definition:

```
FNDEF='01,LN,20,A,DE,NU'    Last-Name
```

Collation descriptor definition:

```
COLDE='1,Y2=LN'
```

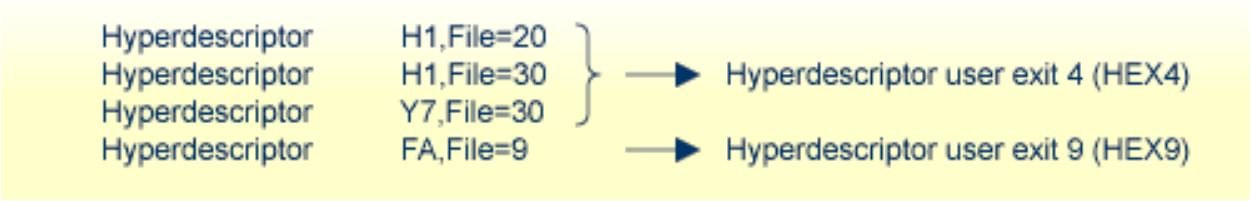
- Collation descriptor user exit 1 (CDX01) is assigned to this collation descriptor, and the name is Y2.
- The collation descriptor length and format are taken from the parent field: 20 and alphanumeric, respectively. The collation descriptor is a multiple value (MU) field with null suppression (NU).
- The values for the collation descriptor are to be derived from the parent field LN.

HYPDE: Hyperdescriptor Definition

The hyperdescriptor option enables descriptor values to be generated, based on a user-supplied algorithm.

The values are based on algorithms coded in special hyperdescriptor user exits (HEX01 through HEX31). Each hyperdescriptor must be assigned to a user exit, and a single user exit may handle multiple hyperdescriptors.

Example:



The exit is called whenever a hyperdescriptor value is to be generated by the Adabas nucleus or by the ADACMP utility.

Input parameters supplied to the user exit are:

- hyperdescriptor name
- file number
- addresses of fields taken from the Data Storage record, together with field name and PE index (if applicable). These addresses point to the compressed values of the fields. The names of these fields must be defined using the HYPDE parameter of ADACMP or ADAINV.

The user exit must return the descriptor value(s) (DVT) in compressed format. No value, or one or more values may be returned depending on the options (PE, MU) assigned to the hyperdescriptor.

The original ISN assigned to the input value(s) may be changed.

For complete information about hyperdescriptor user exits, read *Hyperdescriptor Exits 01 - 31* in *Adabas User, Hyperdescriptor, and Collation Descriptor Exits Manual*.

Notes:

1. The format, the length, and the options of a hyperdescriptor are user-defined. They are not taken from the parent fields defined in the HYPDE parameter.
2. A search using a hyperdescriptor value is performed in the same manner as for standard descriptors.
3. The user is responsible for creating correct hyperdescriptor values. There is no standard way to check the values of a hyperdescriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
4. If a hyperdescriptor is defined as packed or unpacked format, Adabas checks the returned values for validity. The sign half-byte for packed values can contain A, C, E, F (positive) or B, D (negative). Adabas converts the sign to F or D.
5. If a file contains more than one hyperdescriptor, the assigned exits are called in the alphabetical order of the hyperdescriptor names.

Hyperdescriptor Syntax

A hyperdescriptor is defined using the following syntax:

```
HYPDE = 'number, name, length, format [ { , option} ... ] = { parent-field } , ...'
```

where

number	is the user exit number to be assigned to the hyperdescriptor. The Adabas nucleus uses this number to determine the hyperdescriptor user exit to be called.
name	is the name to be used for the hyperdescriptor. The naming conventions for hyperdescriptors are identical to those for Adabas field names.
length	is the default length of the hyperdescriptor.
format	is the format of the hyperdescriptor: **Format** / **Maximum Length** Alphanumeric (A) / 253 bytes Binary (B) / 126 bytes Fixed Point (F) / 4 bytes (always 4 bytes) Floating Point (G) / 8 bytes (always 4 or 8 bytes) Packed Decimal (P) / 15 bytes Unpacked Decimal (U) / 29 bytes **Note:** Wide-character (W) format is not valid for a hyperdescriptor.
option	is an option to be assigned to the hyperdescriptor. The following options may be used together with a hyperdescriptor: ■ MU (multiple-value field) ■ NU (null-value suppression) ■ PE (field of a periodic group) ■ UQ (unique descriptor The parent field of a hyperdescriptor *cannot* be a long alphanumeric (LA) field.
parent-field	is the name of an elementary field. A hyperdescriptor can have 1-20 parent fields. The field names and addresses are passed to the user exit. **Note:** A hyperdescriptor parent-field may not have W (wide-character) format.

If a parent field with the NU option is specified, no entries are made in the hyperdescriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other hyperdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Hyperdescriptor Definition Example:

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'   Last-Name
FNDEF='01,FN,20,A,MU,NU'   First-Name
FNDEF='01,ID,4,B,NU'       Identification
FNDEF='01,AG,3,U'          Age
FNDEF='01,AD,PE'           Address
FNDEF='02,CI,20,A,NU'      City
FNDEF='02,ST,20,A,NU'      Street
FNDEF='01,FA,PE'           Relatives
FNDEF='02,NR,20,A,NU'      R-Last-Name
FNDEF='02,FR,20,A,MU,NU'   R-First-Name
```

Hyperdescriptor definition:

```
HYPDE='2,HN,60,A,MU,NU=LN,FN,FR'
```

- Hyperdescriptor user exit 2 is assigned to this hyperdescriptor, and the name is HN.
- The hyperdescriptor length is 60, the format is alphanumeric, and is a multiple-value (MU) field with null suppression (NU).
- The values for the hyperdescriptor are to be derived from fields LN, FN and FR.

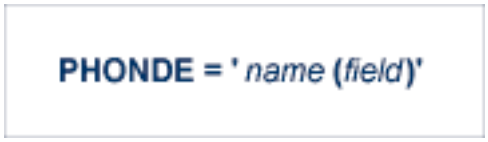
The ADACMP HYPDE= statement may be continued on another line, as shown in the following example. To do so, first specify a minus (-) after a whole argument and before the closing apostrophe on the first line. Then enter the remaining positional arguments, beginning after the statement name (ADACMP) enclosed in apostrophes on the following line:

```
ADACMP HYPDE='1,HY,20,A=AA,BB,CC,-'
ADACMP                'DD,EE,FF'
```

PHONDE: Phonetic Descriptor

The use of a phonetic descriptor in a FIND command results in the return of all the records that contain similar phonetic values. The phonetic value of a descriptor is based on the first 20 bytes of the field value. Only alphabetic values are considered; numeric values, special characters, and blanks are ignored. Lower- and uppercase alphanumeric characters are internally identical.

A phonetic descriptor is defined using the following syntax:



where

name	is the name to be used for the phonetic descriptor. The naming conventions for phonetic descriptors are identical to those for Adabas field names.
field	is the name of the field to be phoneticized.

The field *must* be

- an elementary or a multiple value field; and
- defined with alphanumeric format.

The field can be a descriptor.

The field *cannot* be

- a subdescriptor, superdescriptor, or hyperdescriptor;
- contained within a periodic group;
- used as the source field for more than one phonetic descriptor.
- format W (wide-character)

The parent field of a phonetic descriptor *cannot* be a long alphanumeric (LA) field.

If the field is defined with the NU option, no entries are made in the phonetic descriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the field.

If the field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Phonetic Descriptor Definition Example:

Field definition:

```
FNDEF='01,AA,20,A,DE,NU'
```

Phonetic definition:

```
PHONDE='PA(AA)'
```

SUBDE: Subdescriptor Definition

A subdescriptor is a descriptor created from a portion of an elementary field. The elementary field may or may not be a descriptor itself. A subdescriptor can also be used as a subfield; that is, it can be specified in the format buffer to control the record's output format.

A subdescriptor definition is entered using the following syntax:

```
SUBDE = 'name [, UQ [ XI ] ] = parent-field (begin , end)'
```

where

name	is the subdescriptor name. The naming conventions for a subdescriptor are identical to those for Adabas field names.
UQ	indicates that the subdescriptor is to be defined as unique (see the **definition of option UQ**).
XI	indicates that the uniqueness of the subdescriptor is to be determined with the index (occurrence) number excluded (see the **definition of option XI**).
parent-field	is the name of the field from which the subdescriptor is to be derived.
begin	is the relative byte position within the parent field where the subdescriptor definition is to begin.
end	is the relative byte position within the parent field where the subdescriptor definition is to end.

* *Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with P format, the sign of the resulting subdescriptor value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).*

A parent field of a subdescriptor can be

- a descriptor
- an elementary field
- a multiple-value field (but *not* a particular occurrence of a multiple-value field)
- contained within a periodic group (but *not* a particular occurrence of a periodic group)

A parent field or a subdescriptor *cannot* be

- a sub/super field, subdescriptor, superdescriptor, or phonetic descriptor
- format G (floating point)

- a long alphanumeric (LA) field.

If the parent field is defined with the NU option, no entries are made in the subdescriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the parent field.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Subdescriptor Definition Example 1:

Parent-field definition:

```
FNDEF='01,AR,10,A,NU'
```

Subdescriptor definition:

```
SUBDE='SB=AR(1,5)'
```

The values for subdescriptor SB are derived from the first five bytes (counting from left to right) of all the values for the parent field AR. All values are shown in character format.

AR Values	SB Values
DAVENPORT	DAVEN
FORD	FORD
WILSON	WILSO

Subdescriptor Definition Example 2:

Parent-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PS=PF(4,6)'
```

The values for subdescriptor PS are derived from bytes 4 to 6 (counting from right to left) of all the values for the parent field PF. All values are shown in hexadecimal.

PF Values	PS Values
00243182655F	02431F
00000000186F	0F (see note)
78426281448D	0784262D

Note: If the NU option had been specified for parent field PF, no value would have been created for PS for this value.

Subdescriptor Definition Example 3:

Source-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PT=PF(1,3)'
```

The values for PT are derived from bytes 1 to 3 (counting from right to left) of all the values for PF. All values are shown in hexadecimal.

PF Values	PT Values
00243182655F	82655F
00000000186F	186F
78426281448D	81448D

SUBFN: Subfield Definition

A subfield:

- is a portion of an elementary field that can be read using an Adabas read command;
- cannot be updated;
- can be changed to a subdescriptor using ADAINV INVERT SUBDE=... .

A subfield definition is entered using the following syntax:

SUBFN = ' *name* **=** *parent-field* **(** *begin* **,** *end* **)'**

where

name	is the subfield name. The naming conventions for a subfield are identical to those for Adabas field names.
parent-field	is the name of the field from which the subfield is to be derived.
*begin**	is the relative byte position within the parent field where the subfield definition is to begin.
*end**	is the relative byte position within the parent field where the subfield definition is to end.

** Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with "P" format, the sign of the resulting subfield value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).*

The parent field for a subfield can be:

- a multiple-value field
- within a periodic group

The parent field for a subfield *cannot*:

- have format "G" (floating point)
- be a long alphanumeric (LA) field.

Subfield Definition Example:

```
SUBFN='X1=AA(1,2)'
```

SUPDE: Superdescriptor Definition

A superdescriptor is a descriptor created from several fields, portions of fields, or a combination thereof.

Each source field (or portion of a field) used to define a superdescriptor is called a *parent* . From 2 to 20 parent fields or field portions may be used to define a superdiscriptor. The total size must be less than or equal to 253.

A superdescriptor may be defined as a unique descriptor.

A superdescriptor can be used as a superfield; that is, it can be specified in the format buffer to determine the record's output format.

This section covers the following topics:

- SUPDE Syntax
- Superdescriptor Interfaces with Adabas Commands
- Format Conversions of Superdescriptors
- SUPDE Examples

SUPDE Syntax

A superdescriptor definition has the following syntax:

```
SUPDE = 'name [, UQ [ XI ] ] = { parent-field (begin , end) } , ...'
```

where

name	is the superdescriptor name. The naming conventions for superdescriptors are identical to those for Adabas names.
UQ	indicates that the superdescriptor is to be defined as unique (see the **definition option UQ**).
XI	indicates that the uniqueness of the superdescriptor is to be determined with the index (occurrence) number excluded (see the **definition option XI**).
parent-field	is the name of a parent field from which a superdescriptor element is to be derived; up to 20 parent fields can be specified.
*begin**	is the relative byte position within the field where the superdescriptor element begins.
*end**	is the relative byte position within the field where the superdescriptor element is to end.

** Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format. For any parent field except those defined as "FI", any begin and end values within the range permitted for the parent field's data type are valid.*

A parent field of a superdescriptor can be:

- an elementary field
- a maximum of one MU field (but not a specific MU field value)
- within a periodic group (but not a specific occurrence)
- a descriptor.

A parent field of a superdescriptor *cannot* be

- a super-, sub-, or phonetic descriptor;
- format G (floating point);
- an NC option field if another parent field is an NU option field;
- a long alphanumeric (LA) field.

If a parent field with the NC or NU option is specified, no entries are made in the superdescriptor's inverted list for those records containing a null value for the field. In other words, no value is created if the parent value is empty and the NC/NU option has been specified. This is true regardless of the presence or absence of values for other superdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

The total length of any superdescriptor value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superdescriptor format is B (binary) if no element of the superdescriptor is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superdescriptor is derived from an A or W parent field, the format of the superdescriptor reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superdescriptor is W.

All binary format superdescriptor values are treated as unsigned numbers.

The ADACMP SUPDE= statement may be continued on another line by specifying a minus (-) after an argument just before the closing apostrophe on the first line. Then enter the remaining positional arguments enclosed in apostrophes on the following line beginning after the statement name (ADACMP). For example:

```
ADACMP SUPDE='SI=AA(10,20),BB(20,21),-'
ADACMP       'CC(12,13),DD(14,15)'
```

Superdescriptor Interfaces with Adabas Commands

The following commands can interface with superdescriptors.

Adabas Command	Superdescriptor values can be...
N1 or A1	implicitly built with given fields at insert and update, when parent fields force the creation of superdescriptors
S*x* or L3	specified in the value buffer of search expressions and logical reads.
L9	returned in the record buffer.

Format Conversions of Superdescriptors

Superdescriptors have a final superdescriptor format which is calculated as follows:

Superdescriptor Format	Used if:
A (alphabetic)	At least one parent field has format A.
B (binary)	All other superdescriptor fields.

This section covers the following topics:

- Format Conversions During Updates
- Format Conversions In Value Buffers
- Format Conversions For Output (L9 Command)

Note: Conversions are only performed if a parent field has not been set with the NV option.

Format Conversions During Updates

Superdescriptors should be built so that they have the same collating sequence in all environments. However, problems exist for some combinations, as described in this section:

- Alphanumeric (Format A) Values in IBM (EBCDIC) and UNIX (ASCII) Environments
- Numeric Values (Format U) in IBM (EBCDIC) and UNIX (ASCII) Environments
- Binary Values (Formats B, F, G) with Big-endian and Little-endian Storage Formats
- Different Packed Value Signs (Format P)

Alphanumeric (Format A) Values in IBM (EBCDIC) and UNIX (ASCII) Environments

All alphabetic field values will be converted from EBCDIC to ASCII if an insert or update call comes from an IBM mainframe environment to a UNIX database. Consequently, the superdescriptor parent values are automatically converted to ASCII. In this case, an application might fail if it expects a specific sort sequence (for example using uppercase and lowercase characters). In EBCDIC formats, lowercase characters come prior to uppercase characters; in ASCII formats, this sequence is reversed (uppercase characters come prior to lowercase characters).

One of two methods can be used to resolve this problem:

- Use the NV option on parent fields with EBCDIC-ASCII conflicts. This will disable the EBCDIC-ASCII conversion.
- Use a hyperdescriptor instead of a superdescriptor.

Numeric Values (Format U) in IBM (EBCDIC) and UNIX (ASCII) Environments

All numeric field values will be converted from EBCDIC to ASCII if an insert or update call comes from an IBM mainframe environment to a UNIX database. Consequently, the superdescriptor parent values are automatically converted to ASCII, even if the final superdescriptor requests formats of A (alphabetic) or B (binary).

Binary Values (Formats B, F, G) with Big-endian and Little-endian Storage Formats

Some platforms store binary byte sequences in big-endian format; others store them in little-endian format. For example, IBM and HP-UX processors use big-endian format (the byte significance runs from right to left), while Intel processors use little-endian sequence (the byte significance runs from left to right).

Adabas performs conversions on superdescriptors containing binary values in swapped architectures (little-endian binary values with significance running from left to right) to get them into a standard sort sequence before storing them in the index.

- For alphabetic superdescriptors containing at least one binary field with parent lengths greater than one, the binary parent values will be swapped.
- For binary superdescriptors, the order of the parent entries will be swapped and the non-binary parent values will be swapped.

Different Packed Value Signs (Format P)

Sign information of packed values is represented differently on different platforms. Adabas on open systems converts positive values (A, C, or F) to C and negative values (B or D) to D. Adabas for mainframes uses F to represent positive values. Consequently, collating sequence problems arise if packed values are used in superdescriptors because the packed value signs lose their meaning; they become normal bit patterns. When this happens, positive packed values can be sorted as negative packed values.

In addition, when combined in a superdescriptor in Adabas for mainframes, negative packed values are sorted before positive packed values, while on Adabas for open systems, positive packed values are sorted before negative packed values.

To resolve these problems, we recommend that you use a hyperdescriptor instead of a superdescriptor.

Format Conversions In Value Buffers

When superdescriptors are specified in a value buffer, they are converted so they can be matched to an associated index entry.

Format Conversions For Output (L9 Command)

Superdescriptor values retrieved by L9 commands must be converted before they are returned in the record buffer. Alphabetic fields are converted from ASCII to EBCDIC, if required. In addition, binary parts of the superdescriptor are swapped if necessary. The packed signs of packed value parts of the superdescriptor are not converted.

SUPDE Examples

- Superdescriptor Definition Example 1
- Superdescriptor Definition Example 2
- Superdescriptor Definition Example 3
- Superdescriptor Definition Example 4
- Superdescriptor Definition Example 5

Superdescriptor Definition Example 1

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'   Last-Name
FNDEF='01,FN,20,A,MU,NU'   First-Name
FNDEF='01,ID,4,B,NU'       Identification
FNDEF='01,AG,3,U'          Age
FNDEF='01,AD,PE'           Address
FNDEF='02,CI,20,A,NU'      City
FNDEF='02,ST,20,A,NU'      Street
FNDEF='01,FA,PE'           Relatives
FNDEF='02,NR,20,A,NU'      R-Last-Name
FNDEF='02,FR,20,A,MU,NU'   R-First-Name
```

Superdescriptor definition:

```
SUPDE='SD=LN(1,4),ID(3,4),AG(2,3)'
```

Superdescriptor SD is to be created. The values for the superdescriptor are to be derived from bytes 1 to 4 of field LN (counting from left to right), bytes 3 to 4 of field ID (counting from right to left), and bytes 2 to 3 of field AG (counting from right to left). All values are shown in hexadecimal.

LN	ID	AG	SD
C6D3C5D4C9D5C7	00862143	F0F4F3	C6D3C5D40086F0F4
D4D6D9D9C9E2	02461866	F0F3F8	D4D6D9D90246F0F3
D7C1D9D2C5D9	00000000	F0F3F6	No value is stored (because of ID)
404040404040	00432144	F0F0F0	No value is stored (because of LN)
C1C1C1C1C1C1	00000144	F1F1F1	C1C1C1C10000F1F1
C1C1C1C1C1C1	00860000	F0F0F0	C1C1C1C10086F0F0

The format for SD is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 2

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'   Last-Name
FNDEF='01,FN,20,A,MU,NU'   First-Name
FNDEF='01,ID,4,B,NU'       Identification
FNDEF='01,AG,3,U'          Age
FNDEF='01,AD,PE'           Address
FNDEF='02,CI,20,A,NU'      City
FNDEF='02,ST,20,A,NU'      Street
FNDEF='01,FA,PE'           Relatives
FNDEF='02,NR,20,A,NU'      R-Last-Name
FNDEF='02,FR,20,A,MU,NU'   R-First-Name
```

Superdescriptor definition:

```
SUPDE='SY=LN(1,4),FN(1,1)'
```

Superdescriptor SY is to be created from fields LN and FN (which is a multiple-value field). All values are shown in character format.

LN	FN	SY
FLEMING	DAVID	FLEMD
MORRIS	RONALD RON	MORRR MORRR
WILSON	JOHN SONNY	WILSJ WILSS

The format of SY is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 3

Field definitions:

```
FNDEF='01,PN,6,U,NU'
FNDEF='01,NA,20,A,DE,NU'
FNDEF='01,DP,1,B,FI '
```

Superdescriptor definition:

```
SUPDE='SZ=PN(3,6),DP(1,1)'
```

Superdescriptor SZ is to be created. The values for the superdescriptor are to be derived from bytes 3 to 6 of field PN (counting from right to left), and byte 1 of field DP. All values are shown in hexadecimal.

PN	DP	SZ
F0F2F4F6F7F2	04	F0F2F4F604
F8F4F0F3F9F8	00	F8F4F0F300
F0F0F0F0F1F1	06	F0F0F0F006
F0F0F0F0F0F1	00	F0F0F0F000
F0F0F0F0F0F0	00	no value is stored (because of PN)
F0F0F0F0F0F0	01	no value is stored (because of PN)

The format of SZ is binary since no element is derived from a parent field defined with alphanumeric format. A null value is not stored for the last two values shown because the superdescriptor option is NU (from the PN field) and the PN field value contains unpacked zeros (X'F0'), the null value.

Superdescriptor Definition Example 4

Field definitions:

```
FNDEF='01,PF,4,P,NU'
FNDEF='01,PN,2,P,NU'
```

Superdescriptor definition:

```
SUPDE='SP=PF(3,4),PN(1,2)'
```

Superdescriptor SP is to be created. The values for the superdescriptor are to be derived from bytes 3 to 4 of field PF (counting from right to left), and bytes 1 to 2 of field PN (counting from right to left). All values are shown in hexadecimal.

PF	PN	SP
0002463F	003F	0002003F
0000045F	043F	0000043F
0032464F	000F	No value is stored (because of PN)
0038000F	044F	0038044F

The format of SP is binary since no element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 5

Field definitions:

```
FNDEF='01,AD,PE'
FNDEF='02,CI,4,A,NU'
FNDEF='02,ST,5,A,NU'
```

Superdescriptor definition:

```
SUPDE='XY=CI(1,4),ST(1,5)'
```

Superdescriptor XY is to be created from fields CI and ST. All values are shown in character format.

CI	ST	XY
(1st occ.) BALT	(1st occ.) MAIN	BALTMAIN
(2nd occ.) CHI	(2nd occ.) SPRUCE	CHI SPRUC
(3rd occ.) WASH	(3rd occ.) 11TH	WASH11TH
(4th occ.) DENV	(4th occ.) bbbbb	No value stored (because of ST)

The format of XY is alphanumeric since at least 1 element is derived from a parent field which is defined with alphanumeric format.

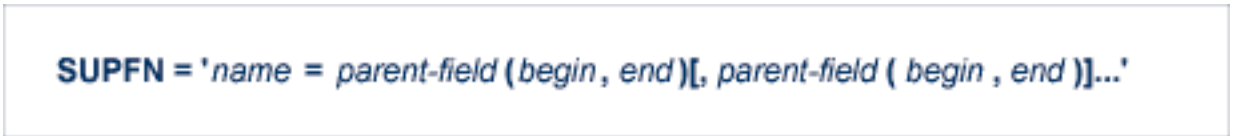
SUPFN: Superfield Definition

A superfield is a field composed of several fields, portions of fields, or combinations thereof, which may be read using an Adabas read command. A superfield *cannot*

- be updated;
- comprise fields defined with the NC option if another parent field has the NU option;
- be used as a descriptor.

A superfield *can* be changed to a superdescriptor using the ADAINV utility function INVERT SUPDE=....

A superfield is defined using the following syntax:



where

name	superfield name. The naming conventions for superfields are identical to those for Adabas names.
parent-field	name of the field from which a superfield element is to be derived.
*begin**	relative byte position within the field where the superfield element is to begin.
*end**	relative byte position within the field where the superfield element is to end.

** Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format.*

A parent field of a superfield can be:

- a multiple-value field
- contained within a periodic group

A parent field of a superfield *cannot*:

- have format "G" (floating point)
- be a long alphanumeric (LA) field.

The total length of any superfield value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superfield format is B (binary) if no element of the superfield is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superfield is derived from an A or W parent field, the format of the superfield reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superfield is W.

Superfield Definition Example:

```
SUPFN='X2=AA(1,2),AB(1,4),AC(1,1)'
```

17 JCL/JCS Requirements and Examples

- User Exits with ADACMP 146
- BS2000 147
- z/OS 150
- z/VM 155
- VSE 157

This section describes the job control information required to run ADACMP with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Note: When the recovery log is active, sequential data sets used by the utilities whose runs are logged on the RLOG must be kept and made available for any recovery operation; for example, the DD/EBAND input to an ADALOD LOAD operation.

User Exits with ADACMP

Compression with User Exit

User exit 6 can be used to perform user processing on a record before it is processed by the ADACMP COMPRESS utility. It can also be used to control the sequence and contents of the de-compressed records that are output from the ADACMP DECOMPRESS utility; when used in this way, the user exit controls which decompressed records ADACMP writes to the DDAUSBA data set. For more information about user exit 6, read *User Exit 6 (User Processing Before Data Compression)* in *Adabas Utilities Manual*.

If user exit 6 is to be used during ADACMP execution, the specified user exit routine must be loadable at execution time; that is, it must be assembled and linked into the Adabas

- load library (or any library concatenated with it) for BS2000, z/OS, z/VM.
- core image library or any library contained in the core image library search chain for VSE.

The ADACMP COMPRESS utility job must specify:

```
ADARUN UEX6 = exit-name
```

where:

exit-name	is the name of a user routine that gets control at the user exit; the name can be up to 8 characters long.

For more information, read *UEXn : User Exit* in *Adabas Operations Manual*.

Collation with User Exit

If a collation user exit is to be used during ADACMP execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is:

```
ADARUN CDXnn= exit-name
```

where:

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). For more information, read *CDXnn : Collation Descriptor User Exit* in *Adabas Operations Manual*.

BS2000

Data Set	Link Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/ disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/ disk	Not used if the parameter INFILE is used
Compressed data for a data base with files containing large object (LB) fields (COMPRESS function)	DDAUSB1	tape/disk	This additional data set receives the compressed large object records to be loaded into the *LOB file* as the compressed records in the first output data set (DDAUSBA) are loaded into the *base file*.
Compressed data (COMPRESS function)	DDAUSBA	tape/ disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/ disk	

Data Set	Link Name	Storage	More Information
Rejected data	DDFEHL	tape/ disk	
ECS encoding objects	DDECSOJ	tape/ disk	Required for universal encoding support (UES)
ADARUN parameters	SYSDTA/ DDCARD		*Adabas Operations Manual*
ADACMP parameters and data definitions	SYSDTA/ DDKARTE		*Adabas Utilities Manual*
ADARUN messages	SYSOUT/ DDPRINT	printer/ disk	*Adabas Messages and Codes Manual*
ADACMP report	SYSLST/ DDDRUCK	printer/ disk	*Adabas Messages and Codes Manual*

JCL Examples (BS2000)

ADACMP COMPRESS

In SDF Format:

```
/.ADACMP LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C M P COMPRESS
/REMARK *
/DELETE-FILE CMP.AUS
/SET-JOB-STEP
/DELETE-FILE CMP.FEHL
/SET-JOB-STEP
/CREATE-FILE CMP.AUS,PUB(SPACE=(48,48)
/SET-JOB-STEP
/CREATE-FILE CMP.FEHL,PUB(SPACE=(48,48))
/SET-JOB-STEP

/ASS-SYSLST L.CMP
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDEBAND,CMP.EIN
/SET-FILE-LINK DDAUSBA,CMP.AUS
/SET-FILE-LINK DDFEHL,CMP.FEHL
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP COMPRESS NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACMP LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C M P COMPRESS
/REMARK *
/ER CMP.AUS
/STEP
/ER CMP.FEHL
/STEP
/SYSFILE SYSLST=L.CMP
/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.EIN,LINK=DDEBAND
/FILE CMP.AUS,LINK=DDAUSBA,SPACE=(48,48)
/FILE CMP.FEHL,LINK=DDFEHL,SPACE=(48,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP COMPRESS NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
/LOGOFF NOSPOOL
```

ADACMP DECOMPRESS

In SDF Format:

```
/.ADACMP LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C M P DECOMPRESS
/REMARK *

/DELETE-FILE CMP.AUS
/SET-JOB-STEP
/DELETE-FILE CMP.FEHL
/SET-JOB-STEP
/CREATE-FILE CMP.AUS,PUB(SPACE=(48,48))
/SET-JOB-STEP
/CREATE-FILE CMP.FEHL,PUB(SPACE=(48,48))
/SET-JOB-STEP
/ASS-SYSLST L.DEC
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDEBAND,CMP.EIN
/SET-FILE-LINK DDAUSBA,CMP.AUS
/SET-FILE-LINK DDFEHL,CMP.FEHL
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP DECOMPRESS
```

```
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACMP LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C M P DECOMPRESS
/REMARK *
/ER CMP.AUS
/STEP
/ER CMP.FEHL
/STEP
/SYSFILE SYSLST=L.CMP.DEC

/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.EIN,LINK=DDEBAND
/FILE CMP.AUS,LINK=DDAUSBA,SPACE=(48,48)
/FILE CMP.FEHL,LINK=DDFEHL,SPACE=(48,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP DECOMPRESS
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/ disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/ disk	Not used if the parameter INFILE is specified
Compressed data for a data base with files containing large object (LB) fields (COMPRESS function)	DDAUSB1	tape/disk	This additional data set receives the compressed large object records to be loaded into the *LOB file* as the compressed records in the first output data set (DDAUSBA) are loaded into the *base file*
Compressed data (COMPRESS function)	DDAUSBA	tape/ disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/ disk	
Rejected data	DDFEHL	tape/ disk	

Data Set	DD Name	Storage	More Information
ECS encoding objects	DDECSOJ	tape/ disk	Required for universal encoding support (UES)
ADACMP report	DDDRUCK	printer	
ADARUN messages	DDPRINT	printer	
ADARUN parameters	DDCARD	reader	
ADACMP parameters and data definitions	DDKARTE	reader	

JCL Examples (z/OS)

In the JOBS data set, refer to ADACMP and ADACMPS for the COMPRESS examples and ADACMPD for the DECOMPRESS example.

ADACMP COMPRESS

This example can be found in the ADACMP member of the JOBS data set.

```
//ADACMP    JOB
//*
//*   ADACMP COMPRESS
//*      COMPRESS A FILE
//*
//CMP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*

//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.INPUT,UNIT=TAPE, <===
//          VOL=SER=TAPE01                                     <===
//DDAUSBA   DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.COMP01,UNIT=DISK, <==
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,UNIT=DISK, <===
//          VOL=SER=DISK01,SPACE=(TRK,1)
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACMP COMPRESS FILE=1
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
```

```
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'
ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'

ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
/*
```

The following example can be found in member ADACMPS of the JOBS data set. This example shows the use of the ADACMP spanned record parameters and extended MU/PE limit parameters.

```
//ADACMP    JOB
//*
//*   ADACMP COMPRESS
//*      COMPRESS A FILE
//*         USING MUPEX AND SPAN OPTIONS WITH 2-BYTE MU/PE COUNTS
//*
//CMP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.INPUT,UNIT=TAPE, <===
//          VOL=SER=TAPE01                                    <===
//DDAUSBA   DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.COMP01,UNIT=DISK, <
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,UNIT=DISK, <==
//          VOL=SER=DISK01,SPACE=(TRK,1)
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=3390,DBID=YYYYY
/*
//DDKARTE   DD   *
ADACMP COMPRESS FILE=1
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
```

```
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'
ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'
ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP FNDEF='01,MA,010,A,MU,NU,DE'
ADACMP FNDEF='01,PB,PE'
ADACMP FNDEF='02,P1,008,A,NU'
ADACMP FNDEF='02,P2,002,A,NU'
ADACMP FNDEF='02,P3,020,A,NU'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
ADACMP MUPEX                        <== EXTENDED MU/PE FILE
ADACMP MUPECOUNT=2                  <== 2-BYTE MU/PE COUNTS IN INPUT
ADACMP SPAN                         <== SPANNED RECORD FILE
ADACMP DATADEVICE=3390              <== DATA STORAGE DEVICE TYPE
/*
//   ↩
```

ADACMP DECOMPRESS

The following example can be found in member ADACMPD of the JOBS data set.

```
//ADACMP    JOB
//*
//*   ADACMP COMPRESS
//*      DECOMPRESS A FILE
//*
//DECMP     EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <===DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <===WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
```

```
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.COMP01,UNIT=TAPE,
//          VOL=SER=TAPE01
//DDAUSBA   DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.DECOMP01,UNIT=DISK,
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,UNIT=DISK,
//          VOL=SER=DISK01,SPACE=(TRK,1)
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACMP DECOMPRESS INFILE=1
/*
```

Using ADACMP with UES Parameters, Wide Field Formats, or Collation Descriptions

The following compression example can be found in member ADACMPU of the JOBS data set. It can be used as a basis for compression jobs that make use of UES parameters, wide-character field formats, or collation descriptors.

```
//ADACMPU   JOB
//*
//*   ADACMP COMPRESS
//*      COMPRESS A FILE
//*         USING UES FEATURES
//*
//* Please update the JCL for current version/release/smlevel numbers
//* for the ADABAS (ADAvrs) and Software AG internal (APSvrs) libraries, for     ↩

//* changed dataset prefixes or when using a single composite
//* load library.
//*
//CMP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD  <=== ADABAS LOAD
//          DD   DISP=SHR,DSN=ADABAS.APSvrs.LDnn  <=== APS LOAD UPD
//          DD   DISP=SHR,DSN=ADABAS.APSvrs.LD00  <=== APS LOAD BASE
//*
//CONFIG    DD  DUMMY                             <--  EXTRA DD FOR UES
//DDECSOJ   DD  DISP=SHR,DSN=ADABAS.ADAvrs.ECS00  <--  EXTRA DD FOR UES
//*
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.INPUT,UNIT=TAPE, <===
//          VOL=SER=TAPE01                                     <===
//DDAUSBA   DD   DSN=EXAMPLE.DByyyyy.COMP01,UNIT=DISK,         <===
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE),DISP=(NEW,KEEP)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,     <===
//          VOL=SER=DISK01,SPACE=(TRK,1),UNIT=DISK
```

```
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=3390,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACMP COMPRESS FILE=1
ADACMP FACODE=273
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'
ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'
ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
/*
//
```

z/VM

Data Set	DD Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/ disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/ disk	Not used if the parameter INFILE is specified
Compressed data for a data base with files containing large object (LB) fields (COMPRESS function)	DDAUSB1	tape/disk	This additional data set receives the compressed large object records to be loaded into the *LOB file* as the compressed records in the first output data

Data Set	DD Name	Storage	More Information
			set (DDAUSBA) are loaded into the *base file*
Compressed data (COMPRESS function)	DDAUSBA	tape/ disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/ disk	
Rejected data	DDFEHL	tape/ disk	
ECS encoding objects	DDECSOJ	tape/ disk	Required for universal encoding support
ADACMP report	DDDRUCK	disk/ terminal/ printer	
ADARUN messages	DDPRINT	disk/ terminal/ printer	
ADARUN parameters	DDCARD	disk/ terminal/ reader	
ADACMP control cards and data definitions	DDKARTE	disk/ terminal/ reader	

JCL Example (z/VM)

ADACMP COMPRESS

```
DATADEF DDEBAND,DSN=FILE015.CMPD015,MODE=A
DATADEF DDAUSBA,DSN=FILE015.LODD015,MODE=A
DATADEF DDFEHL,DSN=FILE015.CMPERROR,MODE=A
DATADEF DDDRUCK,DSN=ADACMP.DDDRUCK,MODE=A
DATADEF DDPRINT,DSN=ADACMP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDCARD,DSN=RUNCMP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=FILE001.CMPC015,MODE=A
ADARUN
```

Contents of RUNCMP CONTROL A1:

```
ADARUN PROG=ADACMP,DEVICE=dddd,DB=yyyyy
```

Contents of FILE001 CMPC015 A1:

```
ADACMP COMPRESS
NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
```

VSE

File	File Name	Storage	Logical Unit	More Information
User input data (COMPRESS function)	EBAND	tape disk	SYS010 *	
Compressed data (DECOMPRESS function)	EBAND	tape disk	SYS010 *	Not used if parameter INFILE is specified
Compressed data for a data base with files containing large object (LB) fields (COMPRESS function)	AUSB1	tape disk		This additional data set receives the compressed large object records to be loaded into the *LOB file* as the compressed records in the first output data set (AUSBA) are loaded into the *base file*
Compressed data (COMPRESS function)	AUSBA	tape disk	SYS012 *	
Decompressed data (DECOMPRESS function)	AUSBA	tape disk	SYS012 *	
Rejected data	FEHL	tape disk	SYS017 *	
ECS encoding objects	ECSOJ	tape disk	SYS020 *	Required for universal encoding support (UES)
ADACMP report	-	printer	SYS009	
ADARUN messages	-	printer	SYSLST	
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADACMP control cards and data definitions	-	reader	SYSIPT	

* *Any programmer logical unit may be used.*

JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures.

Refer to member ADACMP.X for the COMPRESS example and member ADACMPD.X for the DECOMPRESS example.

ADACMP COMPRESS

```
* $$ JOB JNM=ADACMP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       COMPRESS A FILE
// JOB ADACMP
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'EXAMPLE.DByyyyy.UNCOMP01'
// MTC REW,SYS010
// DLBL AUSBA,'EXAMPLE.DByyyyy.COMP01',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK01,SHR
// DLBL FEHL,'EXAMPLE.DByyy.FEHL',,SD
// EXTENT SYS017,,,,sssss,nnnnn
// ASSGN SYS017,DISK,VOL=DISK02,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADACMP COMPRESS FILE=1
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'

ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'
```

```
ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
/*
/&
* $$ EOJ
```

ADACMP DECOMPRESS

```
* $$ JOB JNM=ADACMPD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       DECOMPRESS A FILE
// JOB ADACMPD
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'EXAMPLE.DByyyyy.COMP01'
// MTC REW,SYS010
// DLBL AUSBA,'EXAMPLE.DByyyyy.DECOMP01',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK01,SHR
// DLBL FEHL,'EXAMPLE.DByyy.FEHL',,SD
// EXTENT SYS017,,,,sssss,nnnnn
// ASSGN SYS017,DISK,VOL=DISK02,SHR
*
* ********************************************************************
*     REMEMBER TO CUSTOMIZE PARAMETERS OF ADABAS UTILITY
* ********************************************************************
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADACMP DECOMPRESS INFILE=1
/*
/&
* $$ EOJ
```

18 COMPRESS Function Output

- Compressed Data Records 162
- Rejected Data Records 162
- ADACMP Report 168

This chapter describes the ADACMP COMPRESS function output.

Compressed Data Records

The data records that ADACMP has processed, edited, and compressed are written out together with the file definition information to a sequential data set with the variable blocked record format. This data set may be used as input to the ADALOD utility. The output of several ADACMP executions may also be used as input to ADALOD.

If the output data set contains no records (no records provided on the input data set or all records rejected), the output may still be used as input to the ADALOD utility. In this case, you must ensure that the amount of Associator space allocated to the file is sufficient since an accurate estimate cannot be made by the ADALOD utility without a representative sample of input record values (see the **ADALOD utility** for additional information).

For information on how to identify MU and PE occurrences greater than 191 in the compressed record, read ***Identifying MU and PE Occurrences Greater Than 191 in Compressed Records***, elsewhere in this section.

Rejected Data Records

Any records rejected during ADACMP compression are written to the DD/FEHL error data set. The records are output in variable blocked format and may be segmented into multiple physical records. Each logical rejected record will be preceded by an initial ADAF rejected record header. If the logical record and the ADAF header do not fit in the DD/FEHL physical record length, ADAN rejected record headers will precede the remaining physical rejected record segments that comprise the logical rejected record.

The functions of these two different headers are as follows:

- ADAF headers indicate the error condition and pertinent information.
- ADAN headers are smaller and are used for rejected record continuation and ADAH/ADAC header error reporting.

DSECTs for the ADAF and ADAN headers can be found in members ADAF and ADAN of the distributed Adabas SRCE data set.

Traditionally, the DD/FEHL error data set produced for ADACMP errors has truncated rejected records that exceeded the FEHL physical record length. In Version 8, the rejected records are segmented instead of truncated. Because of this change, the DD/FEHL LRECL setting must be at least 500 bytes.

If HEADER=YES is specified, an error may occur while segmented uncompressed records are being assembled into a logical record. If the ADAH header is in error, the ADAH record is written and subsequent ADAC records are not written until the next ADAH record is processed. If an ADAC header is in error, the preceding ADAH header will be written without its payload data. The ADAC record in error will be written in its entirety. Subsequent ADAC records are not written until the next ADAH record is processed.

The following response codes may occur:

X'E7'(231)	Input record too short (COMPRESS)
X'E8'(232)	Output record length error (COMPRESS)
X'E9'(233)	An invalid ADAH spanned record header has been encountered. The following subcodes provide more detail: ■ 1 - incorrect ADAH eye-catcher ■ 2 - incorrect ADAH header length ■ 3 - unexpected continuation indicator ■ 4 - reserved area not set to binary zeros ■ 5 - invalid segment length ■ 6 - total payload data length exceeds MAXLOGRECLEN setting For complete information about spanned records and the ADAH header, read *Spanned Records* in *Adabas Concepts and Facilities Manual*. The DSECT for the ADAH header can be found in member ADAH of the distributed Adabas SRCE data set.
X'EA'(234)	An invalid ADAC spanned record header has been encountered. The following subcodes provide more detail: ■ 1 - incorrect ADAC eye-catcher ■ 2 - incorrect ADAC header length ■ 3 - unexpected continuation indicator ■ 4 - reserved area not set to binary zeros ■ 5 - invalid segment length ■ 6 - unexpected continuation record sequence number ■ 7 - invalid segment offset ■ 8 - accumulated payload data length exceeds specified total length in ADAH ■ 9 - accumulated payload data length exceeds MAXLOGRECLEN setting For complete information about spanned records and the ADAC header, read *Spanned Records* in *Adabas Concepts and Facilities Manual*. The DSECT for the ADAC header can be found in member ADAC of the distributed Adabas SRCE data set.

Only the first incorrect field within a record is detected and referenced. If there are other errors, they are not detected until subsequent runs are made.

Example of Rejected Data Records

The following table depicts the FEHL output for four rejected records during ADACMP compression. Rejected records 1 and 3 have only one FEHL record (ADAF); rejected record 2 is segmented into two FEHL records (ADAF and ADAN); rejected record 4 is segmented into three FEHL records (one ADAF record and two ADAN records):

Note: DSECTs for the ADAF and ADAN headers can be found in members ADAF and ADAN of the distributed Adabas SRCE data set.

Rejected Record	FEHL Records	FEHL Fields		Description
		Field	Value	
1	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	R	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	E	Continuation indicator. Valid values are: C: Continuation record to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	22000	Segment length (length of payload data following the header)
		ADAFTOTL	22000	Total length. **Note:** The sum of the values of all ADANSLEN fields and ADAFSLEN should equal the value of ADAFTOTL (for record 1 in this example, 0 + 22000 = 22000)
		ADAFISN	1	ISN of record
		ADAFLNUM	1	Logical record number
		ADAFPNUM	1	Physical record number
		ADAFEOFF	5000	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN	ZA	Field name
		ADAFRSP	41	Response code
		ADAFSUB	2	Subcode
		Reserved	0	Reserved
		ADAFDATA	'Record 1 -- Payload Data'	Rejected input data

Rejected Record	FEHL Records	FEHL Fields		Description
		Field	Value	
2	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	R	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	C	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	27962	Segment length (length of payload data following the header)
		ADAFTOTL	50000	Total length. **Note:** The sum of the values of all ADANSLEN fields and ADAFSLEN should equal the value of ADAFTOTL (for record 2 in this example, 22038 + 27962 = 50000)
		ADAFISN	2	ISN of record
		ADAFLNUM	2	Logical record number
		ADAFPNUM	3	Physical record number
		ADAFEOFF	35000	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN	ZA	Field name
		ADAFRSP	41	Response code
		ADAFSUB	2	Subcode
		Reserved	0	Reserved
		ADAFDATA	'Record 2 -- Payload data part 1'	Rejected input data
	ADAN	ADANEYE	ADAN	ADAN header eye-catcher
		ADANLEN	24	ADAN header length
		ADANTYPE	R	Type. Valid values are: C: ADAC header D: ADAC header and payload P: ADAH record segment R: Logical record segment

Rejected Record	FEHL Records	FEHL Fields		Description
		Field	Value	
		ADANIND	E	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADANSLEN	22038	Segment length (length of payload data following the header)
		ADANOFF	27962	Error offset in logical record.
		ADANDATA	'Record 2 -- Payload data part 2'	Continued rejected input data
3	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	P	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	E	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	20000	Segment length (length of payload data following the header)
		ADAFTOTL	20000	Total length. **Note:** The sum of the values of all ADANSLEN fields and ADAFSLEN should equal the value of ADAFTOTL (for record 3 in this example, 0 + 20000 = 20000)
		ADAFISN	3	ISN of record
		ADAFLNUM	3	Logical record number
		ADAFPNUM	4	Physical record number
		ADAFEOFF	0	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN		Field name
		ADAFRSP	233	Response Code
		ADAFSUB	1	Subcode
		Reserved	0	Reserved

Rejected Record	FEHL Records	FEHL Fields		Description
		Field	Value	
		ADAFDATA	ADAH and payload data	Rejected input data
4	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	H	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	C	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	32	Segment length (length of payload data following the header)
		ADAFTOTL	10064	Total length. **Note:** The sum of the values of all ADANSLEN fields and ADAFSLEN should equal the value of ADAFTOTL (for record 4 in this example, 32 + 32 + 10000 = 10064)
		ADAFISN	4	ISN of record
		ADAFLNUM	4	Logical record number
		ADAFPNUM	8	Physical record number
		ADAFEOFF	0	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN		Field name
		ADAFRSP	234	Response Code
		ADAFSUB	3	Subcode
		Reserved	0	Reserved
		ADAFDATA	ADAH header	Rejected input data
	ADAN	ADANEYE	ADAN	ADAN header eye-catcher
		ADANLEN	24	ADAN header length
		ADANTYPE	C	Type. Valid values are: C: ADAC header D: ADAC header and payload P: ADAH record segment R: Logical record segment

Rejected Record	FEHL Records	FEHL Fields		Description
		Field	Value	
		ADANIND	C	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADANSLEN	32	Segment length (length of payload data following the header)
		ADANOFF	32	Error offset in logical record.
		ADANDATA	ADAC header	Continued rejected input data
	ADAN	ADANEYE	ADAN	ADAN header eye-catcher
		ADANLEN	24	ADAN header length
		ADANTYPE	D	Type. Valid values are: C: ADAC header D: ADAC header and payload P: ADAH record segment R: Logical record segment
		ADANIND	E	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADANSLEN	10000	Segment length (length of payload data following the header)
		ADANOFF	64	Error offset in logical record.
		ADANDATA	ADAC and payload data	Continued rejected input data

ADACMP Report

ADACMP calculates the approximate amount of space (in both blocks and cylinders) required for Data Storage for the compressed records. This information is printed as a matrix which contains the required space for the different device types requested by the DEVICE parameter for various Data Storage padding factors between 5 and 30 percent.

The following is an example of ADACMP report output:

```
PARAMETERS:

ADACMP COMPRESS NUMREC=1000
ADACMP FNDEF='01,AA,8,B,DE'
ADACMP FNDEF='01,BA,6,A,NU'
ADACMP FNDEF='01,BB,8,P,NU'
ADACMP FNDEF='01,AD,1,A,FI'
ADACMP SUBDE='CA=BA(1,3)'

COMPRESS PROCESSING STATISTICS:

NUMBER OF RECORDS READ                1,000
NUMBER OF INCORRECT RECORDS               0
NUMBER OF COMPRESSED RECORDS          1,000

RAW DATA                             24,000 BYTES
COMPRESSED DATA                      16,656 BYTES
COMPRESSION RATE                       31.9 %
LARGEST COMPRESSED RECORD                 20 BYTES


DATASTORAGE SPACE REQUIREMENTS:

I  DEVICE  I  PADDING I  BLOCKSIZE I          NUMBER OF     I
I          I  FACTOR  I    BYTES   I      BLOCKS      CYLS  I
I----------I----------I------------I-------------------------I
I   3380   I          I    4,820   I                         I
I          I     5%   I    4,578   I           4         1   I
I          I    10%   I    4,337   I           4         1   I
I          I    15%   I    4,096   I           5         1   I
I          I    20%   I    3,856   I           5         1   I
I          I    25%   I    3,615   I           5         1   I
I          I    30%   I    3,373   I           5         1   I
I          I          I            I                         I
I----------I----------I------------I-------------------------I

TEMP SPACE ESTIMATION:

I  DEVICE  I  BLOCKSIZE I          NUMBER OF     I
I          I    BYTES   I      BLOCKS      CYLS  I
I----------I------------I-------------------------I
I   3380   I    7,476   I           5          1 I
I----------I------------I-------------------------I

THE LARGEST DESCRIPTOR IS AA, IT WILL OCCUPY 1 TEMP BLOCKS


SORT SPACE ESTIMATION:
```

```
I DEVICE I BLOCKSIZE I   LWP              I      NR OF      I
I        I  (BYTES)  I  (BYTES)           I  BLOCKS    CYLS I
-------------------------------------------------------------
I  3380  I     7476  I   139264 (MINIMUM) I       2       1 I
I        I           I  1048576 (DEFAULT) I       2       1 I
I        I           I   139264 (OPTIMUM) I       2       1 I
I--------I-----------I--------------------I-----------------I
```

The compression rate is computed based on the real amount of data used as input to the compression routine. Fields skipped by a format element "nX" (used to fill a field with blanks) are not counted.

If SPAN was specified in an ADACMP COMPRESS run, statistics about the spanned Data Storage records are also printed:

```
Spanned Record Statistics:
--------------------------

Number of Non-Spanned records              2
Number of Spanned records                  8
Min Number of Segments             2
Max Number of Segments             5
Avg Number of Segments             4

Max MU Count                       0
Max PE Count                     300
```

If large object (LB) fields are compressed, statistics about the LB fields (listed as "LOBs" in the report) are printed:

```
Large Objects Statistics:
-------------------------

Number of bytes for largest LOB          10,124,996
Total number of LOBs                                5
Total number of outsourced LOBs                     5
Number of LOB file records                     10,085
```

19 DECOMPRESS Function Output

- Rejected Data Records 172

The ADACMP DECOMPRESS function decompresses each record and then stores the record in a sequential data set. The records are output in variable-length, blocked format. Each decompressed record is output either with or without the ISN option according to the format shown below:

```
length xx [ISN] data
```

where

length	is a two-byte binary length of the data, + 8 (or +4 if the ISN parameter is not specified).
xx	is a two-byte field containing binary zeros.
ISN	is a four-byte binary ISN of the record.
data	is a decompressed data record.

The fields of the data record are provided in the order in which they appeared in the FDT when the file was unloaded. The standard length and format are in effect for each field.

If a field value exceeds the standard length, the value will be truncated to the standard length if the field is alphanumeric and the TRUNCATE parameter was specified; otherwise, ADACMP writes the record to the DD/FEHL error data set (read the next chapter, ***Rejected Data Records***).

Any count bytes for multiple-value fields or periodic groups contained in the record are included in the decompressed data output. ADACMP generates a count of 1 if the MU field or PE group is empty. This makes it possible to use the output of the DECOMPRESS operation as the input to a subsequent COMPRESS operation.

Rejected Data Records

ADACMP rejects a record whenever a compressed field's size is greater than the default length held in the FDT, unless the TRUNCATE parameter is specified.

Any records rejected during ADACMP decompression are written to the DD/FEHL error data set. The records are output in variable blocked format and may be segmented into multiple physical records. Each logical rejected record will be preceded by an initial ADAF rejected record header. If the logical record and the ADAF header do not fit in the DD/FEHL physical record length, ADAN rejected record headers will precede the remaining physical rejected record segments that comprise the logical rejected record.

The functions of these two different headers are as follows:

- ADAF headers indicate the error condition and pertinent information.
- ADAN headers are smaller and are used for rejected record continuation and ADAH/ADAC header error reporting.

DSECTs for the ADAF and ADAN headers can be found in members ADAF and ADAN of the distributed Adabas SRCE data set.

Traditionally, the DD/FEHL error data set produced for ADACMP errors has truncated rejected records that exceeded the FEHL physical record length. In Version 8, the rejected records are segmented instead of truncated. Because of this change, the DD/FEHL LRECL setting must be at least 500 bytes.

The following response codes may occur:

X'E7'(231)	Input record too short (DECOMPRESS)
X'E8'(232)	Output record length error (DECOMPRESS)

Notes:

1. Only the first incorrect field within a record is detected and referenced in DD/FEHL. Other errors within the record are not detected or recorded.

Example of Rejected Data Records

The following table depicts the FEHL output for two rejected records during ADACMP decompression. Rejected record 1 has only one FEHL record (ADAF); rejected record 2 is segemented into two FEHL records (ADAF and ADAN):

Note: DSECTs for the ADAF and ADAN headers can be found in members ADAF and ADAN of the distributed Adabas SRCE data set.

Rejected Record	DDFEHL Records	DDFEHL Fields		Description
		Field	Value	
1	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	R	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	E	Continuation indicator. Valid values are: C: Continuation record to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	22000	Segment length
		ADAFTOTL	22000	Total length
		ADAFISN	1	ISN of record

Rejected Record	DDFEHL Records	DDFEHL Fields		Description
		Field	Value	
		ADAFLNUM	1	Logical record number
		ADAFPNUM	1	Physical record number
		ADAFEOFF	5000	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN	ZA	Field name
		ADAFRSP	41	Response code
		ADAFSUB	2	Subcode
		Reserved	0	Reserved
		ADAFDATA	'Record 1 -- Payload Data'	Rejected input data
2	ADAF	ADAFEYE	ADAF	ADAF header eye-catcher
		ADAFLEN	72	ADAF header length
		ADAFTYPE	R	Type. Valid values are: H: ADAH header P: ADAH header and payload R: Logical record
		ADAFIND	C	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADAFSLEN	27962	Segment length
		ADAFTOTL	50000	Total length
		ADAFISN	2	ISN of record
		ADAFLNUM	2	Logical record number
		ADAFPNUM	3	Physical record number
		ADAFEOFF	35000	Error offset in logical record
		ADAFPEX	0	PE index
		ADAFFN	ZA	Field name
		ADAFRSP	41	Response code
		ADAFSUB	2	Subcode
		Reserved	0	Reserved
		ADAFDATA	'Record 2 -- Payload data part 1'	Rejected input data
	ADAN	ADANEYE	ADAN	ADAN header eye-catcher
		ADANLEN	24	ADAN header length

Rejected Record	DDFEHL Records	DDFEHL Fields		Description
		Field	Value	
		ADANTYPE	R	Type. Valid values are: C: ADAC header D: ADAC header and payload P: ADAH record segment R: Logical record segment
		ADANIND	E	Continuation indicator. Valid values are: C: Continuation record (ADAN) to follow E: End of logical record (last segment)
		Reserved	0	Reserved
		ADANSLEN	22038	Segment length. **Note:** The sum of the values of ADANSLEN and ADAFSLEN should equal the value of ADAFTOTL (in this example, 22038 +27962=50000)
		ADANOFF	27962	Error offset in logical record.
		ADANDATA	'Record 2 -- Payload data part 2'	Continued rejected input data

IV ADACNV Utility: Database Conversion

This chapter covers the following topics:

Functional Overview
CONVERT: Convert Database to Higher Version
REVERT: Revert Database to Lower Version
JCL/JCS Requirements and Examples

20 Functional Overview

- Database Status 215
- Procedure 181

The ADACNV utility can convert (CONVERT) an Adabas mainframe database from version 6.1 or above to a higher version or revert (REVERT) an Adabas database from a higher version back to a lower version.

Caution: Before you convert a database, you must terminate all active nucleus or utility jobs normally.

To ensure database integrity, ADACNV writes changed blocks first to intermediate storage; that is, to the sequential data set DD/FILEA. After all changed blocks have been written out to DD/FILEA, a *point-of-no-return* is reached and the changed blocks are written to the database. If ADACNV terminates abnormally after the *point-of-no-return*, the RESTART parameter can be used to begin the ADACNV run by reading the contents of DD/FILEA and writing them out to the database.

The TEST parameter is provided to check the feasibility of a conversion or reversion without writing any changes to the database. It is therefore not necessary to terminate all activity on the database before running ADACNV when you use the TEST parameter.

Database Status

Internally, the utility converts or reverts one version at a time until the target version is attained. It is therefore important to ensure that all requirements for conversion or reversion between the current and target database levels have been met before you execute ADACNV without the TEST parameter.

Before a conversion or reversion begins, ADACNV checks the status of the database:

- The DIB must be empty; that is, no Adabas nucleus or utility may be active or have been terminated abnormally. If RESTART is specified, the DIB must contain the entry of ADACNV, which includes a time stamp.
- For conversion from version 5.2, the checkpoint block 8 must have enough free space to accommodate the expanded 24-byte header used for version 5.3 and above. For reversion to version 5.2, the checkpoint blocks 20-24 must be empty.
- The Work data set must not have a pending autorestart.

If this check is successful, ADACNV locks the database and creates a DIB entry.

For reversions, ADACNV checks whether any features are used that do not exist in the target version and returns a message if any are found.

Note: There may be files in a database that are not loaded but that have a Field Definition Table (FDT) stored in the FDT blocks. If ADACNV encounters such FDTs while converting a database to Version 8, they are deleted as part of its cleanup processing.

Procedure

The procedure for converting or reverting an Adabas database is as follows:

1. If the nucleus is active, use ADAEND to stop it.
2. Use ADARES PLCOPY/CLCOPY to copy all protection and command logs.

   For your installation, this may be done automatically with user exit 2.

   Wait until the logs have been copied.
3. Optionally, back up the database (full or delta).
4. Execute the ADACNV utility.
5. Start the nucleus of the version to which you have converted or reverted.

Important: To ensure database integrity, DD/FILEA must be defined permanently and be deleted only after ADACNV has completed successfully. The DD/FILEA data set must not be defined as a temporary data set that is automatically deleted at the end of the job.

21 CONVERT: Convert Database to Higher Version

- Optional Parameters 184
- Conversion Considerations 185
- Example 186

The CONVERT function starts from the Adabas version of the last nucleus session.

```
ADACNV  CONVERT  [IGNPPT]
                 [NOUSERABEND]
                 [PLOGDEV = { multiple-PLOG-device-type | ADARUN-device } ]
                 [RESTART]
                 [TEST]
                 [TOVERS = {target-version | ADACNV-version  } ]
```

Optional Parameters

IGNPPT: Ignore Parallel Participant Table PLOG Entries

When converting from a version of Adabas that uses the parallel participant table (PPT) structure to a higher version of Adabas, an error is printed and conversion fails if the system detects one or more protection logs (PLOGs) from the current version that have not been copied/merged.

Specify IGNPPT to continue processing in spite of the uncopied/unmerged PLOGs.

Note: If DDPLOG*x* statements have been specified in the JCL in addition to the DDPLOG*x* data sets in the PPT, the specified data sets must be empty or the error will still be received. IGNPPT only pertains to the PPT processing. If PLOG data sets are supplied in the JCL, they must be empty.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PLOGDEV: Multiple PLOG Device Type

PLOGDEV specifies the physical device type on which the multiple protection log data sets to be converted are contained. If PLOGDEV is not specified, the device type specified by the ADARUN DEVICE parameter is used.

RESTART: Rerun after Point of No Return

If ADACNV terminates abnormally after the *point-of-no-return*, that is, after all changed blocks have been written to DD/FILEA, the RESTART parameter instructs ADACNV to begin its run by reading the contents of DD/FILEA and continue by writing them to the database.

TEST: Test Conversion

The TEST parameter tests the feasibility of the conversion operation without actually writing any changes to the database.

TOVERS: Target Version

The two-character version of Adabas database (version and revision level) to achieve at the end of the ADACNV run. If the TOVERS parameter is

- specified, it must be a version higher than the source version.
- not specified, ADACNV uses its own version as the target version.

The version format is *vr* indicating the version and revision level; for example, "74" or "81".

Note: There may be files in a database that are not loaded but that have a Field Definition Table (FDT) stored in the FDT blocks. If ADACNV encounters such FDTs while converting a database to Version 8, they are deleted as part of its cleanup processing.

Conversion Considerations

The following is an overview of the conversion steps performed by ADACNV.

All Versions

- The data protection area on the Work data set and the multiple PLOG data sets (if supplied) are cleared to binary zeros.

From Version 6.1 to 6.2

- Any Adabas Delta Save DLOG area header is set to the correct version.

From Version 6.2 to 7.1

- Any Adabas Delta Save DLOG area header is set to the correct version.

From Version 7.1 to Version 8

You cannot use CONVERT to convert a database directly from Adabas 7.1 to Adabas 8. To do this, you must first convert the database to Adabas 7.2 or 7.4 and then convert it to Adabas 8.

Any Version to Version 8

There may be files in a database that are not loaded but that have a Field Definition Table (FDT) stored in the FDT blocks. If ADACNV encounters such FDTs while converting a database to Version 8, they are deleted as part of its cleanup processing.

Example

```
ADACNV CONVERT TOVERS=81
```

The version of Adabas selected in the last nucleus session is to be converted to a version 8.1 database.

22 REVERT: Revert Database to Lower Version

- Essential Parameter and Subparameter 188
- Optional Parameter 188
- Reversion Considerations 230
- Example 190

The REVERT function starts from the Adabas version of the last nucleus session.

```
ADACNV  REVERT  [TOVERS = target-version ]
                [IGNPPT]
                [NOUSERABEND]
                [PLOGDEV = { multiple-PLOG-device-type | ADARUN-device } ]
                [RESTART]
                [TEST]
```

Essential Parameter and Subparameter

TOVERS: Target Version

The version of Adabas database (version and revision level) to achieve at the end of the ADACNV run. The TOVERS parameter value must be a version lower than the source version.

The version format is *vr* indicating the version and revision level; for example, *61*.

Optional Parameter

IGNPPT: Ignore Parallel Participant Table PLOG Entries

When reverting from a version of Adabas that uses the parallel participant table (PPT) structure to a lower version of Adabas, an error is printed and conversion fails if the system detects one or more protection logs (PLOGs) from the current version that have not been copied/merged.

If IGNPPT is specified, the utility will continue processing in spite of the uncopied/unmerged PLOGs.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PLOGDEV: Multiple PLOG Device Type

PLOGDEV specifies the physical device type on which the multiple protection log data sets to be reverted is contained. If PLOGDEV is not specified, the device type specified by the ADARUN DEVICE parameter is used.

RESTART: Rerun after Point of No Return

If ADACNV terminates abnormally after the *point-of-no-return*, that is, after all changed blocks have been written to DD/FILEA, the RESTART parameter instructs ADACNV to begin its run by reading the contents of DD/FILEA and continue by writing them to the database.

TEST: Test Conversion

The TEST parameter tests the feasibility of the reversion operation without actually writing any changes to the database.

Reversion Considerations

The following is an overview of the reversion steps performed by ADACNV.

All Versions

- Reversion is not possible if any Adabas feature is used in the current version that is not supported in the target version. This statement applies to all Adabas features that affect the structure of the database.

From Version 8 to Any Prior Version

If a database makes use of any of the following extended features of Adabas 8, ADACNV will not allow you to revert the database to a version prior to Adabas 8:

- More than five ASSO, DATA, or DSST extents
- More than five file extents
- Files that allow spanned records
- Files that allow more than 191 MU and PE occurrences
- Files that make use of large object (LB) fields
- Files with fields that have the NB (no blank compression) option
- System files with two-byte file numbers.

If you want to complete the backward conversion, you must first remove any file with these new features from the Adabas database.

The use of the following other new features provided in Adabas 8 do *not* prevent backward conversion to Adabas 7.4, but, of course, the new features cannot be used in Adabas 7.4:

- Adabas commands issued via the ACBX interface (for example, with long or segmented buffers)
- Commands using the new format buffer features (for example, the length indicator).

From Version 8 to Version 7.1

You cannot use REVERT to revert a database directly from Adabas 8 to Adabas 7.1; instead, you must first revert the database to Adabas 7.4 and then from 7.4 to 7.1.

From Version 7.1 to 6.2

- Version 7.1 extends the free space table (FST) from one RABN (RABN 10) to five RABNs (RABNs 10-14). ADACNV checks whether all FST entries fit into one RABN. If not, the smallest FST extent is removed. This is repeated until the FST fits into one ASSO block. An appropriate message is printed.
- Any Adabas Delta Save DLOG area header is set to the correct version.

From Version 6.2 to 6.1

- Any Adabas Delta Save DLOG area header is set to the correct version.

Example

```
ADACNV REVERT TOVERS=73
```

The Adabas version of the last run of the nucleus is to be converted back (reverted) to a version 7.3 Adabas database.

23 JCL/JCS Requirements and Examples

- BS2000 192
- z/OS 195
- z/VM 196
- VSE 197

This section describes the job control information required to run ADACNV with BS2000, z/OS, z/VM, and VSE systems, and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/ disk	see Note
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADACNV parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADACNV messages	SYSLST/ DDDRUCK		*Messages and Codes*

Note: The intermediate storage is read an undefined number of times. If this storage is on tape/cassette, it is necessary to use the ADARUN parameter TAPEREL=NO to prevent the tape from being released. Software AG then recommends that you put a tape release command in the job to free the tape/cassette unit when the job has finished. See the example following.

ADACNV JCL Example (BS2000)

With Intermediate Disk File Storage

In SDF Format:

```
/.ADACNV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK *
/DELETE-FILE ADAyyyyy.FILEA
/SET-JOB-STEP
/CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
/SET-JOB-STEP
/ASS-SYSLST L.CNV.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
```

```
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
/SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B
ADACNV CONVERT TOVERS=vr
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACNV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK *
/SYSFILE SYSLST=L.CNV.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,SHARUPD=YES
/FILE ADAyyyyy.FILEA,LINK=DDFILEA,SPACE=(4800,480)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B
ADACNV CONVERT TOVERS=vr
/LOGOFF NOSPOOL
```

With Intermediate Tape/Cassette File Storage

In SDF Format:

```
/.ADACNV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK * INTERMEDIATE TAPE/CASSETTE STORAGE
/REMARK *
/DELETE-FILE ADAyyyyy.FILEA
/SET-JOB-STEP
/CREATE-FILE ADAyyyyy.FILEA,TAPE(DEV-TYPE=T-C1,VOL=ADA001)
/SET-JOB-STEP
/ASS-SYSLST L.CNV.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
```

```
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
/SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA,TAPE(FILE-SEQ=1),OPEN-MODE=OUTIN
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
ADACNV CONVERT TOVERS=vr
/SET-JOB-STEP
/REMARK * NOW RELEASE THE TAPE
/REM-FILE-LINK DDFILEA,UNL-REL-TAPE=YES
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACNV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK * INTERMEDIATE TAPE/CASSETTE STORAGE
/REMARK *
/SYSFILE SYSLST=L.CNV.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,SHARUPD=YES
/FILE ADAyyyyy.FILEA,LINK=DDFILEA,DEVICE=T C1,VOLUME=ADA001

/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
ADACNV CONVERT TOVERS=vr
/STEP
/REMARK * NOW RELEASE THE TAPE
/REL DDFILEA,UNLOAD
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/ disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADACNV parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACNV messages	DDDRUCK	printer	*Messages and Codes*

ADACNV JCL Example (z/OS)

Refer to ADACNV in the JOBS data set for this example.

```
//ADACNV    JOB
//*
//*    ADACNV:
//*       EXAMPLE HOW TO USE ADACNV TO CONVERT DATABASE
//*       TO A DIFFERENT VERSION
//*
//CNV      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDPLOGR1  DD   DSN=EXAMPLE.DByyyyy.PLOGR1,DISP=SHR <=== PLOG 1
//DDPLOGR2  DD   DSN=EXAMPLE.DByyyyy.PLOGR2,DISP=SHR <=== PLOG 2

//DDFILEA   DD   DSN=EXAMPLE.DByyyyy.FILEA,         <=== INTERMEDIATE FILE
//               UNIT=SYSDA,SPACE=(TRK,(150,150),RLSE),
//               DISP=(NEW,CATLG)
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADACNV,SVC=xxx,DE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
```

```
ADACNV CONVERT TOVERS=vr
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/ disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADACNV parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADACNV messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADACNV JCL Example (z/VM)

```
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOGR1,VOL=PLOGV
DATADEF DDPLOGR2,DSN=ADABASVv.PLOGR2,VOL=PLOGV
DATADEF DDFILEA,DSN=ADACNV.FILEA,MODE=A
DATADEF DDPRINT,DSN=ADACNV,DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADACNV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNCNV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=CONVERT.CONTROL,MODE=A
ADARUN
```

Contents of RUNCNV CONTROL A1:

```
ADARUN PROG=ADACNV,DEVICE=dddd,DB=yyyyy
```

Contents of CONVERT CONTROL A1:

```
ADACNV CONVERT TOVERS=vr
```

VSE

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
Work	WORKRn	disk	*	
Multiple protection logs	PLOGRn	disk	*	
Intermediate storage	FILEA	tape disk	SYS010 *	
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations*
ADACNV parameters	-	reader	SYSIPT	
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADACNV messages	-	printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADACNV JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures.

Refer to member ADACNV.X for this example.

```
* $$ JOB JNM=ADACNV,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       CONVERT DATABASE TO NEW VERSION
// JOB ADACNV
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// DLBL FILEA,'ADACNV.WORK.FILE',0,SD
// EXTENT SYS015,,,,ssss,nnnn
// ASSGN SYS015,DISK,VOL=vvvvvv,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN DBID=yyyyy,DEVICE=dddd,PROG=ADACNV,SVC=xxx
/*
ADACNV CONVERT TOVERS=vr
/*
/&
* $$ EOJ
```

V ADADBS Utility: Database Services

This chapter covers the following topics:

Functional Overview
ADD: Add Data Set
ALLOCATE: Allocate File Extent
CHANGE: Change Standard Length or Format of a Field
CVOLSER: Print Adabas Extents on Given Volume
DEALLOCATE: Deallocate File Extent
DECREASE: Decrease Last Associator or Data Storage Data Set Size
DELCP: Delete Checkpoint Records
DELETE: Delete File
DSREUSE: Reuse Data Storage Blocks
ENCODEF: Change File Encoding
INCREASE: Increase Last Associator or Data Storage Data Set Size
ISNREUSE: Reuse ISNs
MODFCB: Modify File Parameters
MUPEX: Set Maximum Count for MU and PE Fields
NEWFIELD: Add New Field
ONLINVERT: Start Online Invert Process
ONLREORFASSO: Start Online Reorder Associator for Files
ONLREORFDATA: Start Online Reorder Data for Files
ONLREORFILE: Start Online Reorder Associator and Data for Files
OPERCOM: Issue Adabas Operator Commands
PRIORITY: Change User Priority
RECORDSPANNING: Enable or Disable Record Spanning
RECOVER: Recover Space
REFRESH: Set File to Empty Status

REFRESHSTATS: Reset Statistical Values
RELEASE: Release Descriptor
RENAME: Rename a File or Database
RENUMBER: Change File Number
REPLICATION: Activating or Deactivating Replication
REPTOR: Activate, Deactivate, Open, or Close Event Replicator Resources
RESETDIB: Reset Entries in Active Utility List
RESETPPT: Reset PPT Blocks
SPANCOUNT: Count Spanned Records
TRANSACTIONS: Suspend and Resume Update Transaction Processing
UNCOUPLE: Uncouple Files
JCL/JCS Requirements and Examples

24 Functional Overview

- Syntax Checking with the TEST Parameter 202

Note: All ADADBS functions can also be performed using Adabas Online System (AOS). When using the Adabas Recovery Aid, using AOS is preferable for file change operations because it writes checkpoints that are necessary for recovery operation.

Any number of functions may be performed during a single execution of ADADBS.

Syntax Checking with the TEST Parameter

The ADADBS functions now include a syntax-checking-only mode. When the TEST parameter is specified, the actual ADADBS function is checked, but not performed.

The ADADBS utility can perform multiple functions. As a result, ADADBS reads the parameters up to the next specified ADADBS function, and then executes the function/parameters just read. Then, ADADBS reads the function and parameters up to the following function, and so on. Therefore, to ensure that no functions are executed, the TEST parameter must be specified either before or within the first function/parameter group, as the following example shows:

```
ADADBS TEST
ADADBS DELETE FILE=1
ADADBS DELETE FILE=2
```

25 ADD: Add Data Set

- Associator or Data Storage Data Set 204
- Essential Parameter and Subparameter 205
- Optional Parameters 278
- Examples 205

The ADD function adds a new data set to the Associator or Data Storage.

```
ADADBS  ADD  { ASSOSIZE = size  [ ASSODEV = {device-type | ADARUN-device } ] |
             DATASIZE = size [ DATADEV = {device-type | ADARUN-device } ] }
             [NOUSERABEND]
             [TEST]
```

This chapter describes the syntax, processing, and parameters of the ADADBS ADD function.

Associator or Data Storage Data Set

For the Associator or for Data Storage, the data set to be added may be on the same device type as that currently being used or on a different one. A maximum of 99 physical extents is now set for Associator and Data Storage data sets. However, your actual real maximum could be less because the extent descriptions of all Associator, Data Storage, and Data Storage Space Table (DSST) extents must fit into the general control blocks (GCBs). For example, on a standard 3390 device type, there could be more than 75 Associator, Data Storage, and DSST extents each (or there could be more of one extent type if there are less for another).

Note: The Associator and Data Storage data set sizes must be added separately. It is *not* possible to add both with a single operation.

After an ADD operation is completed for an Associator or Data Storage data set, the ADD function automatically ends the current nucleus session. A message informs you when the nucleus has been stopped. Assuming the added data set has already been formatted and the JCL/JCS has been updated for all nucleus startup and utility procedures, the nucleus can then be restarted. Once it is restarted, the additional free space added by the ADD operation will be available.

Procedure

To add an additional data set to the Associator or Data Storage

1. Allocate the data set with the operating system, then format the additional space using the ADAFRM utility.
2. Add necessary JCL/JCS to all Adabas nucleus and Adabas utility execution procedures.
3. Execute the ADD function.
4. Restart the nucleus.

Essential Parameter and Subparameter

ASSODEV | DATADEV: Device Type

The device type to be used for the new data set. These parameters are required only if a different device type from the device type specified by the ADARUN DEVICE parameter is to be used.

For VSAM data sets, use dynamic device types; that is, DDxxxxR1=9999, DDxxxxR2=8888, ... DDxxxxR5=5555. For example, if DDDATAR3 is added, DATADEV=7777.

ASSOSIZE | DATASIZE: Size of Data Set to be Added

The number of cylinders to be contained in the new data set.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables *cannot* be tested: only the syntax of the specified parameters can be tested. See **Syntax Checking with the TEST Parameter** for more information about using the TEST parameter in ADADBS functions.

Examples

A new data set containing 800 cylinders on 3350 disks is to be added to Data Storage.

```
ADADBS ADD DATASIZE=800, DATADEV=3350
```

A new data set containing 100 cylinders is to be added to the Associator on the Associator's existing device type.

```
ADADBS ADD ASSOSIZE=100
```

26 ALLOCATE: Allocate File Extent

- Essential Parameters 208
- Optional Parameters 209
- Example 209

The ALLOCATE function may be used to allocate an address converter, Data Storage, normal or upper index extent of a specific size. It can also be used to allocate a secondary address converter when spanned records are included in the data. Only one extent may be allocated per ADADBS execution.

```
ADADBS  ALLOCATE    FILE = file-number
                    { ACSIZE | AC2SIZE | DSSIZE | NISIZE | UISIZE } = size
                    [DEVICE = device-type ]
                    [NOUSERABEND]
                    [PASSWORD = ' password ' ]
                    [STARTRABN = start-rabn ]
                    [TEST]
```

This chapter describes the syntax, processing, and parameters of the ADADBS ALLOCATE function.

Essential Parameters

FILE: File for Which an Extent Is Allocated

FILE specifies the number of the file for which the extent is to be allocated.

ACSIZE | AC2SIZE | DSSIZE | NISIZE | UISIZE: Extent Type and Size

These parameters are used to indicate the type and size of the extent to be allocated. One and only one extent type and size can be specified in a single ADADBS ALLOCATE statement. The specified value can be either cylinders or blocks; a size in blocks must be followed by "B" (for example, 2000B).

The extents that can be allocated are:

- the address converter (ACSIZE)
- the secondary address converter, when spanned records are used (AC2SIZE)
- Data Storage (DSSIZE)
- the normal index (NISIZE)
- the upper index (UISIZE).

Note: If the specified address converter size (ACSIZE or AC2SIZE) would increase the existing address converter beyond the maximum size derived from the ISNSIZE attribute of the file (for ISNSIZE=3: 16,777,215 ISNs; for ISNSIZE=4: 4,294,967,294 ISNs), Adabas will allocate additional address converter blocks only up to that limit.

Optional Parameters

DEVICE: Device Type

The device type to be used for file allocation. If none is specified, Adabas chooses one from the available device types with free space in the database.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file. This parameter is required if the file is password-protected.

STARTRABN: Starting RABN for Extent

The beginning RABN of the extent to be allocated. If this parameter is omitted, ADADBS will assign the starting RABN.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

An address converter extent of 30 blocks is to be allocated for file 15.

```
ADADBS ALLOCATE FILE=15,ACSIZE=30B ↩
```

27 CHANGE: Change Standard Length or Format of a Field

- Essential Parameters 212
- Optional Parameters 213
- Example 213

The CHANGE function can be used to change

- the standard length of an Adabas field;
- a normal alphanumeric (A) field to a long-alpha (LA) field; or
- the default field format from unpacked (U) to packed (P).

Only one of these changes may be performed per function execution.

No modifications to records in Data Storage are made by this function. The user is, therefore, responsible for preventing references to the field that would cause invalid results because of an inconsistency between the new standard length as defined to Adabas and the actual number of bytes contained in the record.

When changing the length of an Adabas expanded file field, the change must be made to *each individual component file* of the expanded file. Each CHANGE operation on a component file causes a message that confirms the change, and returns condition code 4.

```
ADADBS  CHANGE  FILE = file-number
                FIELD = 'field-name'
                { FORMAT=P | LENGTH=new-length | OPTION=LA }
                [ NOUSERABEND ]
                [ PASSWORD = 'password ' ]
                [TEST]
```

Essential Parameters

FILE: File Containing the Field

The file in which the field whose length is to be changed is contained. An Adabas system file may not be specified.

FIELD: Field to be Changed

The field whose standard length is to be changed. The field cannot be one that was defined with the FI option, or a field with a defined length of zero (variable-length field). Specify the field name between apostrophes (').

FORMAT=P: New Field Format

The new standard field format. The only field format change supported is from 'U' (unpacked) to 'P' (packed). The field cannot be parent of a sub-/super-/hyperdescriptor.

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

LENGTH: New Field Length

The new standard length for the field. A length of 0 is not permitted, nor can a field with an existing defined length of zero (such as a variable-length field) be redefined to a standard length.

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

OPTION=LA: New Field Option

The new field option. The only field option change supported is from normal alphanumeric (A) to long-alpha (LA).

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file containing the field to be changed. This parameter is required if the file is password-protected.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

The standard length of field AB in file 5 is to be changed to 11 bytes.

```
ADADBS CHANGE FILE=5,FIELD='AB',LENGTH=11
```

28 CVOLSER: Print Adabas Extents on Given Volume

- Essential Parameter 216
- Optional Parameters 286
- Example 216

The CVOLSER function is used to print the Adabas file extents contained on a disk volume.

```
ADADBS  CVOLSER  VOLSER = volume-serial-number
                 [NOUSERABEND]
                 [TEST]
```

Essential Parameter

VOLSER: Volume Serial Number

VOLSER is the volume serial number of the disk volume to be used.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

The Adabas file extents contained on disk volume DISK02 are to be printed.

```
ADADBS CVOLSER VOLSER=DISK02
```

29 DEALLOCATE: Deallocate File Extent

- Essential Parameters 218
- Optional Parameters 287
- Example 219

The DEALLOCATE function may be used to deallocate an address converter, Data Storage, normal index or upper index extent. It can also be used to deallocate a secondary address converter when spanned records are included in the data. Only one extent may be deallocated per ADADBS execution.

```
ADADBS  DEALLOCATE  FILE = file-number
                    { ACSIZE | AC2SIZE | DSSIZE | NISIZE | UISIZE } = size
                    [NOUSERABEND]
                    [PASSWORD = ' password ' ]
                    [STARTRABN = start-rabn ]
                    [TEST]
```

Essential Parameters

ACSIZE | AC2SIZE | DSSIZE | NISIZE | UISIZE: Extent Type and Size

These parameters specify the type and size of extent to be deallocated. One and only one extent type and size may be specified. The size must be in number of RABN blocks followed by "B" (for example, DSSIZE=20B), and cannot exceed the number of unused RABNs at the end of an extent.

The extents that can be deallocated are:

- the address converter (ACSIZE)
- the secondary address converter, when spanned records are used (AC2SIZE)
- Data Storage (DSSIZE)
- the normal index (NISIZE)
- the upper index (UISIZE).

FILE: File for Which an Extent Is Deallocated

FILE specifies the file for which the extent is to be deallocated. Specify a decimal value.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file for which space is to be deallocated. This parameter is required if the file is password-protected. Specify the password between apostrophes (').

STARTRABN: Starting RABN for Extent

The first RABN of the extent in which deallocation is to take place. If this parameter is omitted, the last extent for the file will be deallocated. In the address converter, only the last extent may be deallocated.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

An address converter extent of 30 blocks is to be deallocated for file 15.

```
ADADBS DEALLOCATE FILE=15,ACSIZE=30B
```

30 DECREASE: Decrease Last Associator or Data Storage Data Set Size

- Essential Parameter 222
- Optional Parameters 222
- Example 289
- Procedure 289

The DECREASE function decreases the size of the last data set currently being used for the Associator or Data Storage. The space to be released must be available in the free space table (FST).

The DECREASE function does *not* deallocate any of the specified physical extent space.

```
ADADBS  DECREASE   { ASSOSIZE | DATASIZE } = sizeB
                   [NOUSERABEND]
                   [TEST]
```

Essential Parameter

ASSOSIZE | DATASIZE: Blocks to Be Decreased

ASSOSIZE/DATASIZE define the number of blocks by which the Associator or Data Storage data set is to be decreased, specified as a decimal value followed by "B". Either ASSOSIZE or DATASIZE can be specified, but not both. If both ASSOSIZE and DATASIZE are to be specified, each must be entered on a separate ADADBS DECREASE statement.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

The Associator is to be decreased by 100 blocks and Data Storage is to be decreased by 200 blocks.

```
ADADBS DECREASE ASSOSIZE=100B
ADADBS DECREASE DATASIZE=200B
```

Procedure

To deallocate space, perform the following steps:

1 Decrease the database with the DECREASE function;

2 Save the database with ADASAV SAVE;

3 Reformat the data sets with ADAFRM;

4 Restore the database with ADASAV.

31 DELCP: Delete Checkpoint Records

- Essential Parameter 226
- Optional Parameters 226
- Example 289

The DELCP function deletes checkpoint records.

After running ADADBS DELCP, the remaining records are reassigned ISNs to include those ISNs made available when the checkpoint records were deleted. The lower ISNs are assigned but the chronological order of checkpoints is maintained.

```
ADADBS  DELCP  TODATE = yyyymmdd
               [NOUSERABEND]
               [TEST]
```

Essential Parameter

TODATE: Last Date for Deleted Records

TODATE specifies the latest date for which checkpoint information is deleted. Checkpoint information dated after the date specified by TODATE= is not deleted. TODATE= must be specified; there is no default date. Specify the date as a four-digit decimal value for year followed by two-digit decimal values for month and day, in that order.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

All checkpoint records up to and including February 1, 1996 are to be deleted.

```
ADADBS DELCP TODATE=19960201
```

32 DELETE: Delete File

- Essential Parameter 290
- Optional Parameters 230
- Examples 231

The DELETE function deletes an Adabas file from the database.

```
ADADBS  DELETE   { FILE = fnr  [KEEPFDT] [PASSWORD = 'password' ] }
                 [NOUSERABEND]
                 [TEST]
```

When an Adabas file is deleted from the database, all logical extents assigned to the file are deallocated. The released space may be used for a new file or for a new extent of an existing file.

The file to be deleted may not be coupled. If an Adabas expanded file is specified, the complete expanded file (the anchor and all component files) is deleted.

When the DELETE function completes successfully, any locks previously set with the operator commands LOCKU or LOCKF are reset.

Essential Parameter

FILE: File to Be Deleted

FILE specifies the number of the Adabas file to be deleted. Checkpoint, security, trigger, and any other files loaded with the ADALOD utility's SYSFILE option may be specified *only* if ADADBS DELETE is the only Adabas user; deleting any these files automatically causes Adabas to terminate when finished. To delete an Adabas expanded file, specify the file number (also the anchor file).

Optional Parameters

KEEPFDT: Retain the Field Definition Table

The KEEPFDT parameter, if specified, instructs ADADBS DELETE to keep the deleted file's field definition table (FDT) for later use by ADACMP. If this parameter is specified, a file with the same number as the one now being deleted can only be later loaded if either the new file's FDT is the same as that of the deleted file, or the load operation specifies the IGNFDT parameter to accept the new file's FDT.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

PASSWORD specifies the password of the file to be deleted. This parameter is required if the file is password-protected.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Examples

File 6 is to be deleted.

```
ADADBS DELETE FILE=6
```

Password-protected file 10 is to be deleted. The field definition table is to be retained. File number 10 cannot be used again until another ADALOD LOAD command is issued with the IGNFDT option.

```
ADADBS DELETE
FILE=10,KEEPFDT,PASSWORD='FILE10'
```

33 DSREUSE: Reuse Data Storage Blocks

- Essential Parameters 234
- Optional Parameters 234
- Example 291

The DSREUSE function controls the reuse of Data Storage blocks.

```
ADADBS  DSREUSE   FILE = file-number
                  MODE = { ON | OFF }
                  [NOUSERABEND]
                  [PASSWORD = 'password ' ]
                  [RESET]
                  [TEST]
```

Essential Parameters

FILE: File Number

FILE is the number of the file for which the DSREUSE setting is to apply.

Block reuse is originally determined when the file is loaded into the database with the ADALOD FILE function, or when the system file is defined with the ADADEF DEFINE function. In both cases, block reuse defaults to "YES" unless specified otherwise in those functions.

MODE: Reuse Mode

The Data Storage block assignment mode to be in effect. MODE=OFF indicates that Data Storage blocks which become free as a result of record deletion may not be reused, in effect cancelling the ADADBS DSREUSE function. MODE=ON indicates that Data Storage blocks may be reused. The MODE= parameter has no default, and must be specified.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

PASSWORD specifies the file's security password, and is required if the file is password-protected.

RESET: Reset Space Pointer

The RESET parameter causes searches for new Data Storage space to start at the beginning of the file.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

Data Storage blocks for file 6 are not to be reused.

```
ADADBS DSREUSE FILE=6,MODE=OFF
```

34 ENCODEF: Change File Encoding

- Essential Parameter 238
- Optional Parameters 239
- Example 239

```
ADADBS  ENCODEF   FILE = file-number
                  FACODE =  alpha-key | UWCODE = wide-key
                  [NOUSERABEND]
                  [TEST]
```

This chapter describes the syntax, processing, and parameters of the ADADBS ENCODEF function.

Essential Parameter

FILE: File Number

FILE is the number of the file for which encoding is to be changed.

FACODE: Encoding for Alphanumeric Fields in File

The FACODE parameter defines the encoding for alphanumeric fields stored in the file. It can be applied to files already loaded. The encoding must be derived from EBCDIC encoding; that is, X'40' is the space character. Double-byte character set (DBCS) type encodings are supported with the exception of DBCS-only. See *Supplied UES Encodings*, in *Adabas DBA Tasks Manual*, for a list of supplied code pages.

FACODE and UWCODE are mutually exclusive parameters; if one is specified, the other should not be. But one of them must be specified.

UWCODE: User Encoding for Wide-Character Fields in File

The UWCODE parameter defines the user encoding for wide-character fields stored in the file. It can be applied to files already loaded. Note that the wide file encoding is not changed.

To change the encoding of wide-character fields, the file must be unloaded, decompressed, compressed, and reloaded. See *Supplied UES Encodings*, in *Adabas DBA Tasks Manual*, for a list of supplied code pages.

FACODE and UWCODE are mutually exclusive parameters; if one is specified, the other should not be. But one of them must be specified.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See ***Syntax Checking with the TEST Parameter*** for more information about using the TEST parameter in ADADBS functions.

Example

In the following example, ADADBS ENCODEF is used to change the encoding of alphanumeric fields in file 1425 to use code page 285 (CECP: United Kingdom, EBCDIC-compatible with X'40' fill character). In file 401, ADADBS ENCODEF is used to change the encoding of wide fields to use code page 3396 (IBM, CCSID 4396, Japanese host double byte including 1880 user-defined characters). Note that because UWCODE is changing, file 401 must be unloaded, decompressed, compressed, and reloaded.

```
ADADBS ENCODEF FILE=1425,FACODE=285
```

```
ADADBS ENCODEF FILE=401,UWCODE=3396
```

35 INCREASE: Increase Last Associator or Data Storage Data Set Size

- Essential Parameter 292
- Optional Parameters 242
- Example 243
- General Procedure 243
- Operating-System-Specific Procedures 243

The INCREASE function increases the size of the last data set currently being used for the Associator or Data Storage. This function may be executed any number of times for the Associator. The maximum of 99 Data Storage Space Tables (DSSTs) somewhat limits Data Storage increases before all 99 Data Storage extents must be combined into a single extent with either the REORASSO or REORDB function of the ADAORD utility.

Notes:

1. The Associator and Data Storage data set sizes must be increased separately. It is *not* possible to increase both with a single operation.
2. After an INCREASE operation is completed, the INCREASE function automatically ends the current nucleus session. This allows for the necessary Associator or Data Storage formatting with ADAFRM before a new session is started. An informational message occurs to tell you that the nucleus has been stopped.

```
ADADBS  INCREASE  { ASSOSIZE | DATASIZE } = size
                  [ NOUSERABEND ]
                  [TEST]
```

Essential Parameter

ASSOSIZE | DATASIZE: Size to Be Increased

The additional number of blocks or cylinders needed by the Associator or Data Storage data set. To specify blocks, add a "B" after the value; for example, DATASIZE=50B.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

Use the TEST parameter to test the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and vari-

ables. See **Syntax Checking with the TEST Parameter** for more information about using this parameter.

Example

The Associator is to be increased by 400 cylinders.

```
ADADBS INCREASE ASSOSIZE=400
```

General Procedure

The general procedure for increasing the size of the Associator or Data Storage is as follows:

1 Back up the database using the ADASAV utility. This step is optional but recommended.

2 Execute the ADADBS INCREASE function.

3 Format the additional space being added to the data set with the ADAFRM utility.

Operating-System-Specific Procedures

z/OS Systems

Under z/OS, the same data set may be formatted by specifying the DISP=MOD parameter in the JCL. The SPACE parameter for the data set being increased should be set to

```
SPACE=(CYL,(0,n))
```

where *n* is the amount of space (in cylinders) being added. The ADAFRM control statement should also specify the number of cylinders being added. If the increased part of the data set to be formatted is contained on a new volume, the VOL parameter of the JCL must include references to all volumes containing the data set.

Example 1: OS Single-Volume INCREASE

400 cylinders are to be added to an Associator data set which currently contains 300 cylinders. The control statement for the INCREASE function would be:

```
ADADBS INCREASE ASSOSIZE=400
```

The following JCL example increases the Associator data set using ADAFRM:

```
//DDASSOR1  DD
DSN=....,DISP=MOD,SPACE=(CYL,(0,400))
```

and the actual ADAFRM control statement would be

```
ADAFRM ASSOFRM SIZE=400
```

Example 2: OS Multivolume INCREASE

To provide the increase in example 1 for multiple volumes, specify the volumes in the JCS:

```
//DDASSOR1  DD  DSN=...
//
DISP=(MOD,CATLG),VOL=SER=(V1,V2,...),SPACE=(CYL,(0,400))...
```

Include the following step after the INCREASE step but before the FORMAT step to ensure a correct catalog entry:

```
//UNCATLG  EXEC PGM=IEFBR14
//DDASSOR1  DD  DSN=...,DISP=(SHR,UNCATLG)
```

VSE Systems

The following procedures are recommended for increasing Associator or Data storage:

1 Save the current database.

2 End the Adabas session normally with the ADAEND operator command.

3 Update the JCS defining the database to add the new extent on the same volume.

Before a new Associator or Data extent *on either a different or the same VSE volume* can be increased with ADADBS INCREASE and formatted with ADAFRM, that volume's table of contents (VTOC) must be updated to contain the new extent.

Use a job similar to the following example to update the VTOC for a single volume extent:

```
* $$ JOB JNM=jobname
* $$ LST ...
* $$ PCH ...
// ASSGN SYS001,DISK,VOL=volume,SHR
// DLBL ASSOEXT,'dsname',99/365,DA
// EXTENT SYS001,volume1,1,0,starttrack1,trackcount1
// EXTENT SYS001,volume1,1,1,starttrack2,trackcount2

// EXEC ASSEMBLY,GO
```

```
MODVTOC  CSECT
         BALR 9,0
         BCTR 9,0
         BCTR 9,0
         USING MODVTOC,9
         OPEN  ASSOEXT
         CLOSE ASSOEXT
         EOJ   RC=0
ASSOEXT  DTFPH TYPEFLE=OUTPUT,DEVADDR=SYS001,DEVICE=DISK,MOUNTED=ALL
         END
/*
/&
* $$ EOJ
```

For a two-volume extent, use a job similar to the following example:

```
* $$ JOB JNM=jobname
* $$ LST ...
* $$ PCH ...
// ASSGN SYS001,DISK,VOL=volume1,SHR
// ASSGN SYS002,DISK,VOL=volume2,SHR
// DLBL ASSOEXT,'dsname',99/365,DA
// EXTENT SYS001,volume1,1,0,starttrack1,trackcount1
// EXTENT SYS002,volume2,1,1,starttrack2,trackcount2
// EXEC ASSEMBLY,GO
MODVTOC  CSECT
         BALR 9,0
         BCTR 9,0
         BCTR 9,0
         USING MODVTOC,9
         OPEN  ASSOEXT
         CLOSE ASSOEXT
         EOJ   RC=0
ASSOEXT  DTFPH TYPEFLE=OUTPUT,DEVADDR=SYS001,DEVICE=DISK,MOUNTED=ALL
         END
/*
/&
* $$ EOJ
```

Note: This job causes VSE error message 4733D to be sent to the console, and the operator is asked for a response. After the JCS has been validated, the operator response should be `DELETE`.

4 Perform the ADADBS INCREASE operation.

5 Run the new ADAFRM job to format the new extent. The ADAFRM job must specify the FROMRABN parameter, as shown in the following example:

```
ADAFRM ASSOFRM SIZE=size,FROMRABN=rabn-number
```

where *size* is the number of cylinders or blocks by which the data set is to be increased, and *rabn-number* is the first RABN in the new extent.

6 Start the Adabas nucleus.

Note: In a VM environment, certain restrictions apply to multivolume, multiple extent files. If these restrictions are violated, VSE error 4n83I (invalid logical unit) may occur. Refer to the appropriate IBM documentation for more information about these restrictions.

z/VM Systems

To increase the database:

1 Define a new minidisk that is one cylinder (or pseudo-cylinder) larger than the required size.

2 Issue the FORMAT command:

```
FORMATcuu T nnn
```

where *cuu* is the virtual unit address of the new minidisk and *nnn* is "1" for a CKD device or "20" for an FBA device. When prompted for a volume label, you must specify a unique name of up to six alphanumeric characters.

3 Reserve the minidisk with the following command:

```
RESERVE file-name file-type T
```

where *file-name* and *file-type* match the file name and file type used for the file on the primary minidisk.

4 Execute the ADADBS INCREASE utility as described in this document.

5 End the Adabas session with ADAEND.

6 Produce an ADAREP report, and find the first RABN in the new extent. This may be located in the physical layout of the database. The RABN range on this extent indicates VOLSER NUMBER *xxxxxx*.

7 Add CP LINK statements for the new minidisk to the directory or PROFILE EXEC, as required. Update any PROFILE EXECs or CP directory entries for any other virtual machines with multiple write access to this database (for example, the DBA machine).

8 For any EXECs that require it, modify the DATADEF statements for the file. If the standard Software AG EXECs are being used, these DATADEF statements are found in the ADF*nnnnn* EXEC, where *nnnnn* is the five-digit database ID.

To modify the DATADEF statement, locate the line:

```
volx = vol-id
```

where *volx* is "a" for the Associator or "d" for DATA, and *vol-id* is the previous volume list. Change this line to:

```
volx =(vol-id,vo-label)
```

where *vo-label* is the volume label specified while entering the FORMAT command in step 2.

9 Execute the ADAFRM utility for the file as:

```
ADAFRM xxxxFRM     SIZE=size,FROMRABN=rabn-number
```

where

xxxx	is either ASSO or DATA
size	is the size of the minidisk minus one cylinder (or pseudo-cylinder)
rabn-number	is the first RABN on the new extent as shown in the report created in step 6.

BS2000 Systems

Use the following procedure to increase the database on BS2000 systems:

1 Execute ADADBS INCREASE as described in section **General Procedure**.

2 End the Adabas session with `ADAEND`.

3 Produce a database report by running the ADAREP utility. Use the report to find the first RABN for the new extent in the "Physical Layout of the Database" portion of the report. The RABN range is indicated in the "VOLSER NUMBER" column.

4 Increase the data set with the BS2000 `MODIFY-FILE-ATTRIBUTE` command. For example:

```
/MODIFY-FILE-ATTRIBUTE ADA99.ASSO,SUP=PUB(SPACE=RELATIVE(400))
```

Note: In the old ISP format, this was performed by the `FILE` command; for example, `/FILE ADA99.ASSO, SPACE=400`.

5 Format the new space by running the ADAFRM utility. An example for the space added in step 4 is:

```
ADAFRM ASSOFRM   SIZE=400B,FROMRABN=rabn-number
```

where *rabn-number* specifies the first RABN shown on the new extent, as shown in the report.

36 ISNREUSE: Reuse ISNs

- Essential Parameters 250
- Optional Parameters 297
- Example 251

The ISNREUSE function controls whether ISNs of deleted records may be reassigned to new records.

```
ADADBS  ISNREUSE  FILE = file-number
                  MODE = { ON | OFF }
                  [NOUSERABEND]
                  [PASSWORD = 'password ' ]
                  [RESET]
                  [TEST]
```

Essential Parameters

FILE: File Number

FILE is the number of the file for which the ISNREUSE setting is to be changed. The checkpoint file cannot be specified.

MODE: Reuse Mode

MODE causes the ISN reuse mode to be in effect. MODE=OFF causes Adabas not to reuse the ISN of a deleted record for a new record. Each new record will be assigned the next higher unused ISN. MODE=ON indicates that Adabas may reuse the ISN of a deleted record. The MODE parameter has no default; it must be specified.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

PASSWORD specifies the file's security password, and is required if the file is password-protected.

RESET: Reset ISN Pointer

The RESET parameter causes searches for an unused ISN to start at the beginning of the file.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables *cannot* be tested: only the syntax of the specified parameters can be tested. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

ISNs of deleted records in file 7 may be reassigned to new records.

```
ADADBS ISNREUSE FILE=7,MODE=ON
```

37 MODFCB: Modify File Parameters

- Essential Parameter 254
- Optional Parameters 254
- Example 300

The MODFCB function modifies various parameters for a non-system Adabas file.

```
ADADBS  MODFCB    FILE = file-number
                  [ASSOPFAC = new-padding-factor ]
                  [DATAPFAC =  new-padding-factor ]
                  [LOBFILE = file-number ]
                  [MAXDS = maximum-secondary-allocation ]
                  [MAXNI =  maximum-secondary-allocation ]
                  [MAXUI =  maximum-secondary-allocation ]
                  [MAXRECL = maximum-compressed-record-length ]
                  [NOUSERABEND]
                  [PASSWORD = 'password ' ]
                  [PGMREFRESH = { YES | NO } ]
                  [RPLUPDATEONLY = {YES | NO} ]
                  [TEST]
```

Note: If the LOBFILE parameter is specified, only the FILE parameter (which is required), and the optional PASSWORD and TEST parameters can be specified in the same run.

Essential Parameter

FILE: File Number

FILE is the number of the Adabas file to be modified. An Adabas system file cannot be specified. If large object (LB) fields exist in the file, this is the file number of the *base file*.

Optional Parameters

ASSOPFAC/ DATAPFAC: File Padding Factors

ASSOPFAC/DATAPFAC specify the padding factor (1-90) to be in effect for Associator and Data Storage, respectively. Existing blocks retain their original padding factor (see the ADAORD utility).

The Data Storage padding factor (DATAPFAC) setting is ignored for files containing spanned records. The following warning will appear if DATAPFAC is specified for a file that has data storage spanning enabled:

```
*****DATAPFAC is ignored for files with spanned data storage enabled******
```

A return code of 4 (CC=4) is also returned.

LOBFILE: LOB File Number

LOBFILE specifies the file number of the *LOB file* associated with the *base file*. This parameter is useful in combination with the FILE parameter (which is set to the base file number) to mark the two files as being in sync.

Note: If the LOBFILE parameter is specified, only the FILE parameter (which is required), and the optional PASSWORD and TEST parameters can be specified in the same run.

For more information, read *Getting Started with Large Object (LB) Fields*, in *Adabas DBA Tasks Manual*.

MAXDS/ MAXNI/ MAXUI: Maximum Secondary Allocation

The maximum number of blocks per secondary extent allocation for Data Storage (MAXDS), the normal index (MAXNI), and the upper index (MAXUI). The value specified must specify blocks, be followed by a "B" (for example, MAXDS=8000B), and cannot be more than 65535B. If one of the parameters is either not specified or specifies "0B", the maximum secondary extent allocation for that component has no limit.

In all cases, however, Adabas enforces minimum secondary allocations for these parameters:

```
MAXDS=6B
MAXNI=6B
MAXUI=15B
```

If you specify a value lower than these minimum allocations, the Adabas-enforced minimum value is used.

MAXRECL: Maximum Compressed Record Length

The maximum compressed record length permitted for the file. The value specified should not be less than the current maximum record size in the specified file.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

This parameter is required if the file specified in the FILE parameter is password-protected.

PGMREFRESH: Program-generated File Refresh

The PGMREFRESH option determines whether a user program is allowed to perform a file refresh operation by issuing a special E1 command. If the parameter is not specified, the option remains in its current status: either on (YES) or off (NO).

RPLUPDATEONLY: Allow Only Event Replicator Processing Updates

The `RPLUPDATEONLY` parameter can be used in the ADADBS MODFCB function to indicate whether an Adabas database file may be updated only by the Event Replicator Server as part of Adabas-to-Adabas replication or by other means as well. This parameter is optional. Valid values are "YES" or "NO". A value of "YES" indicates that the file can only be updated via Event Replicator processing; a value of NO indicates that the file can be updated by any normal means, including Event Replicator processing. There is no default; if no value is specified for the `RPLUPDATEONLY` parameter in the ADADBS MODFCB function, the value used previously for the file is used.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

The following modifications are to be made for file 203: the Associator padding factor is set to 5, the Data Storage padding factor to 5, and the maximum Data Storage secondary extent allocation to 100 blocks.

```
ADADBS MODFCB
FILE=203,ASSOPFAC=5,DATAPFAC=5,MAXDS=100B
```

38 MUPEX: Set Maximum Count for MU and PE Fields

- Syntax 258
- Essential Parameters 258
- Optional Parameters 259
- Example 259

The MUPEX function is used to specify MU field or PE group limits for a file. This function cannot be run on files that use a PE field in a descriptor nor can it be run on system files. In addition, the maximum number of occurrences is also limited by the maximum size of a compressed record in data storage.

To increase the MU/PE count for a file that contains a PE group in a descriptor, any descriptors having to do with the PE group must first be released (using the ADADBS RELEASE function). After they have been released, the MUPECOUNT parameter can be set to "2" using the MUPEX function and the descriptors then reinverted (using the ADADBS INVERT function).

Syntax

```
ADADBS  MUPEX   FILE = file-number
                MUPECOUNT = { 1 | 2 }
                [PASSWORD = 'password' ]
                [NOUSERABEND]
                [TEST]
```

Essential Parameters

FILE: File Number

FILE specifies the file for which the maximum number of occurrences are to be set. Specify the valid file number of the file in the database for which you want MU and PE limits set.

System files cannot be changed.

MUPECOUNT: Size of the count field

MUPECOUNT sets the maximum number of MU and PE occurrences allowed in the file. Valid values are "1" or "2".

A specification of "1" indicates that no more than 191 MU fields and PE groups will occur in the file; a specification of "2" indicates that the file may contain as many as 65,534 MU fields and PE groups.

To set the MUPECOUNT parameter to "1", the file must either be empty or have no MU or PE fields. To set the MUPECOUNT to "2", the file cannot contain a PE field that is part of a DE field unless the file is empty.

Optional Parameters

NOUSERABEND: Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file for which MU and PE field counts are to be limited. This parameter is required if the file is password-protected. Specify the password between apostrophes ('). In addition, the password must provide update authority for this function to work correctly.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See ***Syntax Checking with the TEST Parameter*** (elsewhere in this section) for more information on using the TEST parameter in ADADBS functions.

Example

In the following example, the maximum number of MU and PE field elements that can be stored in file 17 is 65,534:

```
ADADBS MUPEX FILE=17,MUPECOUNT=2
```

39 NEWFIELD: Add New Field

- Essential Parameter 262
- Optional Parameters 263
- Example 264

The NEWFIELD function adds one or more fields to a file. The new field definition is added to the end of the field definition table (FDT).

Note: Although the definition of a descriptor field is independent of the record structure, note that if a descriptor field is not ordered first in a record and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to order the field first for each record of the file.

NEWFIELD cannot be used to specify actual Data Storage data for the new field; the data can be specified later using Adabas add/update or Natural commands.

When adding a field to an Adabas expanded file, the field must be added to *each individual component file* . Each NEWFIELD operation on a component file returns a message that confirms the change and condition code 4.

```
ADADBS  NEWFIELD  FILE = file-number
                  [FNDEF = 'Adabas-field-definition' ]
                  [NOUSERABEND]
                  [PASSWORD = 'password' ]
                  [SUBFN = 'name  = parent-field (begin , end )' ]
                  [SUPFN = 'name  = { parent-field (begin, end ) } ...' ]
                  [TEST]
```

Essential Parameter

FILE: File Number

FILE specifies the file in which the field to be added is contained. The file may not be an Adabas system file.

Optional Parameters

FNDEF: Adabas Field Definition

FNDEF specifies an Adabas field (data) definition. One FNDEF statement is required for each field to be added. The syntax used in constructing field definition entries is:

```
FNDEF = 'level, name [ , length, format ] [, MU [(occurrences)] ] [ , option ] ... '
FNDEF = 'level, name [ , PE [(occurrences )] ]'
```

Each definition must adhere to the field definition syntax as described for the ADACMP utility in *FNDEF: Field and Group Definition* and *FNDEF: Periodic Group Definition* in the section entitled *Field Definition Statements*, in the ADACMP documentation elsewhere in this guide.

Note the following restrictions:

- A subdescriptor, superdescriptor, hyperdescriptor, or phonetic descriptor definition cannot be specified.
- Text information or sequence numbers are not permitted.
- If you specify an occurrence number when adding an MU or PE field, it is ignored.

The following rules apply when you set the level number in the first FNDEF statement:

1. A level number 01 is always allowed.
2. A level number of 02 or higher means that this field is to be added to an existing group. If so, the following rules apply:
   - The field can be added if the group is a normal (not periodic) group;
   - If the group is a PE group, the field can be added only if the file control block (FCB) for the file does *not* exist; that is, either the file was deleted with the KEEPFDT option, or the FDT was defined using the Adabas Online System Define FDT function, but the Define File function has not yet been run.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

File password. This parameter is required if the file is password-protected.

SUBFN/ SUPFN: Add Subfields or Superfields

These parameters may be used to add subfields and superfields. Each definition must adhere to the definition syntax for sub/superfields as described for the ADACMP utility. Read *COMPRESS: Compress an Adabas File*, elsewhere in this guide for more information.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

Group AB (consisting of fields AC and AX) is to be added to file 24.

```
ADADBS   NEWFIELD    FILE=24
ADADBS               FNDEF='01,AB'
ADADBS               FNDEF='02,AC,3,A,DE,NU'
ADADBS               FNDEF='02,AX,5,P,NU'
```

40 ONLINVERT: Start Online Invert Process

- Essential Parameters 266
- Optional Parameters 267
- Example 268

The ONLINVERT function starts an online invert process.

Note: If the online invert is stopped or suspended, the nucleus releases the partially inverted descriptor when it next starts up. If the ONLINVERT was processing an expanded (chained) file at the time processing was suspended, the nucleus releases the partially inverted descriptor on all component files of the expanded file.

```
ADADBS  ONLINVERT  FILE = file-number
                        { FIELD = 'field-name [ , option, ... ] ' |
                        SUPDE = ' name [, UQ [, XI ] ] =  parent-field (begin, end) , ...' |
                        SUBDE = 'name [, UQ [, XI ] ] =  parent-field (begin, end)' |
                        PHONDE = ' phonde-name (parent-field )' |
                        HYPDE = 'nr , name ,length, format  [ , options] = parent-field, ...' |
                        COLDE = 'nr , name [ , UQ [, XI ] ] = parent-field ' }
                   [CODE = cipher-code ]
                   [PASSWORD = 'password ' ]
                   [NOUSERABEND]
                   [TEST]
                   [WAIT]
```

Essential Parameters

FILE: File Number

File is the number of the file for which the new descriptor is to be created. If a component file of an expanded file chain is specified, the descriptor is added to all component files of that chain.

FIELD/ SUBDE/ SUPDE/ PHONDE/ HYPDE/ COLDE: Define Descriptor

Exactly one of these parameters must be used to define the type of descriptor to be inverted. Only one descriptor per file can be inverted at a time using the online invert function.

Use the FIELD parameter to define a field as descriptor; use the COLDE parameter for a collation descriptor; the HYPDE parameter for a hyperdescriptor; PHONDE for a phonetic descriptor; SUBDE for a subdescriptor; and SUPERDE for a superdescriptor.

FIELD specifies an existing field to be inverted. The field may be an elementary or multiple-value field and may be contained within a periodic group (unless the field is defined with the FI option).

If the descriptor is to be unique, specify "UQ" following the field name. A field in a periodic group cannot be defined as a unique descriptor. If the uniqueness of the descriptor is to be determined with the index (occurrence number) excluded, specify "XI" as well.

When inverting a sub- or superfield, the respective SUBDE or SUPDE parameter must specify the same parent fields that were specified when the field was created; otherwise, an error occurs. Begin and end values are taken from the original field definitions.

If a parent field with the NU option is specified, no entries are made in the inverted list for those records containing a null value for the field. For super- and hyperdescriptors, this is true regardless of the presence or absence of values for other descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

See the **ADACMP utility** description for detailed information about the individual descriptor syntax, subparameter values, and coding.

Optional Parameters

CODE: Cipher Code

If the file specified with the FILE parameter is ciphered, an appropriate cipher code must be supplied using the CODE parameter.

PASSWORD: File Password

If the file specified with the FILE parameter is security-protected, the file's password must be supplied using the PASSWORD parameter.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT: Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process to make field AA of file 10 a descriptor, without waiting for the end of this process.

```
ADADBS ONLINVERT FILE=10,FIELD=AA
```

41 ONLREORFASSO: Start Online Reorder Associator for Files

- Essential Parameters 270
- Optional Parameters 271
- Example 271

The ONLREORFASSO function starts an online process to reorder the Associator of specified files.

Notes:

1. The online reorder process does not change the existing file extents but only reorganizes the file's index within these extents.
2. The online index reorder process does not move index elements out of blocks that are full (according to the Asso padding factor); it only moves elements into blocks that are not full.
3. Released index blocks are not put into the unused RABN chain.
4. This function is not available in the Adabas Cluster Services or Adabas Parallel Services environments.

```
ADADBS  ONLREORFASSO  FILE = file-number
                      [ASSOPFAC = asso-padding-factor]
                      [PASSWORD = password ]
                      [NOUSERABEND]
                      [TEST]
                      [WAIT]
```

Essential Parameters

FILE: File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFASSO operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Associator is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

ASSOPFAC: Associator Padding Factor

ASSOPFAC defines the Associator block padding factor, which is the percentage of each Associator block *not* used during the reorder process. Specify a value in the range 1-90. The number of bytes free after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current padding factor in effect for the file is used.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility does *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT: Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Associator of file 10 first and then file 11. The Associator padding factor of file 11 is to be 5 percent.

```
ADADBS ONLREORFASSO FILE=10
ADADBS              FILE=11,ASSOPFAC=5
```

42 ONLREORFDATA: Start Online Reorder Data for Files

- Essential Parameters 274
- Optional Parameters 274
- Example 330

The ONLREORFDATA function starts an online process to reorder the Data Storage of specified files.

Notes:

1. The online reorder process does not change the existing file extents but only reorganizes the file's Data Storage records within these extents.
2. This function is not available in the Adabas Cluster Services or Adabas Parallel Services environments.

```
ADADBS  ONLREORFDATA  FILE = file-number
                      [DATAPFAC =  data-padding-factor ]
                      [SORTSEQ = { ISN | de-name | physical-sequence  } ]
                      [PASSWORD = password ]
                      [NOUSERABEND]
                      [TEST]
                      [WAIT]
```

Essential Parameters

FILE: File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFDATA operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Data Storage is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the Data Storage padding factor. The number specified represents the percentage of each Data Storage block that remains unused when the file is reordered. A value in the range 1-90 may be specified (see **ADALOD utility** for additional information about setting and using the Data Storage padding factor).

If this parameter is omitted, the current padding factor in effect for the file is used.

SORTSEQ: File Reordering Sequence

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence.

Note: Records within a single Data Storage block are not sorted according to the specified sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

If ISN is specified, the file is processed in ascending ISN sequence.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility does *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT: Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Data Storage of file 10 first, and then file 11. The Data Storage padding factor of file 11 is to be 5 percent.

```
ADADBS ONLREORFDATA FILE=10
ADADBS              FILE=11,DATAPFAC=5
```

43 ONLREORFILE: Start Online Reorder Associator and Data for Files

- Essential Parameters 330
- Optional Parameters 279
- Example 280

The ONLREORFILE function starts an online process to reorder the Associator and Data Storage of specified files.

Notes:

1. The online reorder process does not change the existing file extents but only reorganizes the file's index and Data Storage records within these extents.
2. This function is not available in the Adabas Cluster Services or Adabas Parallel Services environments.
3. Released index blocks are not put into the unused RABN chain.

```
ADADBS  ONLREORFILE     FILE = file-number
                        [ASSOPFAC = asso-padding-factor]
                        [DATAPFAC = data-padding-factor ]
                        [SORTSEQ = { ISN | de-name | physical-sequence  } ]
                        [PASSWORD = password ]
                        [NOUSERABEND]
                        [TEST]
                        [WAIT]
```

Essential Parameters

FILE: File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFILE operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Associator and Data Storage is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

ASSOPFAC: Associator Padding Factor

ASSOPFAC defines the new Associator block padding factor, which is the percentage of each Associator block *not* used during the reorder process. Specify a value in the range 1-90. The number of bytes free after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current padding factor in effect for the file is used.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor. The number specified represents the percentage of each Data Storage block that remains unused when the file is reordered. A value in the range 1-90 may be specified (see the **ADALOD utility** for additional information about setting and using the Data Storage padding factor).

If this parameter is omitted, the current padding factor in effect for the file is used.

SORTSEQ: File Reordering Sequence

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence.

Note: Records within a single Data Storage block are not sorted according to the specified sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

If ISN is specified, the file is processed in ascending ISN sequence.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility does *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT: Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Associator and Data Storage of file 10 first, and then file 11. The Associator padding factor of file 10 is to be 5 percent; the Data Storage padding factor of file 11 is to be 10 percent.

```
ADADBS ONLREORFILE FILE=10,ASSOPFAC=5
ADADBS             FILE=11,DATAPFAC=10
```

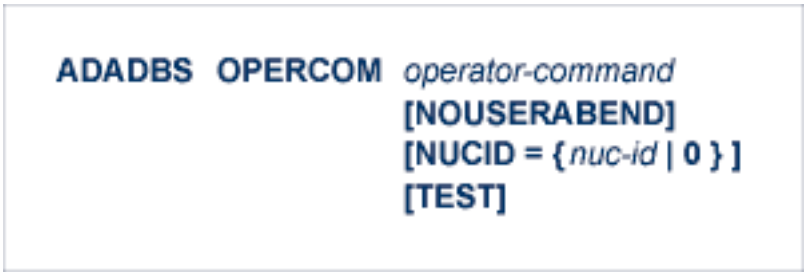
44 OPERCOM: Issue Adabas Operator Commands

- Using OPERCOM Commands in Cluster Environments 282
- Optional Parameters 282
- Operator Commands 283

The OPERCOM function issues operator commands to the Adabas nucleus.

In an Adabas cluster environment, OPERCOM commands can be directed to a single cluster nucleus or to all active nuclei in the cluster. If a particular nucleus is not specified, the command defaults to the local nucleus.

Adabas issues a message to the operator, confirming command execution.



In this section, the discussion of the individual operator commands follows the discussion of the optional parameters, since some of the operator commands behave differently when issued in an Adabas cluster environment.

Using OPERCOM Commands in Cluster Environments

Some ADARUN parameters are *global parameters*; that is, they must have the same values for all nuclei in a cluster. Of these, some are set at session initialization and cannot be changed. Others can be modified on a running system. OPERCOM commands that change these modifiable global parameter values are handled in a special way in cluster environments.

If an Adabas cluster nucleus changes one or more global parameters, that nucleus acquires a *parameter change lock*, makes the changes in its local parameter area, informs the other cluster nuclei of the changes and waits for a reply. The other cluster nuclei make the changes in their own local parameter areas and send an acknowledgment message.

Optional Parameters

GLOBAL: Operate Across All Active Cluster Nuclei

Five OPERCOM commands use the GLOBAL option to operate across all active nuclei in a cluster: ADAEND, CANCEL, FEOFCL, FEOFPL, and HALT. For example:

```
ADADBS OPERCOM ADAEND, GLOBAL
```

All other OPERCOM commands use the NUCID=0 option to operate across all active nuclei in a cluster.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUCID: Cluster Nucleus ID

Any nucleus running in an Adabas nucleus cluster is allowed to run Adabas utilities such as ADADBS.

With certain exceptions, the NUCID parameter allows you to direct the ADADBS OPERCOM commands to a particular nucleus in the cluster for execution, just as though the command had been issued by a locally run ADADBS OPERCOM operation. You can route most OPERCOM commands to all nuclei in a cluster by specifying NUCID=0.

If NUCID is not specified in a cluster environment, the command is routed to the local nucleus.

Note: For ADADBS OPERCOM and Adabas Online System (AOS), a zero value for the NUCID parameter indicates that the command applies to all nuclei in the cluster (global). A nonzero value for the NUCID parameter indicates that the command applies only to the cluster nucleus specified.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; nor the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Operator Commands

The following Adabas operator commands can be used in the ADADBS OPERCOM utility function:

- **ADAEND [, GLOBAL]**

  This command terminates an Adabas session normally. No new users are accepted after this command has been issued. ET logic updating is continued until the end of the current logical transaction for each user. After all activity has been completed as described above, the Adabas session is terminated.

In nucleus cluster environments, the GLOBAL option can be used to terminate the Adabas session in all active cluster nuclei.

- **ALOCKF = *file-number***

  Lock a file in advance to ensure that an EXU, EXF, or UTI user will have exclusive control of the specified file. The advance-lock prevents new transactions from using the file. Once all current users have stopped using the file, the exclusive-control user has the lock. Until then, the exclusive-control user must wait.

  To remove the advance lock without running the utility, see the RALOCKF command.

  This command is not available

  - in single user mode; or
  - for a read-only nucleus.

- **CANCEL [, GLOBAL]**

  Cancel the Adabas session immediately. All command processing is immediately suspended. A pending autorestart is in effect which in turn causes the autorestart routine to be executed during the initialization of the next Adabas session.

  In nucleus cluster environments, the GLOBAL option can be used to cancel the Adabas session in all active cluster nuclei.

- **CLOGMRG = { YES | NO }**

  Switches automatic command log merging (ADARUN CLOGMRG parameter value) on or off in nucleus cluster environments.

  The CLOGMRG command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

CLUFREEUSER = [, TNA = *max-time*] **[, UID =** *userid*] **[, FORCE] [, GLOBAL] [, NUCID =** *nucid*]

Note: The CLUFREEUSER command is only valid in cluster environments. It can be issued against the local nucleus only or, with the GLOBAL option, against all active and inactive nuclei in the cluster.

Delete leftover user table elements (UTEs) in common storage that are no longer associated with user queue elements (UQEs) in a nucleus where:

Parameter	Is...
TNA	A decimal number specifying the timeout value in seconds. UTEs that are not used during the time specified may be deleted if other conditions are fulfilled. If TNA= is not specified, UTEs may be deleted without regard to their recent use.
UID	A character string or hexadecimal byte string, as follows: ■ *cccccccc* where the argument is 1-8 letters, digits, or embedded '-' signs without surrounding apostrophes. ■ '*cccccccc*' ↩ where the argument is 1-8 characters with surrounding apostrophes. ■ X'*xxxxxxxxxxxxxxxx*' ↩ where the argument is an even number of 2-16 hexadecimal digits preceded by an "X" and enclosed in single quotation marks. A character string must be enclosed in apostrophes if it contains characters other than letter, digits, or embedded '-' signs. If a specified character string is less than 8 characters long, it is implicitly padded with blanks. If a specified hexadecimal string is shorter than 16 hexadecimal digits, it is implicitly padded with binary zeros. If the UID string ends with a '*' (or X'5C'), the preceding characters are treated as a user ID prefix: only the prefix is needed for an argument match. If the last 8 bytes of a user's 28-byte communication ID match a specific user ID or user ID prefix, that user's UTE may be deleted if other conditions are fulfilled. If UID= not specified, UTEs may be deleted regardless of their user IDs.
FORCE	Indicates that leftover UTEs are to be deleted even if the users are due a response code 9, subcode 20. If FORCE is not specified, such UTEs are not deleted. Before using the FORCE parameter, ensure that the users owning the UTEs to be deleted will not expect any of their transactions to remain open.
GLOBAL	Indicates that leftover UTEs throughout the Adabas cluster are to be deleted if they are no longer associated with UQEs and are eligible according to the other specified parameters. Additionally and subject to the other rules, leftover UTEs are deleted if their assigned nuclei have terminated since their last use. If GLOBAL is not specified, only UTEs assigned to the local nucleus and used since the nucleus start are eligible for deletion.
NUCID	Identifies the ID of the specific nucleus that should be used to process the command.

- **CT = *timeout-limit***

  Dynamically override the ADARUN CT parameter value; that is, the maximum number of seconds that can elapse from the time an Adabas command has been completed until the results are returned to the user through interregion communication (which depends on the particular operating system being used). The minimum setting is 1; the maximum is 16777215.

  In nucleus cluster environments, the CT command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

- **DAUQ**

  Display the user queue element (UQE) of each user who has executed at least one Adabas command within the last 15 minutes.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DCQ**

  Display all posted command queue elements (CQEs). Each CQE's user ID, job name, and buffer length is displayed.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DDIB**

  Display data integrity block (DIB). This block contains entries indicating which Adabas utilities are active and the resources being used by each utility. The DDIB function can be performed with either an active or an inactive nucleus.

  In nucleus cluster environments, the information displayed by the DDIB command is global; the command can be run on any nucleus.

- **DDSF**

  Display Adabas Delta Save Facility (DSF) status. The Adabas nucleus displays the DSF status on the operator console as well as in the ADADBS job protocol.

  This function is only available if the nucleus is run with the parameter ADARUN DSF=YES.

  In nucleus cluster environments, the information displayed by the DDSF command is global; the command can be run on any nucleus.

- **DFILES= { *n* | *n1* ,..., *n5* }**

  Displays the number of access, update, EXU, and UTI users for the specified files. User types are totaled for each file, and are listed by file. Up to five files can be specified in this command. Up to 798 users are displayed.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DFILUSE = *file-number***

  Displays the count of commands processed for the specified file so far during the current session.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DHQA**

  Display up to 1000 hold queue elements (HQEs).

- **DLOCKF**

  Display locked files.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DNC**

  Display the number of posted command queue elements (CQEs) waiting to be selected.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DNH**

  Display the number of ISNs currently in the hold queue.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DNU**

  Display the number of current users.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DONLSTAT**

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Display status of each active reorder or invert online process together with the process ID.

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DPARM**

  Display the Adabas session parameters currently in effect.

- **DRES**

  Display the allocated pool space and the highest use level ('high water mark') reached so far during the current session by record count and by percent for the following resources:

  - attached buffers (AB) - current allocation not supported
  - command queue (CQ)
  - format pool (FP)
  - hold queue (HQ)
  - pool for the table of ISNs (TBI)
  - pool for the table of sequential commands (TBQ or TBLES)
  - user queue (UQ)
  - unique descriptor pool (DUQPOOL)
  - security pool
  - user queue file list pool
  - work pool (WP)
  - pool for global transaction IDs (XIDs; nonzero only with Adabas Transaction Manager)
  - cluster block update redo pool (nonzero only for a cluster nucleus with ADARUN LRDP greater than zero)

  The actual values are displayed in nucleus message ADAN28 described in the *Adabas Messages and Codes* documentation.

- **DSTAT**

  Display the current Adabas nucleus operating status.

- **DTH**

  Display thread status.

- **DUQ**

  Display up to five active and inactive user queue elements.

- **DUQA**

  Display all user queue elements (UQEs).

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **DUQE = X'***user-id*'

  Display a user queue element for the specified Adabas-assigned user ID as follows:

  ```
  DUQE=X'A3CF2'
  ```

  The user ID must be entered in hexadecimal format. Do not use a job name for the user ID.

  In nucleus cluster environments, NUCID must always be specified because the user ID is not unique to the cluster.

- **DUUQE**

  Display utility user queue elements (UQEs).

  In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

- **FEOFCL [, GLOBAL]**

  Close the current dual or multiple Command log and switch to the other dual or another multiple Command log. This command is valid only if dual or multiple command logging is in effect.

  In nucleus cluster environments, the GLOBAL option can be used to switch the dual or multiple command log in all cluster nuclei at the same time.

- **FEOFPL [, GLOBAL]**

  Close the current dual or multiple data Protection log and switch to the other dual or another multiple Protection log. This command is valid only if dual or multiple data protection logging is in effect.

  In nucleus cluster environments, the GLOBAL option can be used to switch the dual or multiple protection log in all cluster nuclei at the same time.

- **HALT [, GLOBAL]**

  Stop Adabas session. A BT (backout transaction) command is issued for each active ET logic user. The Adabas session is then terminated; no dumps are produced.

  In nucleus cluster environments, the GLOBAL option can be used to halt the Adabas session in all active cluster nuclei.

- **LOCKF =** *file-number*

  Lock the specified file. The specified file will be locked at all security levels.

- **LOCKU =** *file-number*

  Lock the specified file for all non-utility use. Adabas utilities can use the file normally.

- **LOCKX = *file-number***

  Lock the specified file for all users except EXU or EXF users. EXU and EXF users can use the file normally. The lock is released automatically when an EXU user issues an OP command.

- **LOGGING**

  Start command logging.

- **LOG*xx***

  Begin logging as indicated by *xx* for each command logged where *xx* is one of the following:

  - CB -
  - FB - the Adabas format buffer
  - IB - the Adabas ISN buffer
  - IO - Adabas I/O activity
  - RB - the Adabas record buffer
  - SB - the Adabas search buffer
  - UX - user data passed in the seventh parameter of the Adabas parameter list
  - VB - the Adabas value buffer

- **NOLOGGING**

  Stop or prevent command logging.

- **NOLOG*xx***

  Stop or prevent logging of *xx* where *xx* is one of the following:

  - CB - the Adabas control block
  - FB - the Adabas format buffer

- IB - the Adabas ISN buffer
- IO - Adabas I/O activity
- RB - the Adabas record buffer
- SB - the Adabas search buffer
- UX - user data passed in the seventh parameter of the Adabas parameter list
- VB - the Adabas value buffer

- **ONLRESUME = X'***identifier*'

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Resume a previously suspended online reorder or invert process.

  In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

- **ONLSTOP = X'***identifier*'

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Stop an online reorder or invert process cleanly. The process continues up to its next interrupt point in order to produce a consistent state, and then terminates after performing all necessary cleanup.

  In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

- **ONLSUSPEND = X'***identifier*'

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Suspend an online reorder or invert process. The process continues up to its next interrupt point in order to produce a consistent state, performs a command throwback, and enters a state where it cannot be selected for processing. This command is useful if the online process is consuming too much of the nucleus resources.

In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

- **RALOCKF = *n***

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Remove the advance lock on the specified file (see ALOCKF command) without running the utility.

- **RALOCKFA**

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Remove the advance lock on all files for which it has been set (see ALOCKF command) without running the utility.

- **RDUMPST**

  Terminate online dump status. This command is normally used if online execution of the ADASAV utility has terminated abnormally.

- **READONLY = { YES | NO }**

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Switches READONLY status on or off.

  In nucleus cluster environments, the READONLY command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

REVIEW = { NO | LOCAL | *hub-id* }

Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

Deactivate Adabas Review; change from hub mode to local mode; specify or change the Adabas Review hub with which a nucleus communicates.

STOPF = *file-number* [, PURGE]

Stop all users who are using the specified file. Any open transactions of the stopped users are backed out. Unless PURGE is also specified, a stopped user who returns (by sending a command) receives response code 9.

If the optional PURGE parameter is specified, the stopped users are also deleted (their user queue elements are removed from the user queue).

This command does not stop EXF or UTI users.

The following is an example of using the PURGE parameter:

```
ADADBS OPERCOM STOPF=5,PURGE
```

Caution: If Adabas is running with ADARUN OPENRQ=NO (specifying that users are not required to issue an OP as the first command of the session), run the STOPF command with `PURGE` only if you are certain that the users to be deleted are no longer active. If a user with an open transaction is deleted, but then returns (by sending a command), no indication is given about the transaction backout. If the user continues the transaction, logical inconsistencies in the database could occur.

STOPI = *time* [, PURGE]

Stop all users who have not executed a command during the specified time interval (in seconds). Any open transactions of the stopped users are backed out. Unless PURGE is also specified, a stopped user who returns (by sending a command) receives response code 9.

This command does not stop EXF or UTI users.

If the optional PURGE parameter is specified, the stopped users are also deleted (their user queue elements are removed from the user queue).

The following is an example of using the PURGE parameter:

```
ADADBS OPERCOM STOPI=3600,PURGE
```

Caution: If Adabas is running with ADARUN OPENRQ=NO (specifying that users are not required to issue an OP as the first command of the session), run the STOPI command with `PURGE` only if you are certain that the users to be deleted are no longer active. If a user with an open transaction is deleted, but then returns (by sending a command), no indication is given about the transaction backout. If the user continues the transaction, logical inconsistencies in the database could occur.

Stop and delete the user with the Adabas-assigned user ID (in the form shown in the display commands), or stop and delete all users with the specified job name (*job-name*). Any open transaction by the stopped users will be backed out.

Caution: If Adabas is running with ADARUN OPENRQ=NO (specifying that users are not required to issue an OP as the first command of the session), run the STOPU command only if you are certain that the users to be deleted are no longer active. If a user with an open transaction is deleted, but then returns (by sending a command), no indication is given about the transaction backout. If the user continues the transaction, logical inconsistencies in the database could occur.

Note: The `STOPU=X'userid'` command is not allowed for online reorder or invert processes. See the `ONLSTOP=X'identifier'` command instead.

The user ID must be specified in hexadecimal format; for example:

```
STOPU=X'1CF2' ↩
```

In a cluster environment, NUCID must always be specified because the user ID is not unique to the cluster.

Force resynchronization of all ET users on the nucleus. The nucleus waits for all ET users to reach ET status before continuing.

- **TNA***u* = *time*

  Set non-activity time limit (in seconds) for users where *u* is one of the following:

  - A - for access-only (ACC) users
  - E - for ET logic users
  - X - for exclusive control (EXF/EXU) users

  If specified, *time* must be a value greater than zero; it overrides the ADARUN value.

  In nucleus cluster environments, the TNAu commands are global by definition and affect all nuclei in the cluster. If a NUCID is specified, it is ignored.

- **TT** = *time*

  Set transaction time limit (in seconds) for ET logic users. If specified, this value must be greater than zero; it overrides the ADARUN value. In nucleus cluster environments, the TT command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

- **UNLOCKF** = *file-number*

  Unlock the specified file and restore its usage to the prelocked status.

- **UNLOCKU** = *file-number*

  Unlock the specified file for utility use and restore it to its prelocked status for non-utility users.

- **UNLOCKX** = *file-number*

  Unlock the specified file and restore its usage to the prelocked status.

- **UTIONLY = { YES | NO }**

  Note: Not currently available for use with Adabas Parallel Services cluster nuclei.

  Switch UTIONLY status on or off.

  In nucleus cluster environments, the UTIONLY command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

45 PRIORITY: Change User Priority

- Essential Parameter 300
- Optional Parameters 300
- Example 301

The PRIORITY function may be used to set or change the Adabas priority of a user. A user's priority can range from 0 (the lowest priority) to 255 (the highest priority).

The user is identified by the same user ID provided in the Adabas control block (OP command, Additions 1 field).

```
ADADBS  PRIORITY  USERID = 'user-id'
                  [NOUSERABEND]
                  [PRTY = { n | 255 }]
                  [TEST]
```

Essential Parameter

USERID: User ID

The user ID in the checkpoint file of the user for which priority is to be changed. If a record for this user does not exist, a new one is added to the checkpoint file.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PRTY: User Priority

The priority to be in effect for the user. A value in the range 0 for lowest priority to 255 for the highest priority may be specified. The default is 255. This value will be added to the operating system priority by the interregion communications mechanism.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

```
ADADBS PRIORITY USERID='USER24',PRTY=7
```

Set the priority assignment for the user with the user ID "USER24" to "7".

46 RECORDSPANNING: Enable or Disable Record Spanning

- Syntax 304
- Essential Parameters 304
- Optional Parameters 304
- Example 305

The RECORDSPANNING function is used to enable or disable record spanning for a file. When record spanning is enabled, the size of compressed records in a file may exceed the maximum data storage block size.

Syntax

```
ADADBS  RECORDSPANNING  FILE = file-number
                        MODE = { ON | OFF }
                        [PASSWORD = 'password' ]
                        [NOUSERABEND]
                        [TIMELIMIT = { 60  |  timelimit } ]
                        [TEST]
```

Essential Parameters

FILE: File Number

FILE specifies the number of the file for which you want to enable or disable record spanning. Specify the valid file number of the file in the database for which you want to control record spanning.

MODE: Turn Record Spanning On or Off

MODE indicates whether record spanning should be activated for the file or not. Valid values are "ON" and "OFF". A value "ON" turns record spanning on; a value of "OFF" turns record spanning off.

Optional Parameters

NOUSERABEND: Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file for which record spanning is to be enabled or disabled. This parameter is required if the file is password-protected. Specify the password between apostrophes ('). In addition, the password must provide update authority for this function to work correctly.

TIMELIMIT: Number of Seconds

The maximum number of seconds the ADADBS RECORDSPANNING function may run. If record spanning is being turned off for a file, Adabas must verify that no spanned records exist in the file. Since this analysis may take some time, you can use the TIMELIMIT parameter to indicate how long you are willing to wait for the ADADBS RECORDSPANNING function to run. If the file scan takes more time than specified by the TIMELIMIT parameter, the function will be aborted and appropriate error messages are issued.

The default is 60 seconds.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See *Syntax Checking with the TEST Parameter* (elsewhere in this section) for more information on using the TEST parameter in ADADBS functions.

Example

In the following example, record spanning is turned on for file 17. The records in file 17 can therefore exceed the boundaries of a data storage block.

```
ADADBS RECORDSPANNING MODE=ON,FILE=17
```

47 RECOVER: Recover Space

- Optional Parameters 308

The RECOVER function recovers allocated space by rebuilding the free space table (FST). The RECOVER function subtracts file, DSST, and alternate RABN extents from the total available space.

```
ADADBS  RECOVER  [NOUSERABEND]
                 [TEST]
```

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

48 REFRESH: Set File to Empty Status

- Essential Parameter 310
- Optional Parameters 310
- Example 311

The REFRESH function sets the file to 0 records loaded, sets the first extent for the address converter, Data Storage, normal index, and upper index to *empty* status, and deallocates other extents.

When the REFRESH function completes successfully, any locks previously set with the operator commands LOCKU or LOCKF are reset.

```
ADADBS  REFRESH  FILE = file-number
                 [NOUSERABEND]
                 [PASSWORD = 'password ' ]
                 [TEST]
```

Essential Parameter

FILE: File Number
FILE specifies the file that is to be set to *empty* status.

Optional Parameters

NOUSERABEND: Termination Without Abend
When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password
This parameter is required if the file is password-protected.

TEST: Test Syntax
This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

File 116 is to be set to empty status.

```
ADADBS REFRESH FILE=116
```

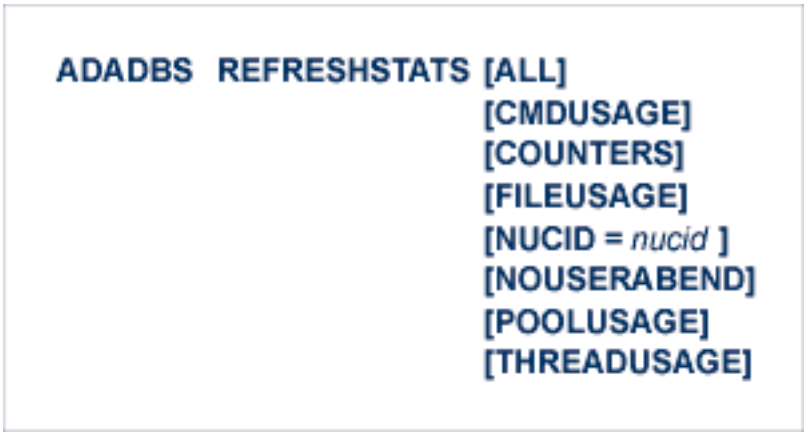
49 REFRESHSTATS: Reset Statistical Values

- Optional Parameters 387
- Example 315

The REFRESHSTATS function resets statistical values maintained by the Adabas nucleus for its current session. Parameters may be used to restrict the function to particular groups of statistical values.

When you invoke REFRESHSTATS, Adabas automatically writes the nucleus shutdown statistics to DD/PRINT.

Important: Refreshing Adabas statistical values affects the corresponding Adabas Statistics Facility (ASF) field values. These values, which normally reflect the period from the start of the nucleus, will then refer to the time after the last refresh. ASF users may therefore find it useful to store the nucleus records with the appropriate ASF function before refreshing the values.



Optional Parameters

ALL: All Statistical Values

The ALL keyword may be specified as an abbreviation for the combination of CMDUSAGE, COUNTERS, FILEUSAGE, POOLUSAGE, and THREADUSAGE.

If none of the option keywords is specified, ALL is the default option.

CMDUSAGE: Command Usage Counters

The CMDUSAGE parameter is specified to reset the counters for Adabas direct call commands such as L*x*, S*x*, or A1.

COUNTERS: Frequency Counters

The COUNTERS parameter is specified to reset the counter fields for local or remote calls, format translations, format overwrites, Autorestarts, protection log switches, buffer flushes, and command throw-backs.

FILEUSAGE: Count of Commands Per File

The FILEUSAGE parameter is specified to reset the count of commands for each file.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUCID: Cluster Nucleus ID

Any nucleus running in an Adabas nucleus cluster is allowed to run Adabas utilities such as ADADBS. The NUCID parameter allows you to direct the ADADBS REFRESHSTATS function to a particular nucleus in the cluster for execution, just as though the command had been issued by a locally run ADADBS REFRESHSTATS operation.

If you specify NUCID=0, the statistical values are refreshed for all active nuclei in the cluster.

POOLUSAGE: High-Water Marks for Nucleus Pools

The POOLUSAGE parameter is specified to reset the high-water marks for the nucleus pools such as the work pool, the command queue, or the user queue.

THREADUSAGE: Count of Commands Per Thread

The THREADUSAGE parameter is specified to reset the count of commands for each Adabas thread.

Example

```
ADADBS  REFRESHSTATS
CMDUSAGE,POOLUSAGE,NUCID=3
```

After the shutdown statistics for the Adabas cluster nucleus with NUCID=3 are written to DD/PRINT, the command counters and the pool high-water marks for the nucleus are reset.

50 RELEASE: Release Descriptor

- Essential Parameters 318
- Optional Parameters 318
- Example 319

The RELEASE function releases a descriptor from the descriptor space.

This function results in the release of all space currently occupied in the Associator inverted list for this descriptor. This space can then be reused for this file by reordering or ADALOD UPDATE. No changes are made to Data Storage.

When releasing descriptor space for an Adabas expanded file, perform the RELEASE function for *each individual component file* of the expanded file. Each RELEASE operation on a component file causes a message that confirms the change, and returns condition code 4.

```
ADADBS  RELEASE  FILE = file-number
                 DESCRIPTOR = 'name'
                 [NOUSERABEND]
                 [PASSWORD = 'password' ]
                 [TEST]
```

Essential Parameters

FILE: File Number

FILE specifies the file that contains the descriptor to be released. The file cannot be an Adabas system file.

DESCRIPTOR: Descriptor to Be Released

DESCRIPTOR specifies the descriptor to be released. Any descriptor type can be specified. A descriptor currently being used as the basis for file coupling cannot be specified. If the descriptor being released is an ADAM descriptor, the file is no longer processed as an ADAM file.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

This parameter is required if the file is password-protected. Specify the password between apostrophes (').

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

```
ADADBS RELEASE FILE=31,DESCRIPTOR='AA'
```

Descriptor AA in file 31 is released from descriptor status.

51 RENAME: Rename File or Database

- Essential Parameter 393
- Optional Parameters 322
- Examples 323

The RENAME function may be used to change the name assigned to a file or database.

```
ADADBS  RENAME   NAME = 'name'
                 [FILE = file-number ]
                 [NOUSERABEND]
                 [PASSWORD = 'password ' ]
                 [TEST]
```

Essential Parameter

NAME: New File Name

NAME is the new name to be assigned to the file. It is specified between apostrophes (for example, 'RESERVATIONS'). A maximum of 16 characters can be used.

Optional Parameters

FILE: File Number

FILE is the number of the file to be renamed: if specified as zero or omitted, the database is renamed.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file. This parameter is required if the file is password-protected.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Examples

The name of file 2 is to be changed to "INVENTORY".

```
ADADBS RENAME FILE=2,NAME='INVENTORY'
```

The database is renamed to "RESERVATIONS".

```
ADADBS RENAME  NAME='RESERVATIONS',FILE=0
```

52 RENUMBER: Change File Number

- Essential Parameter 326
- Optional Parameter 326
- Example 327

The RENUMBER function changes the number of an Adabas file.

```
ADADBS  RENUMBER  FILES = current-number, new-number
                  [NOUSERABEND]
                  [TEST]
```

Essential Parameter

FILES: Current File Number, New File Number

The number currently assigned to the file, and the new number to be assigned to the file. If the new number is assigned to another file, the RENUMBER function will not be performed.

An Adabas system file cannot be used. The file may not be security-protected, may not be coupled to another file, and may not be part of an expanded file.

Optional Parameter

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

The file number for file 4 is to be changed to 40.

```
ADADBS RENUMBER FILES=4,40
```

53 REPLICATION: Activating or Deactivating Replication

- Essential Parameter 330
- Optional Parameter 330
- Examples 332

The REPLICATION function activates or deactivates replication for an Adabas database file. This function should be used with Adabas databases; it is not valid with Event Replicator databases.

```
ADADBS  REPLICATION  FILE = file-number
                     {ON | OFF | MODIFY [ACTIVATE | DEACTIVATE] }
                     [DSBI = { ON | OFF } ]
                     [KEY = field | NOKEY]
                     [TARGET = dbid ]
```

Essential Parameter

FILE: File Number

The file number of the database file for which replication is to be activated or deactivated. This parameter is required; there is no default.

Note: Replication may not be turned on for an Adabas system file or for a ciphered file.

ON, OFF, MODIFY: Set Replication

One of the parameters `ON`, `OFF`, or `MODIFY` must be set. There is no default. Each parameter is described below:

- `ON` turns replication on for the specified file.
- `OFF` turns replication off for the specified file.
- `MODIFY` allows you to modify one or more of the replication-related values (KEY, TARGET, DSBI) for a replicated file. The `ACTIVATE` and `DEACTIVATE` parameters allow you to change the replication status for a replicated file.

Optional Parameter

ACTIVATE: Activate Replication

The `ACTIVATE` parameter is only allowed when MODIFY is specified. It will activate replication for a file that has replication defined. Replication must already be inactive for the file.

DEACTIVATE: Deactivate Replication

The `DEACTIVATE` parameter is only allowed when MODIFY is specified. It will deactivate replication for a file that has replication defined. Replication must already be active for the file.

DSBI: Define Before Image

The DSBI parameter defines whether or not before images of data storage are collected for replication during the update of a record on a file. Following are rules for setting the DSBI parameter:

- Possible values: "ON", "OFF"
- The parameter is optional when "ON" or "MODIFY" is specified.
- The parameter has a default value of "ON" when ADADBS REPLICATION ON is specified.
- The parameter is not allowed when ADADBS REPLICATION OFF is specified.

For more information about how this parameter is used in Adabas database processing during replication, read *Nucleus Processing* in *Event Replicator for Adabas Concepts*.

KEY: Define Primary Key

The KEY parameter defines the primary key for replication. Following are rules for setting the KEY parameter:

- The parameter is optional when ON or MODIFY is specified.
- The parameter is not allowed when OFF is specified.
- The parameter may *not* be specified when OFF or NOKEY is specified.

Note: The field name specified must be a descriptor on the file. Note that "descriptor" in this case is used generically, as the field may be a descriptor, subdescriptor, super-descriptor, etc.

NOKEY: Remove Primary Key Setting

The NOKEY parameter removes the primary key setting when MODIFY is specified. Following are rules for setting the NOKEY parameter:

- The parameter is not allowed when OFF is specified.
- The parameter is the default setting when ON is specified and neither NOKEY nor KEY is specified.
- The parameter may *not* be specified when KEY is specified.

TARGET: Event Replicator Target ID

TARGET defines the Event Replicator target database ID. This parameter is required when ON is specified, is optional when MODIFY is specified, and is not allowed when OFF is specified.

Examples

```
ADADBS REPLICATION FILE=33,ON,TARGET=206,KEY=AA
```

or

```
ADADBS REPLICATION FILE=70,OFF
```

or

```
ADADBS REPLICATION FILE=71,MODIFY
ADADBS ACTIVATE
```

or

```
ADADBS REPLICATION FILE=72,MODIFY
ADADBS DEACTIVATE
```

54 ADADBS REPTOR: Activate, Deactivate, Open, or Close

Event Replicator Resources

- Essential Parameters 334
- Optional Parameters 334
- Examples 336

The ADADBS REPTOR function provides activation and deactivation control of Event Replicator resources. This function should be used with Event Replicator Server databases; it is not valid with other Adabas databases.

This chapter covers the following topics:

Essential Parameters

ACTIVATE: Event Replicator Server Resource Activation Request

The ACTIVATE parameter requests activation for the specified file, database ID, destination, or subscription.

One of the parameters, ACTIVATE, DEACTIVATE, OPEN, or CLOSE must be set. There is no default.

DEACTIVATE: Event Replicator Server Resource Deactivation Request

The DEACTIVATE parameter requests deactivation for the specified file, database ID, destination, or subscription.

One of the parameters, ACTIVATE, DEACTIVATE, OPEN, or CLOSE must be set. There is no default.

OPEN: Event Replicator Server Resource Open Request

The OPEN parameter requests that an unavailable destination or input queue be opened. When this parameter is specified, either the DESTINATION or IQUEUE parameter must be specified.

One of the parameters, ACTIVATE, DEACTIVATE, OPEN, or CLOSE must be set. There is no default.

CLOSE: Event Replicator Server Resource Close Request

The CLOSE parameter requests that an available destination or input queue be closed. When this parameter is specified, either the DESTINATION or IQUEUE parameter must be specified.

One of the parameters, ACTIVATE, DEACTIVATE, OPEN, or CLOSE must be set. There is no default.

Optional Parameters

DBID: Replicated DBID

The DBID parameter, when specified without the FILE parameter, will activate replication for any inactive files or deactivate replication for any active files for a given DBID. When specified with the FILE parameter, the DBID parameter identifies the database in which the file specified by the FILE parameter resides.

The DBID parameter is mutually exclusive with the DESTINATION, IQUEUE, and SUBSCRIPTION parameters.

DESTINATION: Replication Destination

The DESTINATION parameter can only be specified when the ACTIVATE, DEACTIVATE, OPEN, or CLOSE parameters are specified. The DESTINATION parameter supplies the name of the destination that should be activated, deactivated, opened, or closed. The destination specified must be defined to the Event Replicator Server.

If "ACTIVATE" is specified, the destination must already be inactive; if "DEACTIVATE" is specified, the destination must already be activated.

If "OPEN" is specified, the destination must be in an unavailable state; if "CLOSE" is specified, the destination must be in an available state.

The DESTINATION parameter is mutually exclusive with the DBID, FILE, IQUEUE, and SUBSCRIPTION parameters.

FILE: Replicated File

The FILE parameter will activate or deactivate replication for a single file in a specific database. A corresponding DBID parameter must also be specified.

If "ACTIVATE" is specified, the file must already be deactivated; if "DEACTIVATE" is specified, the file must already be activated.

The FILE parameter is mutually exclusive with the DESTINATION, IQUEUE, and SUBSCRIPTION parameters.

IQUEUE: Replication Input Queue

The IQUEUE parameter can only be specified when the OPEN or CLOSE parameters are specified. It supplies the name of the input queue (IQUEUE) that should be opened or closed and must be defined to the Event Replicator Server.

If "OPEN" is specified, the input queue must be in an unavailable state; if "CLOSE" is specified, the input queue must be in an available state.

The IQUEUE parameter is mutually exclusive with the DBID, DESTINATION, FILE, and SUBSCRIPTION parameters.

SUBSCRIPTION: Replication Subscription

The subscription specified for the SUBSCRIPTION parameter must be defined to the Event Replicator Server. It specifies the name of the subscription definition to use.

If "ACTIVATE" is specified, the subscription must already be deactivated. If "DEACTIVATE" is specified, the subscription must already be activated.

The SUBSCRIPTION parameter is mutually exclusive with the DBID, FILE, IQUEUE, and DESTINATION parameters.

Examples

The following example requests that the Event Replicator Server activate all inactive files for database 232.

```
ADADBS REPTOR ACTIVATE,DBID=232
```

The following example requests that the Event Replicator Server activate file 2 on database 232. The file is currently inactive.

```
ADADBS REPTOR ACTIVATE,DBID=232,FILE=2
```

The following example requests that the Event Replicator Server activate the destination defined by the DEST0001 destination. The destination is currently inactive.

```
ADADBS REPTOR ACTIVATE,DESTINATION=DEST0001
```

The following example requests that the Event Replicator Server activate the subscription defined by the SUBS0001 subscription. The subscription is currently inactive.

```
ADADBS REPTOR ACTIVATE,SUBSCRIPTION=SUBS0001
```

The following example requests that the Event Replicator Server deactivate all active files for database 232.

```
ADADBS REPTOR DEACTIVATE,DBID=232
```

The following example requests that the Event Replicator Server deactivate file 2 on database 232. The file is currently active.

```
ADADBS REPTOR DEACTIVATE,DBID=232,FILE=2
```

The following example requests that the Event Replicator Server deactivate the destination defined by the DEST0001 destination. The destination is currently active.

```
ADADBS REPTOR DEACTIVATE,DESTINATION=DEST0001
```

The following example requests that the Event Replicator Server deactivate the subscription defined by the SUBS0001 subscription. The subscription is currently active.

```
ADADBS REPTOR DEACTIVATE,SUBSCRIPTION=SUBS0001
```

The following example requests that Event Replicator Server open (restart) destination DEST0001. Each output task will be asked to process the destination if that output task does not already have the destination open for processing.

```
ADADBS REPTOR OPEN,DESTINATION=DEST0001
```

The following example requests that Event Replicator Server close destination DEST0001. Each output task will be asked to close the destination if that output task does not already have the destination closed for processing.

```
ADADBS REPTOR CLOSE,DESTINATION=DEST0001
```

The following example requests that Event Replicator Server open (restart) input queue IQUEUE01. Each input task will be asked to open the input queue if that input task does not already have the input queue open for processing.

```
ADADBS REPTOR OPEN,IQUEUE=IQUEUE01
```

The following example requests that Event Replicator Server close input queue IQUEUE01. Each input task will be asked to close the input queue if that input task does not already have the input queue closed for processing.

```
ADADBS REPTOR CLOSE,IQUEUE=IQUEUE01
```

55 RESETDIB: Reset Entries in Active Utility List

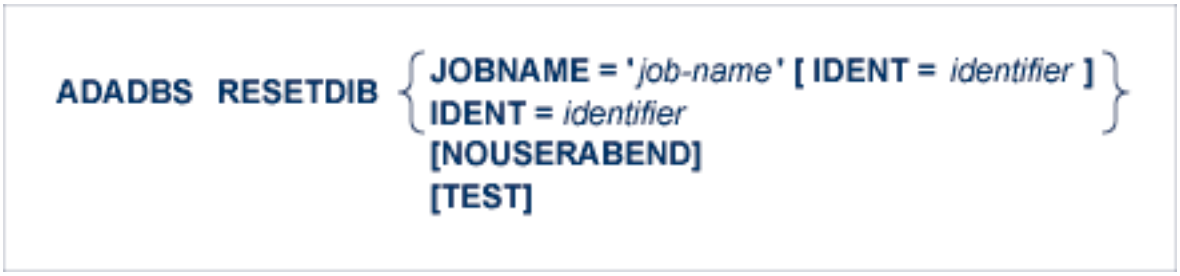
- Essential Parameters 399
- Optional Parameters 340
- Examples 341

The RESETDIB function resets entries in the active utility list (that is, the data integrity block or DIB).

Adabas maintains a list of the files used by each Adabas utility in the DIB. The DDIB operator command (or Adabas Online System) may be used to display this block to determine which jobs are using which files. A utility removes its entry from the DIB when it terminates normally. If a utility terminates abnormally (for example, the job is cancelled by the operator), the files used by that utility remain *in use*. The DBA may release any such files with the RESETDIB function.

Note: The RESETDIB function can be executed either with or without an active nucleus.

To remove a DIB from an abended ADAORD REORDB, REORDATA, REORASSO, ADADBS RESETDIB has to run without an active nucleus.



Essential Parameters

JOBNAME: Job Name

This parameter specifies the name of the job whose entry is to be reset. If it is not unique, the IDENT parameter must also be specified.

IDENT: Utility Execution Identifier

A unique number that identifies a utility execution. It may be specified alone or to qualify a job name when the same name has been used for various utility executions. The identifier may be obtained using the operator command DDIB or Adabas Online System.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Examples

The entry in the DIB block for job "JOB1" is to be deleted.

```
ADADBS RESETDIB JOBNAME='JOB1'
```

The entry in the DIB block for "JOB2" with IDENT=127 is to be deleted.

```
ADADBS RESETDIB
JOBNAME='JOB2',IDENT=127
```

56 RESETPPT: Reset PPT Blocks

- Syntax 344
- Essential Parameters 344
- Optional Parameters 345

The RESETPPT function resets the PPT blocks on the Associator data set. The PPT blocks contain information about the PLOG, CLOG, and work data sets that the nucleus is using. This information is used to perform validity checks on the work, PLOG, and CLOG data sets and is also used to determine if an autorestart is necessary.

Resetting the PPT is not without some risks:

- An attempt to reset the PPT when an autorestart is pending will prevent the nucleus from being able to recover from the current failure.
- An attempt to reset the PPT when PLOGs and CLOGs remain to be copied will disable all validity checks on the PLOG and CLOG data sets until new PPT information is written. This will, therefore, allow a data set to be overwritten, losing data.

Because of these risks, Software AG recommends that you use the RESETPPT function with extreme caution and only with direction from your Software AG customer support representative.

Caution: The nucleus must be down before you run this function. Otherwise, errors will result.

Syntax

```
ADADBS  RESETPPT  [NOUSERABEND]
                  [TEST]
```

Essential Parameters

There are no required parameters.

Optional Parameters

NOUSERABEND: Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See ***Syntax Checking with the TEST Parameter*** (elsewhere in this section) for more information on using the TEST parameter in ADADBS functions.

57 SPANCOUNT: Count Spanned Records

- Syntax 348
- Essential Parameters 348
- Optional Parameters 348
- Example 349

The SPANCOUNT function counts and displays the number of records in a file that are spanned.

Syntax

```
ADADBS  SPANCOUNT   FILE = file-number
                    [PASSWORD = 'password' ]
                    [NOUSERABEND]
                    [TIMELIMIT = { 60 | timelimit } ]
                    [TEST]
```

Essential Parameters

FILE: File Number

FILE specifies the number of the file whose spanned records are to be counted. Specify the valid file number of the file in the database for which you want spanned records counted.

Optional Parameters

NOUSERABEND: Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

The password of the file for which spanned records are to be counted. This parameter is required if the file is password-protected. Specify the password between apostrophes (').

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See *Syntax Checking with the TEST Parameter* (elsewhere in this section) for more information on using the TEST parameter in ADADBS functions.

TIMELIMIT: Number of Seconds

The maximum number of seconds the ADADBS SPANCOUNT function may run. If the function takes more time than specified by the TIMELIMIT parameter, the function will be aborted and appropriate error messages are issued.

The default is 60 seconds.

Example

In the following example, the spanned records in file 17 are counted.

```
ADADBS SPANCOUNT FILE=17
```

The results might look like this:

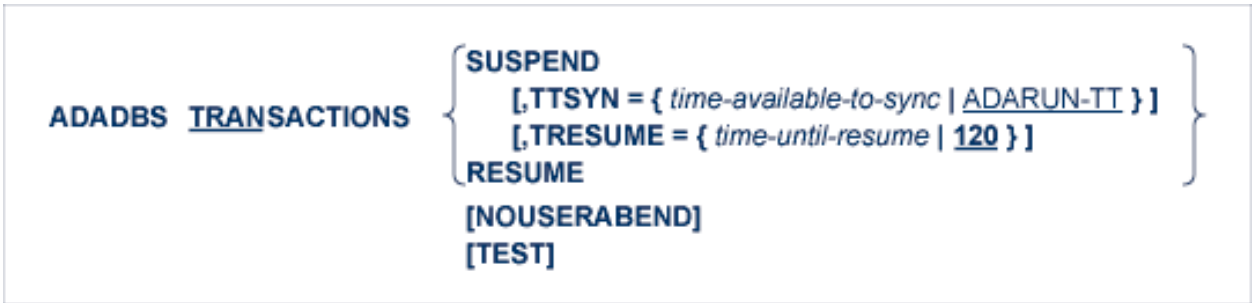
Records not spanned	15132
Primary records	19345
Secondary records	43897

58 TRANSACTIONS: Suspend and Resume Update

Transaction Processing

- Essential Parameters 353
- Optional Parameters 353
- Example 354

The TRANSACTIONS function may be used to suspend and resume update transaction processing; that is, to reach a quiesced state that could be a recoverable starting point.



Once the SUSPEND function has been submitted, new update transactions are held in the user queue. Executing transactions are allowed to finish if they can do so within the time allotted by the TTSYN parameter. Any transactions that exceed this time are backed out. In a cluster environment, all cluster nuclei are likewise quiesced.

Note: If you are trying to use the SUSPEND or RESUME functions of ADADBS TRANSACTIONS and if Natural Security is being used to log on, the FSEC Natural profile parameter for the FSEC Natural Security system file should be set with the RO (read-only) setting. If RO is not specified for the FSEC parameter, you will not be able to log onto Natural while Adabas is suspended because some of the logon programs in Natural Security require updates to the FSEC.

Once the quiesce is successful, the buffers are flushed for all nuclei so that the DASD files are current with the content of the buffers. A checkpoint SYNC-73 is written and ADADBS is notified.

At this point, you may run a non-Software AG fast backup product such as IBM's FlashCopy or StorageTek's SnapShot to copy off the database; that is, copy pointers to the data created by the fast backup product in the electronic memory of the array storage device.

Caution: Software AG does not recommend using such a database fast backup as a substitute for a regular Software AG database (or delta) save. Not only does Software AG have no control over the data sets that are included in the database fast backup, but it also cannot vouch for the success of the fast backup. Moreover, delta saves cannot sensibly be run on a copy of the database, as the DSF status change effected by the delta save would occur on the database copy instead of the original.

If the COPY completes before the TRESUME timeout and the RESUME function is issued, the nucleus writes a SYNS-74 checkpoint, leaves the suspended state and resumes update processing. The database was in a valid state over the whole duration of the COPY process.

If the COPY does not complete before the TRESUME timeout, Adabas automatically leaves the suspended state and resumes update processing. If the RESUME function is issued subsequently, Adabas rejects it with a response code and ADADBS terminates abnormally with an error message. This means that whatever COPY has been produced while update processing was suspended is

invalid and must not be used, because Adabas may have resumed updating the database while the COPY process was still in progress.

If the so-created copy of the database is used for recovery, removing the need to restore the database as of the time of the COPY, the subsequent regenerate should be started at the SYNC-73 checkpoint written at the end of the SUSPEND function.

Important: In a job where a SUSPEND function is followed by other job steps and then by a RESUME function, none of the job steps in between should be update-type commands or functions; otherwise, job execution will stall until the nucleus times out the suspended state.

Essential Parameters

SUSPEND: Suspend Transactions and Quiesce the Database

Use this parameter to suspend update transaction processing and quiesce the database.

RESUME: Resume Transaction Processing that was Previously Suspended

Use this parameter to resume update transaction processing that was previously suspended. If this parameter is used while Adabas is not in a suspended state or is no longer in a suspended state, this function terminates with an error.

Optional Parameters

TRESUME

Use this parameter to specify the amount of time in seconds the system is to remain quiesced after being suspended before the nucleus automatically resumes normal update transaction processing. If this parameter is not specified, the default is 120 seconds and the maximum is 86400 seconds or about 24 hours. The count begins when the nucleus has been successfully quiesced.

TTSYN

Use this parameter to specify the maximum amount of time the nucleus is to wait for all ET users to reach ET status before it forcibly ends and backs out update transactions that are still running in order to quiesce the system. If this parameter is not specified, the default is the ADARUN TT value.

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

Quiesce a database allowing 300 seconds for the currently running update transactions to finish and 150 seconds thereafter for the suspension to last before Adabas automatically resumes normal processing:

```
ADADBS TRANSACTIONS SUSPEND,TTSYN=300,TRESUME=150
```

59 UNCOUPLE: Uncouple Files

- Essential Parameter 356
- Optional Parameters 356
- Example 357

The UNCOUPLE function is used to eliminate the coupling relationship between two files.

```
ADADBS  UNCOUPLE   FILES = number, number
                   [NOUSERABEND]
                   [PASSWORD = 'password ' ]
                   [TEST]
```

Essential Parameter

FILES: Files to Be Uncoupled

FILES specifies the two files to be uncoupled.

Optional Parameters

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

PASSWORD specifies the security password for one or both files, and is required if either of the files is password-protected. If both files are password-protected, the password applies to both files. The password must be enclosed in single quotation marks.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See **Syntax Checking with the TEST Parameter** for more information on using the TEST parameter in ADADBS functions.

Example

Files 62 and 201 are to be uncoupled. One or both are protected with the password "PAIR05".

```
ADADBS UNCOUPLE
FILES=62,201,PASSWORD='PAIR05'
```

60 JCL/JCS Requirements and Examples

- Collation with User Exit 360
- BS2000 360
- z/OS 361
- z/VM 362
- VSE 363

This section describes the job control information required to run ADADBS with BS2000, z/OS, z/VM, and VSE systems, and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADADBS ONLINVERT execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

```
ADARUN CDXnn= exit-name
```

where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference* documentation for more information.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn		Required for OPERCOM DDIB or RESETDIB with inactive nucleus
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADADBS parameters	SYSDTA/ DDKARTE		*Utilities*
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADADBS messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADADBS JCL Example (BS2000)

In SDF Format:

```
/.ADADBS LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A D B S ALL FUNCTIONS
/REMARK *
/ASS-SYSLST L.DBS.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADBS,DB=yyyyy,IDTNAME=ADABAS5B
ADADBS REFRESH FILE=1
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADBS LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A D B S ALL FUNCTIONS
 /REMARK *
 /SYSFILE SYSLST=L.DBS
 /FILE ADA.MOD,LINK=DDLIB
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADADBS,DB=yyyyy,IDTNAME=ADABAS5B
 ADADBS REFRESH FILE=1
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	Required only for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADADBS messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations*
ADADBS parameters	DDKARTE	reader	

ADADBS JCL Example (z/OS)

Refer to ADADBS in the JOBS data set for this example.

```
//ADADBS    JOB
//*
//*     ADADBS:
//*        DATA BASE SERVICES (BATCH)
//*
//DBS       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD  <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADBS,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADBS      REFRESH FILE=1
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk/ terminal/ reader	Required only for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADADBS parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADADBS messages	DDDRUCK	disk/ terminal/ printer	

ADADBS JCL Example (z/VM)

Refer to ADADBS in the JOBS data set for this example.

```
DATADEF DDPRINT,DSN=ADADBS,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADADBS.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDBS.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADBS.CONTROL,MODE=A
ADARUN
```

Contents of RUNDBS CONTROL A1:

```
ADARUN PROG=ADADBS,DEVICE=dddd,DB=yyyyy
```

Contents of ADADBS CONTROL A1:

```
ADADBS REFRESH FILE=1
```

VSE

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	Required for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADBS parameters	-	reader	SYSIPT	*Utilities*
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADADBS messages	-	printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADADBS JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures.

Refer to member ADADBS.X in the JOBS data set for this example.

```
* $$ JOB JNM=ADADBS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       DATABASE SERVICES (BATCH)
// JOB ADADBS
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADBS,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADBS REFRESH FILE=1
/*
/&
* $$ EOJ
```

VI ADADCK Utility: Check Data Storage and DSST

This chapter covers the following topics:

Functional Overview
DSCHECK: Check Data Storage
JCL/JCS Requirements and Examples

61 Functional Overview

ADADCK checks Data Storage and the Data Storage space table (DSST) of a specific file (or files) in the database.

ADADCK reads each used Data Storage block (according to the Data Storage extents in the file control block) and performs the following checks:

- Is the block length within the permitted range? (4 block length physical block size)
- Is the sum of the length of all records in the Data Storage block plus 4 equal the block length?
- Is there any record with a record length greater than the maximum compressed record length for the file or with a length 0?
- Are there any duplicate ISNs within one block?
- If spanned records are used, are the ISNs in the header valid? Does each header contain the ISN of the primary record in the chain, the ISN of the previous spanned record in the chain, and the ISN of the next spanned record in the chain?
- If spanned records are used, are the primary and secondary spanned record identifier bits correct? Is only one of them turned on?
- Does the associated DSST element contain the correct value? If not, a REPAIR of the DSST is necessary (see **REPAIR parameter**).

Notes:

1. ADADCK does not require the Adabas nucleus to be active.
2. If the nucleus is active, ADADCK synchronizes its operation with the active nucleus unless the NOOPEN parameter is specified.
3. Any pending autorestart condition is ignored.
4. This utility should be used only for diagnostic purposes.

ADADCK returns a condition code 4 or 8 if an error occurs.

ADADCK and Spanned Records

If spanned records are used, the entire spanned record chain is checked for accuracy if any Data Storage RABN specified is part of the chain. If the FROMRABN and TORABN are specified and the file contains spanned Data Storage records, we recommend that the FROMRABN parameter point to the RABN of a primary record. Otherwise, a warning message may result. The ADADCK run will still result in a condition code of zero; the warning just indicates that it is not possible to check the entire chain because the FROMRABN that was specified was not for a primary record, but for a secondary record.

It is also possible that the secondary ISNs reside in a data RABN other than what was specified in the FROMRABN and TORABN parameter specifications. Therefore, additional Data Storage RABNs may need to be read to validate the secondary spanned record chain. ADADCK builds a secondary ISN table on the fly in an effort to optimize performance. If any secondary ISNs are noted missing, the entire data storage will be searched to attempt to locate the missing secondary ISNs.

The maximum number of ISNs that will be checked for a spanned Data Storage file is determined by the setting of the MAXPISN parameter, which defaults to 1000. If the file contains more than primary ISNs than the MAXPISN parameter setting, execution will continue but the following warning message will be displayed.

```
*** Warning ***
           More than MAXPISN primary spanned ISNs.  Only
           the first MAXPISN ISNs will be checked.  Run                    ↩

           ADADCK again specifying a different                             ↩

           FROMRABN to check the remaining RABNs or
           specify a higher MAXPISN value.
           Any errors reported after this warning
           may be due to the table limitation.
```

If this warning message appears, Any errors or CC=8 conditions that are reported after this warning may be due to the fact that the utility was unable to track the ISNs due to the size limitation of 1000. The file may be intact and the error may be because the internal table used for ADADCK processing could not hold all of the ISNs. To be sure that the file is fine, run the ADAVAL utility on it.

62 DSCHECK: Check Data Storage

- Optional Parameters and Subparameters 370
- Examples 371

```
ADADCK DSCHECK [FILE = { file [FROMRABN = DS-blknum ] [TORABN = DS-blknum ] | file - file } ]
               [MAXPISN = { num | 1000 } ]
               [NOOPEN]
               [NOUSERABEND]
               [REPAIR]
               [USAGE]
```

This chapter describes the syntax and parameters of the DSCHECK function.

Optional Parameters and Subparameters

FILE: Files to Be Checked

The file (or a single range of files) to be checked. If omitted, all files in the database are checked.

FROMRABN: Data Storage Block Number

The RABN of the Data Storage block where the check is to start. This parameter is applicable only if a single file is to be checked. In other words, only one FROMRABN/TORABN range can be specified in a single ADADCK run.

If more than one FROMRABN/TORABN range is specified in an ADADCK request for multiple files, only the last range is used in the run. In addition, if the range specified by the FROMRABN/TORABN parameters is outside the range for any given file DS extent, ADADCK will not check the blocks in the extent. Consequently, if more than one FROMRABN/TORABN range is specified, or if the range is outside the range for any given file DS extent, Adabas issues a warning message.

If this parameter is omitted, the check starts at the beginning of the first allocated Data Storage extent for the file.

MAXPISN

The maximum number of primary ISNs that will be checked for a spanned Data Storage file. The default is 1000. If the file contains more primary ISNs than the MAXPISN setting, execution will continue but the following warning message will be displayed.

```
*** Warning ***
           More than MAXPISN primary spanned ISNs.  Only
           the first MAXPISN ISNs will be checked.  Run                    ↩

           ADADCK again specifying a different                             ↩

           FROMRABN to check the remaining RABNs or
           specify a higher MAXPISN value.
           Any errors reported after this warning
           may be due to the table limitation.
```

NOOPEN: Prevent Open Synchronization

When starting, ADADCK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADADCK from issuing the open call and blocking file usage for other users.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

REPAIR: Repair the Data Storage Space Table

If ADADCK finds any invalid Data Storage Space Table (DSST) elements, it automatically repairs the table if this parameter is supplied.

TORABN: Ending Data Storage Block Number

The RABN of the Data Storage block where the check is to end. This parameter is applicable only if a single file is to be checked. In other words, only one FROMRABN/TORABN range can be specified in a single ADADCK run.

If more than one FROMRABN/TORABN range is specified in an ADADCK request for multiple files, only the last range is used in the run. In addition, if the range specified by the FROMRABN/TORABN parameters is outside the range for any given file DS extent, ADADCK will not check the blocks in the extent. Consequently, if more than one FROMRABN/TORABN range is specified, or if the range is outside the range for any given file DS extent, Adabas issues a warning message.

If this parameter is omitted, the check ends at the end of the last allocated Data Storage extent for the file.

USAGE: Print Data Storage Block Usage

If USAGE is specified, ADADCK prints a bar graph that shows the number of bytes used in each Data Storage block, the block size, and the percentage of blocks used.

Examples

Check Data Storage and its DSST for file 20, print a bar graph of the Data Storage block utilization and repair the space table if required.

```
ADADCK DSCHECK FILE=20, USAGE, REPAIR
```

Check Data Storage and its DSST for the files 8 through 12.

```
ADADCK DSCHECK FILE=8-12
```

Check Data Storage and its DSST for file 12 in the RABN range 878 through 912.

```
ADADCK DSCHECK FILE=12,
FROMRABN=878,TORABN=912
```

63 JCL/JCS Requirements and Examples

- BS2000 374
- z/OS 375
- z/VM 376
- VSE 377

This section describes the job control information required to run ADADCK with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSOR*n* or DDASSO*nn*	disk	
Data Storage	DDDATAR*n* or DDDATA*nn*	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADADCK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADADCK messages	SYSLST DDDRUCK		*Messages and Codes*

ADADCK JCL Example (BS2000)

In SDF Format:

```
/.ADADCK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A D C K DATA STORAGE CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.DCK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADCK,DB=yyyyy,IDTNAME=ADABAS5B
ADADCK DSCHECK FILE=27
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADCK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A D C K DATA STORAGE CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.DCK.DATA
/FILE ADA.MOD,LINK=DDLIB

/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADCK,DB=yyyyy,IDTNAME=ADABAS5B
ADADCK DSCHECK FILE=27
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSOR*n* or DDASSO*nn*	disk	
Data Storage	DDDATAR*n* or DDDATA*nn*	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADADCK parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADADCK messages	DDDRUCK	printer	*Messages and Codes*

ADADCK JCL Example (z/OS)

Refer to ADADCK in the JOBS data set for this example.

```
//ADADCK    JOB
//*
//*    ADADCK:
//*          DATA STORAGE CHECK
//*
//DCK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
```

```
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADCK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADCK DSCHECK FILE=27
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSOR*n* or DDASSO*nn*	disk	
Data Storage	DDDATAR*n* or DDDATA*nn*	disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADADCK parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADADCK messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADADCK JCL Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADADCK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADADCK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDCK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADCK.CONTROL,MODE=A
ADARUN
```

Contents of RUNDCK CONTROL A1:

```
ADARUN PROG=ADADCK,DEVICE=dddd,DB=yyyyy
```

Contents of ADADCK CONTROL A1:

```
ADADCK DSCHECK FILE=27
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSOR*n* or ASSO*nn*	disk	*	
Data Storage	DATAR*n* or DATA*nn*	disk	*	
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADCK parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADADCK messages		printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADADCK JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to member ADADCK.X for this example.

```
* $$ JOB JNM=ADADCK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADCK
*       DATA STORAGE CHECK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADCK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADCK DSCHECK FILE=27
/*
/&
* $$ EOJ
```

VII ADADEF Utility: Define a Database

This chapter covers the following topics:

Functional Overview
DEFINE: Defining a Database and Checkpoint File
MODIFY: Change Field Encodings
NEWWORK: Defining a Work File
JCL/JCS Requirements and Examples

64 Functional Overview

- Database Components 382
- Checkpoint File 382

The following database characteristics are defined with ADADEF:

- database name and ID
- database components (Associator, Data Storage, and Work)
  - device type
  - size
- checkpoint system file
- database default encodings

Database Components

Each database component (Associator, Data Storage, and Work) must be formatted by the ADAFRM utility before it is defined with ADADEF. The ADADEF utility may also be used to define a new Work data set for an existing database.

Systems using the Recovery Aid feature require a recovery log (RLOG) data set, which must first be formatted with the ADAFRM utility, and then defined using the ADARAI utility.

Checkpoint File

Adabas uses the checkpoint system file to store checkpoint data and user data provided with the Adabas CL and ET commands. It is required and must be specified using the ADADEF DEFINE (database) function.

65 DEFINE: Defining a Database and Checkpoint File

- Essential Parameters 386
- Optional Parameters 387
- Examples 391

The database and the checkpoint file must be defined at the same time.

The database parameters include the required ASSOSIZE, DATASIZE, and WORKSIZE parameters and the optional (non-indented) parameters ASSODEV through WORKDEV shown in the syntax diagram.

The FILE=...,CHECKPOINT,... statement is also required for database definition. The checkpoint file parameters (indented under the FILE statement in the syntax diagram) should be specified immediately following the FILE statement. See the **examples**.

```
ADADEF  DEFINE   ASSOSIZE = size-list
                 DATASIZE = size-list
                 WORKSIZE = size
                 FILE = file-number, CHECKPOINT
                       DSSIZE =  size
                       MAXISN =  maximum-number-of-records-expected
                       [ACRABN = starting-rabn ]
                       [ASSOPFAC = { Associator-padding-factor |  10  } ]
                       [ASSOVOLUME = ' Associator-extent-volume ' ]
                       [DATAPFAC = { Data-Storage-padding-factor |  10  } ]
                       [DATAVOLUME = ' Data-Storage-extent-volume' ]
                       [DSDEV = device-type ]
                       [DSRABN = starting-rabn ]
                       [DSREUSE = { NO |  YES  } ]
                       [ISNSIZE = {  3 | 4 } ]
                       [MAXDS = maximum-Data-Storage-secondary-allocation ]
                       [MAXNI = maximum-normal-index-secondary-allocation ]
                       [MAXUI =  maximum-upper-index-secondary-allocation ]
                       [NAME = { ' file-name' |  CHECKPOINT  } ]
                       [NIRABN = starting-rabn ]
                       [NISIZE = size ]
                       [UIRABN = starting-rabn ]
                       [UISIZE =  size ]
                 [ASSODEV = { device-type-list  |  ADARUN-device  } ]
                 [DATADEV = { device-type-list |  ADARUN-device  } ]
                 [DBIDENT = { database-id |  ADARUN-dbid  } ]
                 [DBNAME = {  database-name |  GENERAL-DATABASE } ]
                 [FACODE = {  alpha-EBCDIC-key  |  37  } ]
                 [FWCODE = {  wide-key  |  4095  } ]
                 [MAXFILES = { maximum-number-of-files |  255  } ]
                 [NOUSERABEND]
                 [OVERWRITE]
                 [RABNSIZE = {  3  | 4 } ]
                 [REPTOR = YES | NO ]
                 [UACODE = { alpha-ASCII-key |  437  } ]
                 [UES = {  YES | NO } ]
                 [UWCODE = {  wide-key | FWCODE-definition } ]
                 [WORKDEV = {  device-type-list | ADARUN-device } ]
```

Essential Parameters

ASSOSIZE/ DATASIZE/ WORKSIZE: Database Size

ASSO-/DATA-/WORKSIZE specifies the number of blocks or cylinders to be assigned to the Associator, Data Storage, or Work. A block value must be followed by a "B"; otherwise, the value is assumed to be cylinders.

If the Associator or Data Storage is to be contained on more than one data set, the size of each data set must be specified. If a companion ASSODEV or DATADEV parameter specifies two or more extents, the equivalent ASSOSIZE or DATASIZE parameter must specify the extent sizes as positional operands in the corresponding order (see the **examples**).

The minimum WORKSIZE allowed is 300 blocks.

Note: If ASSOSIZE or DATASIZE is not specified, the ADADEF DEFINE function will not execute. If WORKSIZE is not specified, the function will allocate three (3) cylinders to the Work data set. Because 3 cylinders are usually not enough to start the database, WORKSIZE is considered to be a required parameter.

DSSIZE: Data Storage Size

DSSIZE specifies the number of blocks or cylinders to be assigned to checkpoint/Data Storage. For blocks, the value specified must be followed by a "B" (for example, DSSIZE=80B).

The size of the checkpoint file specified with the DSSIZE and MAXDS parameters depends on

- the amount of ET data to be stored;
- the number of utility runs for which checkpoint information is to be retained;
- the number of user IDs.

FILE . . . CHECKPOINT Parameter

The FILE...CHECKPOINT parameter indicates the file number to be used for the checkpoint system file. This parameter is required; the file number must be 5000 or lower.

Adabas uses the checkpoint system file to store checkpoint data and user data provided with the Adabas CL and ET commands.

MAXISN: Highest ISN to be Used

The highest ISN that may be assigned to the file. The value specified is used to determine the space allocation for the address converter. When determining the MAXISN, consider the importance of ET data and checkpoint data to your site.

Adabas considers ET data to be more important than checkpoint data. As soon as the ET data ISN range in the checkpoint system file is exhausted, the first checkpoint ISN is deleted and given to the ET data. This is an ongoing process. As soon as the MAXISN is reached, a new address converter extent is allocated and given to the checkpoint data. You can delete checkpoint data piece by piece using the Adabas Online System function DELCP.

Note: The way the checkpoint handles data is subject to change in a future release of Adabas.

Optional Parameters

ACRABN/ DSRABN/ NIRABN/ UIRABN: Starting RABN

These parameters may be used to cause allocation for their respective areas to begin with the specified RABN:

- ACRABN for the address converter
- DSRABN for Data Storage
- NIRABN for the normal index
- UIRABN for the upper index

ASSODEV/ DATADEV/ WORKDEV: Device Type

ASSO-/DATA-/WORKDEV specify the device type(s) to be assigned to the Associator, Data Storage, and Work. These parameters are required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

WORKDEV, if specified, can only be one device type. If the Associator (ASSODEV) or Data Storage (DATADEV) is to be contained on more than one data set, the device type for each data set must be specified, even if both extents are on the ADARUN DEVICE type.

If multiple extents are used with VSAM data sets, ASSODEV and DATADEV must reflect the dynamic device type; that is, DD/xxxxR1=9999; DD xxxxR2=8888; ... DD/xxxxR5=5555. For example, when defining DDDATAR1 and DDDATAR2, DATADEV=9999,8888.

Space allocation for specified device types must be given in companion ASSOSIZE, DATASIZE, and WORKSIZE parameters on this or another ADADEF statement in the same job. If a ASSODEV or DATADEV parameter specifies more than one extent on the same or different device types (DATADEV=3380,3350, for example), the companion ASSOSIZE or DATASIZE parameter must specify the related extent sizes in corresponding order.

ASSOPFAC/ DATAPFAC: Padding Factor

ASSOPFAC defines the percentage of space in each Associator RABN block to be reserved for later entries (padding space). This space is used for later descriptor extensions or ISN additions. The percentage value specified, which can range 1-90, should be large enough to avoid the overhead caused when block overflow forces splitting of an existing address block into two new blocks. If ASSOPFAC is not specified, ADADEF assumes a padding factor of 10%.

DATAPFAC defines the percentage of space in each Data Storage RABN block to reserve for later entries (padding space). This space is used when changes to an existing data record cause it to need more space in the block; an updated record that no longer fits in the existing block must be moved to another block. The percentage value specified, which can range 1-90, should

be large enough to avoid the overhead caused when block overflow forces splitting of an existing address block into two new blocks. If DATAPFAC is not specified, ADADEF assumes a padding factor of 10%.

ASSOVOLUME/ DATAVOLUME: Extent Volume

Note: Values for ASSOVOLUME and DATAVOLUME must be enclosed in apostrophes.

ASSOVOLUME specifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated.

If the requested number of blocks cannot be found on the specified volume, ADADEF retries the allocation while disregarding the ASSOVOLUME or DATAVOLUME parameter value.

If ACRABN, UIRABN, or NIRABN is specified, ADADEF ignores the ASSOVOLUME value when allocating the corresponding extent type.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If ASSOVOLUME and/or DATAVOLUME are not specified, the file's Associator and/or Data Storage space, respectively, is allocated according to ADADEF's default allocation rules.

DBIDENT: Database Identifier

DBIDENT specifies the identification number to be assigned to the database. A value in the range 1-65535 may be specified. If this parameter is omitted, the value specified with the ADARUN DBID parameter is used.

If multiple databases are to be established, the DBIDENT parameter is required in order to uniquely identify each database.

DBNAME: Database Name

DBNAME is the name to be assigned to the database. This name appears in the title of the Database Status Report produced by the ADAREP utility. A maximum of 16 characters may be specified. Enclose the name in single quotation marks if the name includes any special characters other than dashes, or if the name contains embedded blanks.

If this parameter is omitted, a default value of "GENERAL-DATABASE" is assigned.

DSDEV: Device Type for Data Storage

DSDEV specifies the device type to be used for the checkpoint file's Data Storage. There is no default value; if DSDEV is not specified, an arbitrary device type is used.

DSREUSE: Storage Reusage

DSREUSE indicates whether space which becomes available in the checkpoint file is to be reused. The default is YES.

FACODE: Encoding for Alphanumeric Fields

The FACODE parameter specifies the default encoding for alphanumeric fields for all files in the database. The encoding must be derived from EBCDIC encoding; that is, X'40' is the space character. Modal or shift-type double-byte character set (DBCS) encodings are supported; fixed type DBCS (DBCS-only) encodings are not supported. The default encoding key is 37.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can change the default encoding set in this parameter using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

FWCODE: Encoding for Wide-Character Fields

The FWCODE parameter specifies the default encoding for wide-character (W) format fields for all files in the database. The default encoding is 4095; that is, Unicode.

The FWCODE parameter can be used to set a wide-character encoding that defines the superset of code points of all user encodings. For example, Unicode encompasses about 50,000 code points as opposed to Host-DBCS and Shift-JIS with about 10,000 code points each.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can change the default encoding set in this parameter using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE indicates whether ISNs in the file are 3 or 4 bytes long. The default is 3 bytes.

MAXDS/ MAXNI/ MAXUI: Maximum Secondary Allocation

MAXDS/NI/UI specify the maximum number of blocks per secondary extent for Data Storage, the normal index, and the upper index, respectively. The value specified must be followed by a "B" for blocks (for example, MAXDS=8000B) and cannot be more than 65535B.

MAXFILES: Highest File Number

MAXFILES specifies the maximum number of files that can be loaded into the database. The minimum value for this parameter is 3. The highest value permitted is 5000 or one less than the ASSOR1 block size, whichever is lower. For example, 2003 is the highest MAXFILES value for a database whose ASSOR1 is stored on a 3380 DASD.

The value specified determines the number of file control blocks and field definition tables to be allocated when the database is being established. Each file control block requires one Associator block and each field definition table requires four Associator blocks.

If this parameter is omitted, a value of 255 is assigned.

Once the database has been established, the value for MAXFILES may be changed only by executing the REORASSO or REORDB functions of the ADAORD utility.

NAME: Name of the Checkpoint File

NAME specifies the name for the checkpoint file being defined. This name appears on the Database Status Report produced by the ADAREP utility. The maximum number of characters permitted is 16. The default file name is CHECKPOINT.

NISIZE: Normal Index Size

NISIZE specifies the number of blocks or cylinders to be assigned to the normal index. For blocks, the value specified must be followed by "B" (for example, NISIZE=80B).

NOUSERABEND: Termination Without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing Database

Specify OVERWRITE to write over an existing database. OVERWRITE cannot be specified when creating a database with newly formatted data sets.

RABNSIZE: 3- or 4-Byte RABN

RABNSIZE specifies the length of RABNs in the database. Specify 3 for 24-bit RABNs or 4 for 31-bit RABNs. The default is 3.

REPTOR: Set Event Replicator Database

The `REPTOR` parameter is used with ADADEF DEFINE to specify whether a normal Adabas database is being defined, or whether a Event Replicator database is being defined. Possible values are "YES" or "NO" (the default).

Specify `REPTOR=NO` for a normal Adabas database. Specify `REPTOR=YES` for an Event Replicator database.

UACODE: User Encoding for Alphanumeric Fields

The parameter UACODE specifies the default encoding for alphanumeric fields for ASCII users. The encoding must be derived from ASCII encoding; that is, X'20' is the space character. Encodings for multiple-byte character sets are supported. The default encoding is 437.

The UACODE value is not stored in the file being loaded.

You can override the default encoding set in this parameter for a user session using the OP command. You can change it generally using the ADADEF MODIFY function.

UES: Universal Encoding Support

Setting the parameter UES activates universal encoding support for the database. Any valid xxCODE parameter (FACODE, FWCODE, UACODE, UWCODE) implicitly sets UES=YES.

To deactivate UES, you must explicitly set UES=NO.

You can change the default setting of this parameter generally using the ADADEF MODIFY function.

UISIZE: Upper Index Size

UISIZE specifies the number of blocks or cylinders to be assigned to the upper index. For blocks, the value specified must be followed by "B" (for example, UISIZE=80B).

UWCODE: User Encoding for Wide-Character Fields

The UWCODE parameter specifies the user encoding for wide-character (W) format fields. If the parameter is not specified, the default value is the current value of FWCODE.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can override the default encoding set in this parameter for a user session using the OP command. You can change the default setting generally using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

Examples

Example 1:

```
ADADEF DEFINE
ADADEF  ASSOSIZE=200,DATASIZE=600,WORKSIZE=50
ADADEF  DBIDENT=1,DBNAME=DATABASE-1
ADADEF  MAXFILES=150
ADADEF  FILE=1,CHECKPOINT
ADADEF  NAME='DB1-CHECKPOINT',MAXISN=5000
ADADEF  DSSIZE=2,NISIZE=50B,UISIZE=10B
```

The Associator, Data Storage and Work sizes are equal to 200, 600 and 50 cylinders, respectively. The numeric identifier for the database is 1 and the database name is DATABASE-1. The maximum number of files (and the highest file number) that may be loaded into the database is 150. File 1 is to be reserved for the Adabas checkpoint file. The name of the first system file is to be DB1-CHECKPOINT. The Data Storage size for this file is to be 2 cylinders; the normal index size 50 blocks; the upper index size 10 blocks; and the MAXISN is to be 5000.

Example 2:

```
ADADEF DEFINE
ADADEF  ASSODEV=3380,DATADEV=3380,3390,WORKDEV=3380
ADADEF  ASSOSIZE=100,DATASIZE=200,300,WORKSIZE=25
ADADEF  DBIDENT=2,DBNAME='DATABASE_2'
ADADEF  MAXFILES=255
ADADEF  FILE=255,CHECKPOINT,MAXISN=5000
ADADEF  DSSIZE=3,NISIZE=100B,UISIZE=20B
```

The Associator is to be contained on a 3380 device type, and occupies 100 cylinders. Data Storage comprises two data sets: the first data set is 200 cylinders contained on the first DATADEV (3380)

device type, and the second data set is 300 cylinders contained on the second DATADEV (3390) device type. The Work space is 25 cylinders on the WORKDEV device (3380).

The numeric identifier for the database is 2, and the database name is DATABASE_2. A maximum of 255 files may be loaded into the database. An Adabas checkpoint file is loaded during this step.

66 MODIFY: Change Field Encodings

- Optional Parameters 394
- Examples 396

The MODIFY function is used to modify encodings set for the database using ADADEF DEFINE. At least one of the optional encoding parameters must be specified.

Changing the FACODE, FWCODE, or UWCODE parameters does not affect files already loaded since the actual encoding of their fields is stored in the FCB. The purpose of the database-wide setting is to serve as a default when loading files.

```
ADADEF  MODIFY  [FACODE = { alpha-EBCDIC-key | current-setting} ]
                [FWCODE = { wide-key  | current-setting} ]
                [NOUSERABEND]
                [REPTOR = YES | NO ]
                [UACODE = { alpha-ASCII-key | current-setting } ]
                [UES = { YES | NO } ]
                [UWCODE = { wide-key |
```

Optional Parameters

FACODE: Encoding for Alphanumeric Fields

The FACODE parameter specifies the default encoding for alphanumeric fields for all files in the database. The encoding must be derived from EBCDIC encoding; that is, X'40' is the space character. Modal or shift-type double-byte character set (DBCS) encodings are supported; fixed type DBCS (DBCS-only) type encodings are not supported. The default encoding key is the current setting.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

FWCODE: Encoding for Wide-Character Fields

The FWCODE parameter specifies the default encoding for wide-character (W) format fields for all files in the database. The default encoding is the current setting.

The FWCODE parameter can be used to set a wide-character encoding that defines the superset of code points of all user encodings. For example, Unicode encompasses about 50,000 code points as opposed to Host-DBCS and Shift-JIS with about 10,000 code points each.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

REPTOR: Set Event Replicator Database

The REPTOR parameter is used with ADADEF MODIFY to indicate whether the database that is running is an Adabas database or an Event Replicator Server database. Possible values are "YES" or "NO" (default).

For a normal Adabas database, the REPTOR parameter must always be set to "NO". When a Event Replicator Server is running, the REPTOR parameter must be set to "YES".

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

UACODE: User Encoding for Alphanumeric Fields

The parameter UACODE specifies the default encoding for alpha fields for ASCII users. The encoding must be derived from ASCII encoding; that is, X'20' is the space character. Encodings for multiple-byte character sets is supported. The default encoding is the current setting.

The UACODE setting is not stored in the loaded file. You can override this encoding for a user session with the OP command.

UES: Universal Encoding Support

The parameter UES can be used to enable or disable universal encoding support for an existing database. Disabling is only possible if no files are loaded with wide-character (W) format fields.

Any valid xxCODE parameter (FACODE, FWCODE, UACODE, UWCODE) implicitly sets UES=YES.

To deactivate UES, you must explicitly set UES=NO.

UWCODE: User Encoding for Wide-Character Fields

The UWCODE parameter specifies the user encoding for wide-character (W) format fields. If the parameter is not specified, the default value is the current setting.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

You can override the default encoding for a user session with the OP command.

Examples

Example 1:

Disable universal encoding support for an existing database. The database contains no files with wide (W) format.

```
ADADEF MODIFY UES=NO
```

Example 2:

Change the default encoding for wide-character (W) format fields for all files in the database from the current setting to code page 835 (traditional Chinese host double byte including 6204 user-defined characters).

```
ADADEF MODIFY FWCODE=835
```

Files already loaded are not affected by this change since the actual encoding of their fields is stored in the FCB. The purpose of the database-wide setting is to serve as a default when loading files.

67 NEWWORK: Defining a Work File

- Essential Parameter 398
- Optional Parameters 398
- Example 399

The following parameters are used for Work data set definition:

```
ADADEF  NEWWORK   WORKSIZE = size
                  [NOUSERABEND]
                  [WORKDEV = {device-type-list | ADARUN-device } ]
```

Notes:

1. The Adabas nucleus must not be active during this function, and the old Work must be specified in the JCL/JCS.
2. The ADADEF NEWWORK function cannot be executed if a pending autorestart exists.

Essential Parameter

WORKSIZE: Work Data Set Size

The number of blocks or cylinders to be assigned to the Work data set.

Optional Parameters

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

WORKDEV: Device Type

The device type to be assigned to the new Work data set.

This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

Example

A new Work data set is defined with a size of 50 cylinders. The device type is obtained from the ADARUN DEVICE parameter.

```
ADADEF NEWWORK
ADADEF WORKSIZE=50
```

68 JCL/JCS Requirements and Examples

- BS2000 402
- z/OS 403
- z/VM 405
- VSE 407

This section describes the job control information required to run ADADEF with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADADEF parameters	SYSDTA/ DDKARTE		*Utilities*
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADADEF messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADADEF JCL Examples (BS2000)

Define Database

In SDF Format:

```
/.ADADEF LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE DATABASE
/REMARK *
/ASS-SYSLST L.DEF.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF MAXFILES=120
ADADEF FILE=1,CHECKPOINT
ADADEF NAME= CHECKPOINT ,MAXISN=5000,UISIZE=10B
ADADEF DSSIZE=500B,NISIZE=100B
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADEF LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE DATABASE
/REMARK *
/SYSFILE SYSLST=L.DEF.DEFI
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF MAXFILES=120
ADADEF FILE=1,CHECKPOINT
ADADEF NAME= CHECKPOINT ,MAXISN=5000,UISIZE=10B
ADADEF DSSIZE=500B,NISIZE=100B
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work (Current)	DDWORKR1 DDWORKR4	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADADEF parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADADEF messages	DDDRUCK	printer	*Messages and Codes*

ADADEF JCL Examples (z/OS)

Define Database

```
//ADADEF    JOB
//*
//*   ADADEF:
//*      DEFINE THE PHYSICAL LAYOUT OF THE DATABASE
//*      DEFINE THE NUCLEUS SYSTEM FILE: CHECKPOINT FILE
//*
//DEF       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADEF,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADEF DEFINE DBNAME=EXAMPLE-DB,DBIDENT=YYYYY
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILES=120
*

ADADEF FILE=19,CHECKPOINT
ADADEF   NAME='CHECKPOINT',MAXISN=5000
ADADEF   DSSIZE=100B,NISIZE=3B,UISIZE=3B
/*
```

Refer to ADADEF in the JOBS data set for this example.

Define New Work

```
//ADADEFNW  JOB
//*
//*    ADADEF: DEFINE NEW WORK
//*
//DEF       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
```

```
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADEF,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
/*
```

Refer to ADADEFNW in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADADEF parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADADEF messages	DDDRUCK	disk/ terminal/ printer	

ADADEF JCL Examples (z/VM)

Define Database

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.ASSO,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADADEF.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADADEF.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDEF.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADEF.CONTROL,MODE=A
ADARUN
```

Contents of RUNDEF CONTROL A1:

```
ADARUN PROG=ADADEF,DEVICE=dddd,DB=yyyyy
```

Contents of ADADEF CONTROL A1:

```
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILE=120
*

ADADEF FILE=1,CHECKPOINT
ADADEF NAME='CHECKPOINT',MAXISN=5000,UISIZE=10B
ADADEF                   DSSIZE=500B,NISIZE=100B
```

Define New Work

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.ASSO,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADADEF.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADADEF.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDEF.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADEF.CONTROL,MODE=A
ADARUN
```

Contents of RUNDEF CONTROL A1:

```
ADARUN PROG=ADADEF,DEVICE=dddd,DB=yyyyy
```

Contents of ADADEF CONTROL A1:

```
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
Work (Current)	WORKR1	disk	*	
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADEF parameters	-	reader	SYSIPT	
ADARUN messages	-	printer	SYSLST	
ADADEF messages	-	printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADADEF JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures.

Define Database

Refer to member ADADEF.X for this example.

```
* $$ JOB JNM=ADADEF,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADEF
*       DEFINE THE PHYSICAL LAYOUT OF THE DATABASE
*       DEFINE THE NUCLEUS SYSTEMFILE: CHECKPOINT FILE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADEF,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADEF DEFINE DBNAME=EXAMPLE-DB,DBIDENT=yyyyy
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILES=120
*
```

```
ADADEF FILE=19,CHECKPOINT
ADADEF   NAME='CHECKPOINT',MAXISN=5000
ADADEF   DSSIZE=100B,NISIZE=3B,UISIZE=3B
/*
/&
* $$ EOJ
```

Define New Work

Refer to member ADADEFNW.X for this example.

```
* $$ JOB JNM=ADADEFNW,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADEFNW
*       DEFINE NEW WORK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADEF,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
/*
/&
* $$ EOJ
```

VIII ADAFRM Utility: Format Adabas Database Components

This chapter covers the following topics:

Functional Overview
Formatting Database Components
JCL/JCS Requirements and Examples

69 Functional Overview

- Statement Restrictions 412
- Formatting Operation 412

Primary Adabas direct access (DASD) data sets must be formatted using the ADAFRM utility.

These data sets include the Associator, Data Storage, and Work data sets as well as the intermediate storage (Temp, Sort, and Command/Protection/Recovery logging) data sets.

Formatting must be performed before any new data set can be used by the Adabas nucleus or an Adabas utility. After increasing a data set with the ADADBS INCREASE or ADD function, new RABNs must also be formatted.

ADAFRM also provides functions to reset existing Associator, Data Storage, or Work blocks/cylinders to binary zeros (nulls). Resetting fills the specified blocks in an existing Associator, Data Storage, or Work data set with binary zeros.

Finally, if you are using Adabas 8 or later, you can use this utility to clear multiple PLOG headers from the PLOG, without requiring that you reformat the entire PLOG. For more information, read about using the FROMRABN, NUMBER, and SIZE parameters together, as described in the ADAFRM ***FROMRABN***, ***NUMBER***, and ***SIZE*** parameter documentation, elsewhere in this chapter.

Statement Restrictions

More than one ADAFRM function (ASSOFRM, DATAFRM, RLOGFRM, and so on) can be performed in the same job. However, each function must be specified on separate statements. See the examples at the end of the do for more information.

Formatting Operation

Formatting with ADAFRM comprises two basic operations:

1. creating blocks (called RABNS) on the specified tracks/cylinders;
2. filling the created blocks with binary zeros (nulls).

70 Formatting Database Components

- Formatting Modes 414
- Syntax 414
- Essential Parameter 416
- Optional Parameters 416
- Examples 417

This chapter describes the syntax, parameters, and processing of the ADAFRM utility.

Formatting Modes

There are several ADAFRM formatting modes:

1. Format a *new* data set (...FRM functions). Only the data set specified by the function name and the NUMBER parameter is accessed and formatted. The FROMRABN parameter cannot be specified when formatting a new data set.
2. Format *part of an existing* data set (ASSOFRM, DATAFRM, WORKFRM, and TEMPFRM functions). Here, the FROMRABN parameter *must* be specified, except on z/OS platforms. When formatting Work and Data Storage (WORKFRM and DATAFRM functions), the ADAFRM job control must also contain the Associator data sets.

   This formatting mode is used in combination with the ADADBS INCREASE function for ASSO and DATA. If a greater WORK is needed, then ADADEF NEWWORK should be used.
3. Reformat *blocks of an existing* data set (...RESET functions). This mode opens all Associator, Data Storage, and Work data sets in the database for access. The FROMRABN parameter is *must* be specified for these functions.
4. If you are using Adabas 8 or later, you can use this utility to clear multiple PLOG headers from the PLOG, without requiring that you reformat the entire PLOG. For more information, read about using the FROMRABN, NUMBER, and SIZE parameters together, as described in the ADAFRM ***FROMRABN***, ***NUMBER***, and ***SIZE*** parameter documentation, elsewhere in this section.

Syntax

To format the Associator (ASSO..) or Data Storage (DATA..) data set, use this syntax:

```
ADAFRM  { ASSOFRM | DATAFRM }   SIZE = size
                                [DEVICE = { device-type |  ADARUN-device  } ]
                                [ { FROMRABN = {starting-rabn | NEXT } |
                                    NUMBER = {dataset-number |  1 } } ]
                                [NOUSERABEND]
```

To format the Work (WORK..) data set, Command log (CLOG..), Protection log (PLOG..), or Sort (SORT..) data set, use this syntax:

Note: If you are using Adabas 8 or later, you can also use this syntax to clear multiple PLOG headers from the PLOG, without requiring that you reformat the entire PLOG. For more information, read about using the FROMRABN, NUMBER, and SIZE parameters together, as described in the ADAFRM *FROMRABN*, *NUMBER*, and *SIZE* parameter documentation, elsewhere in this section.

```
ADAFRM  { WORKFRM | CLOGFRM | PLOGFRM | SORTFRM }
                    SIZE = size
                    [DEVICE = { device-type | ADARUN-device } ]
                    [ { FROMRABN = starting-rabn  |  NUMBER = {dataset-number | 1 } } ]
                    [NOUSERABEND]
```

To format the Recovery log (RLOG..) data set, use this syntax:

```
ADAFRM  RLOGFRM  SIZE = size
                 [DEVICE = { device-type |  ADARUN-device } ]
                 [NOUSERABEND]
```

To format a Temp (TEMP..) data set, use this syntax:

```
ADAFRM  TEMPFRM  SIZE = size
                 [DEVICE = { device-type |  ADARUN-device } ]
                 [FROMRABN =  starting-rabn ]
                 [NOUSERABEND]
```

To reformat blocks of an existing Associator, Data Storage, or Work data set, use this syntax:

```
ADAFRM  { ASSORESET | DATARESET | WORKRESET }
                    SIZE = size
                    FROMRABN = start-rabn
                    [NOUSERABEND]
```

Essential Parameter

SIZE: Size of Area to be Formatted

SIZE specifies the size of the area to be formatted (or reset). Blocks (a decimal value followed by a "B") or cylinders may be specified. For the RLOGFRM function, the size must be the same as that specified by the RLOGSIZE parameter on the ADARAI utility's PREPARE function. See section **Essential Parameter**.

If you are using Adabas 8 or later and you want to clear multiple PLOG headers from the PLOG without reformatting the entire PLOG, set the value of the **SIZE** parameter to "1" and specify values for both the **FROMRABN** and **NUMBER** parameters.

Optional Parameters

DEVICE: Device Type

DEVICE is the physical device type on which the area to be formatted is contained. If DEVICE is not specified, the device type specified by the ADARUN DEVICE parameter is used.

FROMRABN: Starting RABN

FROMRABN specifies the RABN at which formatting or resetting is to begin. This parameter may only be used for an existing data set; NUMBER cannot be specified in the same ADAFRM job as FROMRABN.

When FROMRABN is specified with a *xxxx*FRM function, formatting begins at the FROMRABN point and continues up to the *highest complete track* before the RABN computed from FROMRABN + SIZE (assuming a size specified in or converted to blocks). This means that the last track within the specified range (FROMRABN + SIZE) will be formatted *only* if all the track's RABNs are within that range.

If you are using Adabas 8 or later and you want to clear multiple PLOG headers from the PLOG without reformatting the entire PLOG, set the value of the **SIZE** parameter to "1" and specify values for both the **FROMRABN** and **NUMBER** parameters.

When increasing the size of an ASSO or DATA data set, FROMRABN is available as an option only under VSE, z/VM, and BS2000. The specified RABN must be one higher than the highest allocated RABN before the logical increase using ADADBS (which must precede the physical increase using ADAFRM). FROMRABN=NEXT instructs ADAFRM to take the first unformatted RABN as the value for FROMRABN. ADAFRM then verifies that the range of blocks determined for formatting by the NEXT value is contained in the free space table (FST). If not, ADAFRM terminates with ERROR-126. On z/OS, FROMRABN should only be used to reformat existing blocks as the last record pointer in the VTOC cannot be modified by function FROMRABN. See the **examples for ADADBS INCREASE** .

This parameter is *required* for the ASSORESET, DATARESET and WORKRESET functions. When specified with the function ASSORESET, the FROMRABN value must be greater than 30.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMBER: Data Set Number

NUMBER selects the nonsequential command log, nonsequential protection log, Associator, Data Storage and sort data set to be formatted. The default is 1 (first data set). Values allowed for:

- the Associator (ASSO) or Data Storage (DATA) are 1 through 99;
- protection logs (PLOGs) or command logs (CLOGs) are 1 through 8;
- the recovery log (RLOG) is just 1;
- SORT is either 1 or 2 (1 only on VSE systems); and
- WORK or TEMP is either 1 or the default.

ADAFRM ...FRM function statements cannot specify (and will not default to) a NUMBER value if other ADAFRM statements in the same job specify a FROMRABN value.

NUMBER must match the number suffix of the related data definition (DD) statement. See the tables of allowed statements and the examples in section **JCL/JCS Requirements and Examples**.

If you are using Adabas 8 or later and you want to clear multiple PLOG headers from the PLOG without reformatting the entire PLOG, set the value of the **SIZE** parameter to "1" and specify values for both the **FROMRABN** and **NUMBER** parameters.

Examples

Example 1:

Format 50 cylinders for the Associator, 200 cylinders for Data Storage, 10 cylinders for Work, and 2 cylinders for the recovery log (RLOG).

```
ADAFRM   ASSOFRM SIZE=50,DEVICE=3380
ADAFRM  DATAFRM SIZE=200,DEVICE=3380
ADAFRM  WORKFRM SIZE=10,DEVICE=3380
ADAFRM  RLOGFRM SIZE=2
```

Example 2:

One cylinder for nonsequential command log data set 1, and 1 cylinder for nonsequential command log data set 2 are to be formatted.

```
ADAFRM   CLOGFRM SIZE=1,DEVICE=3350,NUMBER=1
ADAFRM  CLOGFRM SIZE=1,DEVICE=3350,NUMBER=2
```

Example 3:

The first two blocks of an existing Work data set are to be reset to binary zeros.

```
ADAFRM WORKRESET FROMRABN=1,SIZE=2B
```

Example 4:

Assuming the Data Storage data set is on a 3380 disk (9 blocks/track, 15 tracks/cylinder), 100 cylinders-starting at cylinder position 201 relative to the beginning of the data set-will be formatted.

```
ADAFRM DATAFRM SIZE=100,FROMRABN=26992
```

Example 5:

Under VSE, z/VM, or BS2000, assuming the Associator of the database has just been increased by 200 cylinders, this job formats the new space in the database. For more detailed examples across all supported platforms, see the ADADBS INCREASE examples in section **Operating-System-Specific Procedures**.

```
ADAFRM ASSOFRM SIZE=200,FROMRABN=NEXT
```

71 JCL/JCS Requirements and Examples

- BS2000 420
- z/OS 422
- z/VM 423
- VSE 425

This section describes the job control information required to run ADAFRM with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Note: When running with the optional Recovery Aid (RLOG), all temporary data sets must also be cataloged in the job control.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	data sets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1 DDWORKR4		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAFRM parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAFRM messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADAFRM JCL Example (BS2000)

In SDF Format:

```
/.ADAFRM LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A F R M ALL FUNCTIONS
/REMARK *

/ASS-SYSLST L.FRM
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,OPEN-MODE=OUTIN
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,OPEN-MODE=OUTIN
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,OPEN-MODE=OUTIN
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP,OPEN-MODE=OUTIN
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT,OPEN-MODE=OUTIN
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,OPEN-MODE=OUTIN
```

```
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,OPEN-MODE=OUTIN
/SET-FILE-LINK DDRLOGR1,ADAyyyyy.RLOGR1,OPEN-MODE=OUTIN
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY ADARUN ↩
PROG=ADAFRM,DB=yyyyy,IDTNAME=ADABAS5B
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40
ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40,NUMBER=1
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAFRM LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A F R M ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.FRM
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,OPEN=OUTIN
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,OPEN=OUTIN
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,OPEN=OUTIN
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1,OPEN=OUTIN
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1,OPEN=OUTIN
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,OPEN=OUTIN
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,OPEN=OUTIN
/FILE ADAyyyyy.RLOGR1,LINK=DDRLOGR1,OPEN=OUTIN
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAFRM,DB=yyyyy,IDTNAME=ADABAS5B
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40

ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40,NUMBER=1
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	data sets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1 DDWORKR4		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	DDCARD	reader	*Operations*
ADAFRM parameters	DDKARTE	disk	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAFRM messages	DDDRUCK	printer	*Messages and Codes*

ADAFRM JCL Example (z/OS)

Refer to ADAFRM in the JOBS data set for this example.

```
//ADAFRM    JOB
//*
//*     ALLOCATE AND FORMAT THE DATABASE COMPONENTS
//*
//*     MORE THAN ONE DATA SET CAN BE FORMATTED IN A SINGLE RUN
//*
//*

//FRM       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD                <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.ASSOR1, <=== ASSO
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL001
//DDDATAR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.DATAR1, <=== DATA
//            SPACE=(CYL,(0,200)),UNIT=DISK,VOL=SER=VOL002
//DDWORKR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.WORKR1, <=== WORK
//            SPACE=(CYL,(0,40)),UNIT=DISK,VOL=SER=VOL003
//DDSORTR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.SORTR1, <=== SORT
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL003
//DDTEMPR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.TEMPR1, <=== TEMP
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL003
```

```
//DDPLOGR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.PLOGR1, <=== PLOG1
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDPLOGR2  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.PLOGR2, <=== PLOG2
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDCLOGR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.CLOGR1, <=== CLOG1
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDCLOGR2  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.CLOGR2, <=== CLOG2
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAFRM,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAFRM ASSOFRM SIZE=100,DEVICE=dddd
ADAFRM DATAFRM SIZE=200,DEVICE=dddd
ADAFRM WORKFRM SIZE=40,DEVICE=dddd
ADAFRM SORTFRM SIZE=100,DEVICE=dddd
ADAFRM TEMPFRM SIZE=100,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	data sets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1 DDWORKR4		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAFRM parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAFRM messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAFRM JCL Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOG1,VOL=PLOGV1
DATADEF DDPLOGR2,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDRLOGR1,DSN=ADABASVv.RLOG1,VOL=RLOGV1
DATADEF DDPRINT,DSN=ADAFRM.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAFRM.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNFRM.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAFRM.CONTROL,MODE=A
ADARUN
```

Contents of RUNFRM CONTROL A1:

```
ADARUN PROG=ADAFRM,DEVICE=dddd,DB=yyyyy
```

Contents of ADAFRM CONTROL A1:

```
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40
ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	files to be formatted
Data Storage	DATARn			
Work	WORKR1			
Temp	TEMPR1			
Sort	SORTR1			
Multiple command log	CLOGRn			
Multiple protection log	PLOGRn			
Recovery log	RLOGR1			
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAFRM parameters	-	reader	SYSIPT	
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADAFRM messages	-	printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADAFRM JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to member ADAFRM.X for this example.

```
* $$ JOB JNM=ADAFRM,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAFRM
*       FORMAT THE DATABASE COMPONENTS
?/ EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAFRM,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAFRM ASSOFRM SIZE=100,DEVICE=dddd
ADAFRM DATAFRM SIZE=200,DEVICE=dddd
ADAFRM WORKFRM SIZE=40,DEVICE=dddd
ADAFRM SORTFRM SIZE=100,DEVICE=dddd
ADAFRM TEMPFRM SIZE=100,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
```

```
ADAFRM CLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
/*
/&
* $$ EOJ
```

IX ADAICK Utility: Check Index and Address Converter

This chapter covers the following topics:

Functional Overview
ACCHECK: Check Address Converter
ASSOPRINT: Print/Dump Associator Blocks
BATCH: Set Printout Width to 132 Characters Per Line
DATAPRINT: Print/Dump Data Storage Blocks
DSCHECK: Print/Dump Content of Data Storage Record
DUMP: Activate Dump Print Format
FCBPRINT: Print/Dump File Control Block
FDTPRINT: Print/Dump Field Definition Table
GCBPRINT: Print/Dump General Control Blocks
ICHECK: Check Index Against Address Converter
INT: Activate Interpreted Print Format
NIPRINT: Print/Dump Normal Index
NOBATCH: Set Print Width to 80 Characters Per Line
NODUMP: Suppress Dump Print Format
NOINT: Suppress Interpreted Format
PPTPRINT: Print/Dump Parallel Participant Table
UIPRINT: Print/Dump Upper Index
Examples
JCL/JCS Requirements and Examples

72 Functional Overview

ADAICK checks the physical structure of the Associator. This includes validating the index based upon the descriptor value structures and the Associator extents defined by the general control blocks (GCBs) and file control blocks (FCBs).

The ADAICK utility should be used only for diagnostic purposes.

ADAICK can perform the following functions:

- Check index and address converter for specific files;
- Print/dump the contents of any ASSO or DATA block in the database;
- Print/dump the contents of normal (NI) and upper (UI) indexes.
- Print/dump formatted the contents of GCBs, FCBs, FDTs, and PPTs.

When specifying an ISN for a spanned record in an ADAICK utility run, be sure to specify the primary ISN of the spanned record, not a secondary ISN of the spanned record. ADAICK utility processing assumes all specified ISNs are primary ISNs; secondary ISNs will automatically be processed.

Notes:

1. ADAICK can run with or without an active Adabas nucleus.
2. A pending autorestart condition is ignored.
3. If the nucleus is active, ADAICK synchronizes its operation with the active nucleus unless the NOOPEN parameter is specified.
4. If spanned records are used in the database, the report produced by ADAICK DSCHECK identifies the primary and secondary spanned record ISNs.

73 ACCHECK: Check Address Converter

- Essential Parameter 432
- Optional Parameters 432
- Sample Output 432

The ACCHECK function checks the address converter of a specified database file. In addition, it automatically checks the secondary address converter if the file contains spanned records.

```
ADAICK   ACCHECK   FILE = file-number
                   [NOPEN]
                   [NOUSERABEND]
```

Essential Parameter

FILE: File to be Checked

The file to be checked. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file used by ADAICK is checked.

Optional Parameters

NOOPEN: Prevent Open Synchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

Sample Output

The following sample of the output produced from an ADAICK ACCHECK run is for an Adabas 8 database that makes use of spanned records, and thus includes a secondary address converter.

```
FILE 002 AC CHECK
FILE 002                   AC                                LOW            HIGH   ↩
         NR OF                                THRU
FILE 002                   ISN-ISN                          RABN          DS RABN   ↩
  DS RABN        RECORDS ISN (DEC)
FILE 002                   00000001-0000034F 000000E1      000001D9 0000023C      ↩
      25       847
FILE 002                   00003850-00003B9F 000000F2      00000000 00000000      ↩
      25       15,263
FILE 002 AC2 CHECK
FILE 002                                            AC2         LOW AC2   HIGH AC2   ↩
         NR OF      THRU
FILE 002      AC2 ISN-ISN              RABN       DS RABN     DS RABN            ↩
RECORDS AC2 ISN
FILE 002      00000001-0000034F 000000F3 000001DA 0000023F           100          ↩
 848
```

74 ASSOPRINT: Print/Dump Associator Blocks

- Essential Parameter 523
- Optional Parameter 436

```
ADAICK   ASSOPRINT   RABN = { rabn | rabn-rabn }
                     [NOUSERABEND]
```

This chapter describes the syntax and parameters of the ASSOPRINT function.

Essential Parameter

RABN: RABNs to be Processed

The RABN (or a single range of RABNs) to be printed/dumped. If ADAICK can determine the type of information stored in the block (for example. UI, NI,...), it produces a formatted printout.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

75 BATCH: Set Printout Width to 132 Characters Per Line

- Optional Parameter 438

```
ADAICK  BATCH  [NOUSERABEND]
```

If ADAICK is to be used in batch mode, this function may be used to set the printout width to 132 characters per line. See the **NOBATCH function** for information about resetting the printout width.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

76 DATAPRINT: Print/Dump Data Storage Blocks

- Essential Parameter 440
- Optional Parameter 440

The ADAICK DATAPRINT function prints (and dumps) the Data Storage blocks for specified RABNs. In the output, when a spanned record RABN is requested, the record is identified as a primary ("Primary ISN") or secondary ("AC2 ISN") record. In addition, the master ISN (the primary ISN of the spanned record) and the next ISN number are listed. The next ISN number is always the next secondary record ISN (zero if it is the last record).

The phrase "ERRORSPAN*" may appear in the output if the spanned Data Storage record is not flagged correctly with the primary or secondary ISN bit set. This will be indicated with a condition code of "4".

```
ADAICK   DATAPRINT   RABN = {rabn | rabn-rabn}
                     [NOUSERABEND]
```

Essential Parameter

RABN: RABNs to be Processed

The RABN (or a single range of RABNs) to be printed/dumped.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

77 DSCHECK: Print/Dump Content of Data Storage Record

- Essential Parameter 524
- Optional Parameters 442
- Sample Output 443

Use the DSCHECK function to print and dump the content of the Data Storage records in a file.

```
ADAICK   DSCHECK   FILE = file-number
                   [ISN = isn-of-record ]
                   [NOOPEN]
                   [NOUSERABEND]
```

If you are running Adabas 8 or later, primary and secondary ISNs are identified in the output from an ADAICK DSCHECK run, if spanned Data Storage records are in use.

Essential Parameter

FILE: File Number
The number of the file for which the record is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

ISN: ISN of Data Storage Record
The ISN of the Data Storage record to be printed. If ISN is omitted, the DSCHECK function prints the last ISN plus 1.

If the Data Storage record is a spanned record, be sure to specify the primary ISN of the spanned record, not a secondary ISN of the spanned record. ADAICK utility processing assumes all specified ISNs are primary ISNs; secondary ISNs will automatically be processed.

NOOPEN: Prevent Open Resynchronization
When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND: Termination without Abend
When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

Sample Output

Here is a sample of the output produced from an ADAICK DSCHECK run for an Adabas 8 database that makes use of spanned records. Note that the primary and secondary spanned record ISNs are identified in the report.

```
000001D9 0004   0000  LEN      130D       AB IS THE LAST FIELD IN THE RECORD
000001D9
000001D9 000A   0006  ISN      00000001    Primary ISN=1
000001D9 000E   000A  FLAGS    0081
000001D9 0010   000C  Next ISN 00000001
000001D9 0014   0010  Primary  00000001
000001DA 0004   0000  LEN      1303        BA IS THE LAST FIELD IN THE RECORD
000001DA
000001DA 000A   0006  AC2 ISN  00000001    AC2 ISN=1
000001DA 000E   000A  FLAGS    0041
000001DA 0010   000C  Next ISN 00000002
 000001DA 0014   0010  Primary  00000001                                        ↩
```

78 DUMP: Activate Dump Print Format

- Optional Parameter 446

```
ADAICK   DUMP [NOUSERABEND]
```

This function requests that blocks be printed in dump format. See the **NODUMP function** for information about suppressing dump format printing. This function should only be used in conjunction with the **ASSOPRINT**, **DATAPRINT**, **FCBPRINT**, **FDTPRINT**, **GCBPRINT**, and **PPTPRINT** functions.

If both the INT and DUMP functions are specified, blocks are printed in both formats.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

79 FCBPRINT: Print/Dump File Control Block

- Essential Parameter 448
- Optional Parameters 448
- Output Considerations 448

```
ADAICK   FCBPRINT  FILE = file-number
                   [NOPEN]
                   [NOUSERABEND]
```

Use the FCBPRINT function to print and dump the file control block (FCB) of a file.

Essential Parameter

FILE: File Number

The number of the file for which the FCB is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOOPEN: Prevent Open Resynchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

Output Considerations

If the first unused RABN is equal to the last RABN plus 1, then it is very likely that the extent is full and there *may* be an additional extent. This is true of the first extent in the following example (highlighted in *blue*). In this case, the first unused RABN is 00002BFE, one more than the last RABN of the extent (00002BFD):

```
FI 00018 FCB +1A4           First NI RABN: 00002945
FI 00018 FCB +1A8            Last NI RABN: 00002BFD
FI 00018 FCB +1AC    First unused NI RABN: 00002BFE
FI 00018 FCB +1B0           First NI RABN: 00002EE3
FI 00018 FCB +1B4            Last NI RABN: 00002FCB
FI 00018 FCB +1B8    First unused NI RABN: 00002FBC
```

The first unused RABN does not necessarily lie in the next extent.

80 FDTPRINT: Print/ Dump Field Definition Table

- Essential Parameter 452
- Optional Parameters 452

```
ADAICK   FDTPRINT   FILE = file-number
                    [NOUSERABEND]
```

Use the FDTPRINT function to print and dump the field definition table (FDT) of a file.

Essential Parameter

FILE: File Number

The number of the file for which the FDT is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

81 GCBPRINT: Print/Dump General Control Blocks (GCBs)

- Optional Parameter 454

```
ADAICK   GCBPRINT  [NOUSERABEND]
```

Use the GCBPRINT function to print and dump the general control blocks (GCBs).

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

82 ICHECK: Check Index Against Address Converter

- Essential Parameter 456
- Optional Parameters 456

Use the ICHECK function to check the index against the address converter.

```
ADAICK   ICHECK   FILE = {file-number | file-number-file-number }
                  [NOOPEN]
                  [NOUSERABEND]
```

Essential Parameter

FILE: Files to be Checked

The specified file (or a single range of files) to be checked. FILE must be specified.

Optional Parameters

NOOPEN: Prevent Open Resynchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

83 INT: Activate Interpreted Print Format

- Optional Parameter 458

```
ADAICK INT [NOUSERABEND]
```

This function requests that blocks are printed in an interpreted format. See the **NOINT function** for information about suppressing interpreted print format. This function should only be used in conjunction with the **ASSOPRINT**, **DATAPRINT**, **FCBPRINT**, **FDTPRINT**, **GCBPRINT**, and **PPTPRINT** functions.

If both the INT and DUMP functions are specified, blocks are printed in both formats.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

84 NIPRINT: Print/Dump Normal Index

- Essential Parameter 460
- Optional Parameter 460

Use the NIPRINT function to print and dump the normal index.

```
ADAICK   NIPRINT   FILE = file-number
                   [NOUSERABEND]
```

Essential Parameter

FILE: File Number

The number of the file for which the normal index is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

85 NOBATCH: Set Print Width to 80 Characters Per Line

- Optional Parameter 462

```
ADAICK  NOBATCH  [NOUSERABEND]
```

The printout width is set to 80 characters per line. See the **BATCH function** for information about resetting the printout width.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

86 NODUMP: Suppress Dump Print Format

- Optional Parameter 464

This function suppresses the dump print format produced, by default, for ADAICK dumps. See the **DUMP function** for information about dump print format. This function should only be used in conjunction with the **ASSOPRINT**, **DATAPRINT**, **FCBPRINT**, **FDTPRINT**, **GCBPRINT**, and **PPTPRINT** functions.

```
ADAICK   NODUMP [NOUSERABEND]
```

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

87 NOINT: Suppress Interpreted Format

- Optional Parameter 466

```
ADAICK NOINT [NOUSERABEND]
```

This function suppresses the interpreted print format produced, by default, by ADAICK. See the **INT function** for information about interpreted print format. This function should only be used in conjunction with the **ASSOPRINT**, **DATAPRINT**, **FCBPRINT**, **FDTPRINT**, **GCBPRINT**, and **PPTPRINT** functions.

Optional Parameter

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

88 PPTPRINT: Print/Dump Parallel Participant Table

- Optional Parameters 468
- Example Output 468

```
ADAICK   PPTPRINT  [NOUSERABEND]
```

Use the PPTPRINT function to dump and print the parallel participant table (PPT) for the Adabas cluster. Note that in the dump/print, 'PPH' is the tag for the PPT header and 'PPE' is the tag for the PPT entries.

Each of the 32 blocks (RABNs) allocated for the PPT represents a single nucleus in the cluster and comprises

- a single header of fixed length; and
- multiple entries of variable length.

In the dump/print, 'PPH' is the tag for a PPT block's header and 'PPE' is the tag for a PPT block's entries.

Optional Parameters

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

Example Output

```
ADAICK PPTPRINT

   MEANING: DUMP ASSO BLOCK 000000BF THRU 000000DE
DB 00072 PPT AT RABN      000000BF
DB 00072 PPT BLOCK NUMBER 01
DB 00072 PPH+000                 NUMBER OF ENTRIES: 03
DB 00072 PPH+001                 NUCLEUS INDICATOR: C0
DB 00072 PPH+002                    EXTERNAL NUCID: 0000
DB 00072 PPH+004                            UNUSED: 00000000
DB 00072 PPE+000               LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
```

```
DB 00072 PPE+00A                    PTT STATUS FLAG: 00
DB 00072 PPE+00B                    ID OF PPT ENTRY: W
DB 00072 DATASET=ADABAS.GB.UTI.72.WORKR1
DB 00072 PPE+000                LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
DB 00072 PPE+00A                    PTT STATUS FLAG: 00
DB 00072 PPE+00B                    ID OF PPT ENTRY: 1
DB 00072 DATASET=ADABAS.GB.UTI.72.PLOGR1
DB 00072 PPE+000                LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
DB 00072 PPE+00A                    PTT STATUS FLAG: 00
DB 00072 PPE+00B                    ID OF PPT ENTRY: 2
DB 00072 DATASET=ADABAS.GB.UTI.72.PLOGR2

ASSO BLOCK 000000BF  PPT
0000  03C00000 00000000  00230000 00000000  *.�       .      *
0010  000000E6 7AC1C4C1  7A5BC7C5 C24BE4E3  *   WADABAS.GB.UT*
0020  C94BF7F2 4BE6D6D9  D2D9F100 23000000  *I.74.WORKR1 .   *
0030  00000000 0000F17A  C1C4C17A 5BC7C5C2  *      1ADABAS.GB*
0040  4BE4E3C9 4BF7F24B  D7D3D6C7 D9F10023  *.UTI.74.PLOGR1 .*
0050  00000000 00000000  00F27AC1 C4C17A5B  *         2ADABAS*
0060  C7C5C24B E4E3C94B  F7F24BD7 D3D6C7D9  *.GB.UTI.74.PLOGR*
0070  F2000000 00000000  00000000 00000000  *2               *
0080  00000000 00000000  00000000 00000000  *                *
      SAME
0FF0  00000000 00000000  00000000           *            *

DB 00072 PPT RABNS 000000C0 - 000000DE (02-32) ARE UNUSED

A D A I C K  TERMINATED NORMALLY                          2000-07-26 09:45:19
```

89 UIPRINT: Print/Dump Upper Index

- Essential Parameter 472
- Optional Parameters 536

Use the UIPRINT function to print and dump the upper index.

```
ADAICK   UIPRINT   FILE = file-number
                   [NOUSERABEND]
```

Essential Parameter

FILE: File Number

The number of the file for which the upper index(es) is/are to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

90 Examples

Example 1:

Check the index and address converter for file 18 and print/dump the FDT for this file.

```
ADAICK ICHECK FILE=18
ADAICK FDTPRINT
```

Example 2:

Set printout width to 120 characters per line (printer). Check index and address converter for file 1 and print/dump Associator RABNs 123 through 135.

```
ADAICK BATCH
ADAICK ICHECK  FILE=1
ADAICK ASSOPRINT RABN=123-135
```

Example 3:

The following produces an interpreted format (INT) printout of the Associator blocks (ASSOPRINT). The NODUMP function indicates that a dump of the blocks should not also be produced.

```
ADAICK INT NODUMP ASSOPRINT RABN=1
```

```
DB 00204 GCB 00000001
DB 00204 GCB+000                    DATA BASE ID: 00CC
DB 00204 GCB+002              MAXIMUM NR OF FILES: 07D3
DB 00204 GCB+004                  FILE 1 FCB RABN: 0000001F
DB 00204 GCB+008                  FILE 1 FDT RABN: 000007F2
DB 00204 GCB+00C              CURRENT SIBA NUMBER: 0003
.......
```

Example 4:

The following produces a dump (DUMP) of the Associator blocks (ASSOPRINT). The NOINT function indicates that an interpreted format printout of the blocks should not also be produced.

```
ADAICK NOINT DUMP ASSOPRINT RABN=1
```

```
ASSO BLOCK 00000001
0000  00CC07D3 0000001F  000007F2 00034000  * ..L   .  .2 .  *
0010  D4C1E3E3 C8C9C1E2  60C1C4C1 C2C1E240  *MATTHIAS-ADABAS *
```

91 JCL/JCS Requirements and Examples

- Collation with User Exit 476
- BS2000 476
- z/OS 478
- z/VM 479
- VSE 480

This section describes the job control information required to run ADAICK with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADAICK execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding subsystem (UES), the format of the collation descriptor user exit parameter is

```
ADARUN CDXnn= exit-name
```

where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference* documentation for more information.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAICK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAICK messages	SYSLST DDDRUCK		*Messages and Codes*

ADAICK JCL Example (BS2000)

In SDF Format:

```
/.ADAICK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A I C K INDEX CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.ICK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAICK,DB=yyyyy,IDTNAME=ADABAS5B
ADAICK ICHECK FILE=27
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAICK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A I C K INDEX CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.ICK.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAICK,DB=yyyyy,IDTNAME=ADABAS5B
ADAICK ICHECK FILE=27
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADAICK parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAICK messages	DDDRUCK	printer	*Messages and Codes*

ADAICK JCL Example (z/OS)

Refer to ADAICK in the JOBS data set for this example.

```
//ADAICK    JOB
//*
//*   ADAICK:
//*      INDEX AND ADDRESS CONVERTER CHECK
//*
//ICK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAICK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAICK ICHECK FILE=1-3
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAICK parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAICK messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAICK JCL Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAICK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAICK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNICK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAICK.CONTROL,MODE=A
ADARUN
```

Contents of RUNICK CONTROL A1:

```
ADARUN PROG=ADAICK,DEVICE=dddd,DB=yyyyy
```

Contents of ADAICK CONTROL A1:

```
ADAICK ICHECK FILE=27
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAICK parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	Messages and Codes
ADAICK messages		printer	SYS009	Messages and Codes

* *Any programmer logical unit may be used.*

ADAICK JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to member ADAICK.X for this example.

```
* $$ JOB JNM=ADAICK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAICK
*       INDEX AND ADDRESS CONVERTER CHECK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAICK,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAICK ICHECK FILE=1-3
/*
/&
* $$ EOJ
```

X ADAINV Utility: Inverted List Management

This chapter covers the following topics:

Functional Overview
COUPLE: Define File-Coupling Descriptors
INVERT: Create Descriptors
JCL/JCS Requirements and Examples

92 Functional Overview

The INVERT function:

- modifies the field definition table (FDT) to indicate that the specified field is a descriptor; and
- adds all values and corresponding ISN lists for the field to the inverted list.

The newly defined descriptor may then be used in the same manner as any other descriptor. This function may also be used to create a subdescriptor, superdescriptor, phonetic descriptor, or hyperdescriptor.

The COUPLE function adds a common descriptor to two files (updates their inverted lists). Any two files may be coupled provided that a common descriptor with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

Note: Only files with numbers 255 or lower can be coupled.

Changes affecting a coupled file's inverted lists are automatically made to the other file. The DBA should consider the additional overhead required to update the coupling lists when the descriptor used as the basis for coupling is updated, or when records are added to or deleted from either file. If a field that is not defined with the NU option is used as the basis for coupling and the field contains a large number of null values, a considerable amount of additional execution time and required disk space to store the coupling lists may result.

An interrupted ADAINV operation can be restarted without first having to restore the file.

The ADAINV utility requires that the nucleus be active. If the nucleus is canceled (terminated) while the ADAINV utility is running, all the work performed at that point is lost. This is because the FDT and the FCB have not been rewritten to include the new descriptor. The rewrite occurs at the very end of processing and is triggered by a special Adabas call sent by ADAINV.

93 COUPLE: Define File-Coupling Descriptors

- Essential Parameters 486
- Optional Parameters 487
- Example 488
- Temporary Space for File Coupling 488
- Associator Coupling Lists 489
- Space for Coupling Lists 490
- Space Allocation 491

Use the COUPLE function to define one descriptor for each of two files to be coupled.

```
ADAINV   COUPLE    FILES = file-number1 , file-number2
                   DESCRIPTOR = 'fieldname , fieldname'
                   SORTSIZE = size
                   TEMPSIZE = size
                   [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                   [LWP = { workpool-size | 1048576 } ]
                   [NOUSERABEND]
                   [PASSWORD = 'password' ]
                   [SORTDEV = {device-type | ADARUN-device } ]
                   [TEMPDEV = {device-type | ADARUN-device } ]
                   [TEST]
```

Essential Parameters

DESCRIPTOR: Descriptors Used as Basis for Coupling

The DESCRIPTOR parameter defines one descriptor in each file to provide the basis for coupling the files. Subdescriptors or superdescriptors may also be used, or may be defined as or derived from a multiple-value field. The descriptors specified may not be contained within a periodic group, nor be derived from a periodic group. The descriptors can have different names, but must have the same length and format definitions.

FILES: Files to Be Coupled

FILES specifies the two files to be coupled. The number of each file must be 255 or lower. The files specified may not be currently coupled to each other.

SORTSIZE: Sort Size

SORTSIZE specifies the space available for the sort data set or data sets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by a "B"). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference* documentation for more information on estimating the sort space.

TEMPSIZE: Temporary Storage Size

TEMPSIZE defines the space available for the temp data set. The value may be in cylinders (a numeric value only) or blocks (a numeric value followed by a "B"). This parameter must be specified.

Optional Parameters

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter; ADAINV may also reduce a specified LPB value if the LU value is too small.

LWP: Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADAINV run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the sort data set, a smaller LWP value should be specified when defining descriptors for relatively small files.

The minimum work pool size depends on the sort data set's device type:

Sort Device	Minimum LWP	Minimum LWP
	Bytes	Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If one or both of the files being coupled is security protected, a valid password for the file (or files) must be specified with this parameter. If both files are password-protected, both must have the same password.

SORTDEV: Sort Device Type

ADAINV uses the sort data set to sort descriptor values. The SORTDEV parameter indicates the device type to be used for the sort data set. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter. See the **z/OS job control information** for specific SORTDEV considerations.

TEMPDEV: Temporary Storage Device Type

ADAINV uses the temp data set to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Example

```
ADAINV COUPLE
FILES=3,4,DESCRIPTOR='AA,BB'
```

Files 3 and 4 are to be coupled. Descriptor AA from file 3 and descriptor BB from file 4 are to be used as the basis for the coupling.

Temporary Space for File Coupling

An intermediate data set is generated for *each* of the files being coupled.

An entry is written to the data set for each record contained in the file. Each entry contains the ISN of the record (3 or 4 bytes, depending on the ISNSIZE defined for the files) and the value (in compressed form) of the descriptor being used as the basis for the coupling. If the descriptor is defined with the NU option, no entries are written for records in which the descriptor contains a null value. If the descriptor is a multiple-value field, an entry is written for each different value.

The space required for *each* of the intermediate data sets is a function of the number of records contained in each Adabas file, and the length and the number of the different values present for the coupling descriptor in each record.

Use the following equation to determine the space needed for an intermediate data set:

```
SP = RECS x UV x (ISNSIZE + (AVLEN x 4)
```

where

SP	intermediate data set space required (in bytes).
RECS	number of records contained in the coupled file.
UV	average number of unique values per record for the descriptor. If the descriptor is not defined with the NU option, UV is equal to or less than 1. If the descriptor is defined with the NU option, UV is equal to the average number of values per record minus the percentage of records that contain a null value. For example, if the average number of values per record is 1 and 20 percent of the values are null, UV is equal to 1 - 0.2 = 0.8.
ISNSIZE	length of ISNs in the file (3 or 4 bytes).
AVLEN	average length (after compression) of each value for the descriptor.

Example: Calculating Intermediate Space Requirements for File Coupling

The file being coupled has 3-byte ISNs and contains 50,000 records. The descriptor being used as the basis for coupling contains 1 value per record (with no null values) and has an average value length of 5 bytes.

```
SP = 50,000 x  1 x  (3 + (5 + 1))
SP = 50,000 x  9
SP = 450,000 bytes
```

Associator Coupling Lists

ADAINV matches the two lists, sorts each resulting list, and writes each list to the Associator coupling lists.

The temp data set stores the matched (coupled) ISNs for each file. An entry is written to the temp data set for each match found. The entry contains the ISN of each record containing a matching value.

ADAINV sorts the entries stored on the temp data set using the sort area and writes the sorted entries to the Associator coupling lists for file A. The same process is then repeated for file B.

The temp area size requirement depends on the number of matching values in the two files for the descriptor used to couple the files. Each match requires 6 or 8 bytes, depending on the ISNSIZE defined for the files.

The sort area generally requires twice the amount of space as that needed for the temp area.

File coupling is bidirectional rather than hierarchical in that two coupling lists are created with each list containing the ISNs which are coupled to the other file.

Example: Coupling Lists

Assume that 2 files containing the descriptors AA and BB, respectively, are to be coupled. The values for the first five records of each file are as follows:

File A		File B	
ISN	Field AA value	ISN	Field BB value
1	20	1	18
2	25	2	40
3	27	3	25
4	30	4	20
5	40	5	20

If the two files were coupled using AA and BB as the basis for the coupling, the resulting coupling lists would be:

File A			File B		
ISN in FILE B*	COUNT	COUPLED ISNs	ISN in FILE A*	COUNT	COUPLED ISNs
2	1	5	1	2	4,5
3	1	2	2	1	3
4	1	1	5	1	2
5	1	1			

** Internally, Adabas uses this field like a descriptor to determine the number and the ISNs of the coupled records.*

Space for Coupling Lists

The total space requirement for the coupling lists depends upon the number of common values that exist between the two descriptors used as the basis for the coupling.

The space requirement for *each common value* may be estimated as follows:

```
SP = 4a + 4b + 6ab
```

where

SP	space requirement for one common value (in bytes);
a	number of records in file A containing the common value;
b	number of records in file B containing the common value.

The total coupling list requirement is the sum of the space requirements of each common value.

Using sample files A and B as previously defined, space requirements per common value are

Common Value	Space Requirements
20	SP = 4(1) + 4(2) + 6(1 • 2) = 24 bytes
25	SP = 4(1) + 4(1) + 6(1 • 1) = 14 bytes
40	SP = 4(1) + 4(1) + 6(1 • 1) = 14 bytes

Total space required = 24 + 14 + 14 = 52 bytes

Example: Coupling List Space Requirements

Assume that 2 files are being coupled on the field ID. The values for ID are unique within each file. There are 5,000 common values in the coupled files.

Common Value	Space Requirements
n	SP = 4(1) + 4(1) + 6(1) SP = 14 bytes for one common value

There are 5,000 common values, each of which requires 14 bytes. The total space requirement for the coupling lists is 70,000 bytes.

Space Allocation

The coupling lists constructed by ADAINV are contained within the normal (NI) and upper (UI) index for each file being coupled. If the NI or UI component's logical extents currently allocated to the file are used up during ADAINV execution, ADAINV attempts to allocate an additional extent to the component. The size of the extent allocated is equal to 25 percent of the current total size of all logical extents currently assigned to the component. If insufficient space is available or if the maximum number of allocated extents has been reached for the component, ADAINV terminates with an error message.

94 INVERT: Create Descriptors

- Essential Parameters 494
- Optional Parameters and Subparameters 495
- Space Allocation for the INVERT Function 497
- Examples 497

The INVERT function creates descriptors, subdescriptors, superdescriptors, hyperdescriptors, phonetic descriptors or collation descriptors for existing fields in a file. Several descriptors may be created in a single ADAINV INVERT run, but only for a single file.

```
ADAINV   INVERT   FILES = file-num
                  SORTSIZE = size
                  TEMPSIZE = size
                  [FIELD = 'field-name [ , option]... ' ] ...
                  [COLDE = 'num, name [ , UQ [, XI ] ] = parent-field' ]
                  [HYPDE = 'num, name,length,format  [ ,option  ]... = parent-field, ...' ]
                  [PHONDE = 'name (field-name)' ]
                  [SUBDE = 'name [, UQ [, XI ] ] =  parent-field (begin, end)' ]
                  [SUPDE = 'name [, UQ [, XI ] ] = {parent-field (begin, end) } , ...' ]
                  [CODE = cipher-code ]
                  [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                  [LWP = {workpool-size | 1048576 } ]
                  [NOUSERABEND]
                  [PASSWORD = 'password' ]
                  [SORTDEV = {device-type  | ADARUN-device } ]
                  [TEMPDEV = {device-type  | ADARUN-device } ]
                  [TEST]
```

Essential Parameters

FILE: File Number

FILE specifies the file in which the descriptor(s) to be created is contained.

SORTSIZE: Sort Size

SORTSIZE specifies the space available for the sort data set or data sets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by a "B"). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference* documentation for more information on estimating the sort space.

TEMPSIZE: Temporary Storage Size

TEMPSIZE defines the space available for the temp data set. The value may be in cylinders (a numeric value only) or blocks (a numeric value followed by a "B"). This parameter must be specified.

Optional Parameters and Subparameters

CODE: Cipher Code

If the file specified with the FILE parameter is ciphered, an appropriate cipher code must be supplied using the CODE parameter.

FIELD/ COLDE/ HYPDE/ PHONDE/ SUBDE/ SUPDE: Define Descriptor(s)

These parameters may be used to define various types of descriptors. You must specify at least one descriptor definition for the file specified; you may specify more than one descriptor or type of descriptor.

Use the FIELD parameter to define one or more fields as descriptors; use the COLDE parameter for a collation descriptor; HYPDE parameter for a hyperdescriptor; PHONDE for a phonetic descriptor; SUBDE for a subdescriptor; and SUPDE for a superdescriptor.

If provided, a FIELD specification must come before any collation descriptor, hyper-, super-, sub-, or phonetic descriptor specification.

FIELD specifies an existing field (or fields) to be inverted. The field may be an elementary or multiple-value field and may be contained within a periodic group (unless the field is defined with the FI option).

If the descriptor is to be unique, specify "UQ" following the field name. If the uniqueness of the descriptor is to be determined with the index (occurrence number) excluded, specify "XI" as well.

Note: For Adabas expanded files, ADAINV can only detect unique descriptor violations within the specified component file. If an identical value exists for a unique descriptor in one of the other component files, ADAINV cannot detect it. You must therefore ensure that unique descriptor values remain unique throughout an expanded file.

Although multiple fields can be specified for inversion using the FIELD parameter, only one collation descriptor, hyper-, sub-, super-, or phonetic descriptor is defined per instance of its parameter. Multiple instances of the parameters are allowed per execution of ADAINV.

When inverting a sub- or superfield, the respective SUBDE or SUPDE parameter must specify the same parent fields that were specified when the field was created; otherwise, an error occurs. Begin and end values are taken from the original field definitions.

If a parent field with the NU option is specified, no entries are made in the inverted list for those records containing a null value for the field. For super- and hyperdescriptors, this is true regardless of the presence or absence of values for other descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the in-

verted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

For detailed information about the individual descriptor syntax, subparameter values, and coding, read *Field Definition Statements* in the description of the ADACMP utility, elsewhere in this guide.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32,760 bytes. The default depends on the ADARUN LU parameter; ADAINV may also reduce a specified LPB value if the LU value is too small.

LWP: Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADAINV run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the Sort data set, a smaller LWP value should be specified when defining descriptors for relatively small files.

The minimum work pool size depends on the Sort data set's device type:

Sort Device	Minimum LWP Bytes	Minimum LWP Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If the file specified with the FILE parameter is security protected, the file's password must be supplied using this parameter.

SORTDEV: Sort Device Type

ADAINV uses the sort data set to sort descriptor values. The SORTDEV parameter indicates the device type to be used for the sort data set. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter. See the z/OS job control information at the end of this section for specific z/OS SORTDEV considerations.

TEMPDEV: Temporary Storage Device Type

ADAINV uses the temp data set to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Space Allocation for the INVERT Function

The values for the field being inverted and the ISNs of the records containing the values are written to the inverted list (normal and upper indexes).

If either the normal or upper index logical extent is exhausted during ADAINV execution, ADAINV allocates an additional extent. The size of the extent allocated is equal to 25 percent of the current total size of all the normal index extents currently allocated to the file.

If sufficient space is not available for the new extent or if the maximum number of allocated extents has been reached, ADAINV terminates with an error message.

Examples

Example 1:

```
ADAINV INVERT
FILE=3,FIELD='AR',TEMPSIZE=10,SORTSIZE=5
```

Field AR in file 3 is to be made a descriptor.

Example 2:

```
ADAINV INVERT FILE=5,SUBDE='SA=AA(1,4)'
ADAINV            TEMPSIZE=6,SORTSIZE=3
```

Subdescriptor SA is to be created using field AA (positions 1-4) in file 5 as the parent field.

Example 3:

```
ADAINV INVERT FILE=6,SUPDE='SB=AA(1,4),AB(1,1)'
ADAINV            TEMPSIZE=5,SORTSIZE=3
```

Superdescriptor SB is to be created using fields AA (positions 1-4) and AB (position 1) in file 6.

Example 4:

```
ADAINV INVERT FILE=1,PHONDE='XX(AA)'
ADAINV            TEMPSIZE=5,SORTSIZE=3
```

A phonetic descriptor XX is created using field AA as the source field.

Example 5:

```
ADAINV INVERT FILE=6,COLDE='1,Y1=AA'
ADAINV            TEMPSIZE=5,SORTSIZE=4
```

Collation descriptor CDX=01 named Y1 is created using AA as the source field.

95 JCL/JCS Requirements and Examples

- Collation with User Exit 500
- BS2000 500
- z/OS 503
- z/VM 504
- VSE 506

This section describes the job control information required to run ADAINV with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADAINV execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

```
ADARUN CDXnn= exit-name
```

where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference* documentation for more information.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes (see Note).
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAINV parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*

Data Set	Link Name	Storage	More Information
ADAINV messages	SYSLST/ DDDRUCK		*Messages and Codes*

Note: Performance can be improved when sorting large files if the sort data set is split across two volumes. If two data sets are specified, they must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.

ADAINV JCL Examples (BS2000)

Couple Files

In SDF Format:

```
/.ADAINV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A I N V COUPLE FIELD (REFLECTIVE)
 /REMARK *
 /ASS-SYSLST L.INV.COUP
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
 /SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
 ADAINV COUPLE FILE=1,3,DESCRIPTOR= AA,AA
 ADAINV TEMPSIZE=100,SORTSIZE=50
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAINV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A I N V COUPLE FIELD (REFLECTIVE)
 /REMARK *
 /SYSFILE SYSLST=L.INV.COUP
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSOR ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
 /FILE ADAyyyyy.SORT ,LINK=DDSORTR1
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
 ADAINV COUPLE FILE=1,3,DESCRIPTOR= AA,AA
```

```
 ADAINV TEMPSIZE=100,SORTSIZE=50
 /LOGOFF NOSPOOL
```

Invert File

In SDF Format:

```
/.ADAINV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A I N V INVERT FIELD (REFLECTIVE)
 /REMARK *
 /ASS-SYSLST L.INV.INVE
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
 /SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
 ADAINV INVERT FILE=1
 ADAINV TEMPSIZE=100,SORTSIZE=50
 ADAINV FIELD= AC
 ADAINV SUPDE= S1,UQ=AA(1,3),AD(2,4)
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAINV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A I N V INVERT FIELD (REFLECTIVE)
 /REMARK *
 /SYSFILE SYSLST=L.INV.INVE
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSOR ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
 /FILE ADAyyyyy.SORT ,LINK=DDSORTR1
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
 ADAINV INVERT FILE=1
 ADAINV TEMPSIZE=100,SORTSIZE=50
 ADAINV FIELD= AC
 ADAINV SUPDE= S1,UQ=AA(1,3),AD(2,4)
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes (see Note).
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	DDCARD	reader	*Operations*
ADAINV parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAINV messages	DDDRUCK	printer	*Messages and Codes*

Note: Performance can be improved when sorting large files if the sort data set is split across two volumes, but this is difficult to accomplish under OS. Two sort data sets may be specified instead. They must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.

*

ADAINV JCL Example (z/OS)

Couple Files

Refer to ADAINVCO in the JOBS data set for this example.

```
//ADAINVCO  JOB
//*
//*    ADAINV:  COUPLE FILES
//*
//INV       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <===== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <===== DATA

//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <===== WORK
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <===== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <===== SORT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
```

```
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAINV COUPLE FILE=2,3,DESCRIPTOR='BB,BB'
ADAINV        TEMPSIZE=100,SORTSIZE=100
/*
```

Invert File

Refer to ADAINV in the JOBS data set for this example.

```
//ADAINVDE  JOB
//*
//*     ADAINV:  INVERT A FIELD TO A DE
//*
//INV       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <===== ASSO
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <===== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <===== SORT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAINV INVERT FILE=1
ADAINV FIELD='AC'
ADAINV SUPDE='S1,UQ=AA(1,3),AD(2,4)'
ADAINV TEMPSIZE=100,SORTSIZE=100
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes.*

Data Set	DD Name	Storage	More Information
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAINV parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAINV messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

** Performance can be improved when sorting large files if the sort data set is split across two volumes, but this is difficult to accomplish under CMS. Two sort data sets may be specified instead. They must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.*

ADAINV JCL Examples (z/VM)

Couple Files

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDPRINT,DSN=ADAINV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAINV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNINV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAINV.CONTROL,MODE=A
ADARUN
```

Contents of RUNINV CONTROL A1:

```
ADARUN PROG=ADAINV,DEVICE=dddd,DB=yyyyy
```

Contents of ADAINV CONTROL A1:

```
ADAINV COUPLE FILE=1,3,DESCRIPTOR='AA,AA'
ADAINV        TEMPSIZE=100,SORTSIZE=50
*
```

Invert File

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDPRINT,DSN=ADAINV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADAINV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNINV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAINV.CONTROL,MODE=A
ADARUN
```

Contents of RUNINV CONTROL A1:

```
ADARUN PROG=ADAINV,DEVICE=dddd,DB=yyyyy
```

Contents of ADAINV CONTROL A1:

```
ADAINV INVERT FILE=1
ADAINV        TEMPSIZE=100,SORTSIZE=50
*
ADAINV        FIELD='AC'
ADAINV        SUPDE='S1,UQ=AA(1,3),AD(2,4)'
```

VSE

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Intermediate storage	TEMPR1	disk	*	
Sort area	SORTR1	disk	*	
Recovery log (RLOG)	RLOGR1	disk	*	Required with recovery log (RLOG) option
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAINV parameters	-	reader	SYSIPT	
ADARUN messages	-	printer	SYSLST	Messages and Codes
ADAINV messages	-	printer	SYS009	Messages and Codes

** Any programmer logical unit can be used.*

ADAINV JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures (PROCs).

Couple Files

Refer to member ADAINVCO.X for this example.

```
* $$ JOB JNM=ADAINVCO,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAINVCO
*       COUPLE FILES
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAINV COUPLE FILE=2,3,DESCRIPTOR='BB,BB'
ADAINV        TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

Invert File

Refer to member ADAINV.X for this example.

```
* $$ JOB JNM=ADAINV,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAINV
*       INVERT A FIELD TO A DESCRIPTOR
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAINV INVERT FILE=1
ADAINV FIELD='AC'
ADAINV SUPDE='S1,UQ=AA(1,3),AD(2,4)'
ADAINV TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

XI ADALOD Utility: File Loader

This chapter covers the following topics:

Functional Overview
LOAD: Load a File
UPDATE : ADD/Delete Records
Loader Storage Requirements and Use
Temp Data Set Space Usage
ADALOD Space/Statistics Report
JCL/JCS Requirements and Examples

96 Functional Overview

The **ADALOD LOAD function** loads a file into the database. Compressed records produced by the ADACMP or ADAULD utility may be used as input.

ADALOD loads each compressed record into Data Storage, builds the address converter for the file, and enters the field definitions for the file into the field definition table (FDT). ADALOD also extracts the values for all descriptors in the file together with the ISNs of all records in which the value is present, to an intermediate data set. This data set is then sorted into value/ISN sequence and then entered into the Associator inverted lists.

The **ADALOD UPDATE function** is used to add or delete a large number of records to/from an Adabas file. The UPDATE function requires considerably less processing time than the repetitive execution of the Adabas add/delete record commands. Records to be added may be the compressed records produced by the ADACMP or ADAULD utility. The ISNs of records to be deleted can be provided either in an input data set or by using control statements.

Records may be added and other records deleted during a single execution of ADALOD.

Note: You can load files from input data sets that were created by ADAULD or ADACMP utilities running under Adabas 5.1 - 5.3, 6.1, 6.2, 7.1, 7.2, or 7.4. In addition, you can load Adabas 8 files into Adabas databases releases after 5.3, as long as the file to be loaded does not use any of the new features that were introduced after the version used to load the file.

97 LOAD: Load a File

- Essential Parameters 515
- Optional Parameters and Subparameters 517
- Examples 531
- LOAD Data and Space Requirements 533
- Loading Expanded Files 536
- Loading Multiclient Files 538

Use the LOAD function to load a file into a database.

```
ADALOD LOAD  FILE = file-number [ , filetype ]
             DSSIZE = size
             MAXISN = max-number-of-records [MAXISN2 = max-number-of-secondary-spanned-records ]
             SORTSIZE = size
             TEMPSIZE = size
             [ACRABN = starting-rabn ] [AC2RABN = starting-rabn ]
             [ADAMFILE  ADAMDE = {field | ISN } [ADAMOFLOW = size ] [ADAMPARM = { number | 0 } ] ]
             [ALLOCATION = { FORCE  | NOFORCE } ]
             [ANCHOR = file-number  MINISN =  lowest-allocated-isn, NOACEXTENSION ]
             [ASSOPFAC = {padding-factor  | 10 } ] [ASSOVOLUME = ' Associator-extent-volume' ]
             [{BASEFILE | LOBFILE} = file-number ]
             [DATAFRM = { YES| NO }]
             [DATAPFAC = { padding-factor  | 10} ] [DATAVOLUME = 'Data-Storage-extent-volume' ]
             [DSDEV = device-type ]  [DSRABN = start-rabn ] [DSREUSE = { YES | NO }]
             [ETID =  owner-id ]
             [IGNFDT]
             [INDEXCOMPRESSION = { YES | NO }]
             [ISNREUSE = { YES | NO ]  [ISNSIZE = { 3 | 4 } ]
             [LIP = { isn-pool-size |  2000 } ]
             [LOWNERID = { owner-id-length |  0 } ]
             [LWP = { work-pool-size  |  1048576 } ]
             [MAXDS = {max-DS-secondary-allocation  |  no-limit  } ]
             [MAXNI = { max-NI-secondary-allocation  |  no-limit  } ]
             [MAXRECL = {max-compressed-record-length |  max-possible } ]
             [MAXUI= { max-UI-secondary-allocation  |  no-limit  } ]
             [MINISN = { lowest-allocated-isn  |  1 } ]
             [MIXDSDEV]
             [NAME = { name  |  TESTFILE  } ]
             [NIRABN = start-rabn ] [NISIZE =  size  ]
             [NOACEXTENSION]
             [NOUSERABEND]
             [NUMREC = { max-number-of-records-to-load  |  all-records } ]
             [PGMREFRESH = { YES |  NO  } ]
             [REPLICATOR]
             [RESTART]
               RPLTARGETID = ' reptor-target-id '
                  [RPLDSBI]
                  [RPLKEY = ' primary-key-for-replication ' ]
                  [RPLLOAD = { YES | FILE | NO } ]
             [RPLUPDATEONLY = { YES | NO }]
             [SKIPREC = { number |  0 } ]
             [SLOG]
             [SORTDEV = { device-type  | ADARUN-device  } ]
             [TEMPDEV = { device-type  | ADARUN-device  } ]
             [TEST]
             [UIRABN = start-rabn ]  [UISIZE= size ]
             [UQDE =  descriptor-list ]
             [USERISN = { YES | NO } ]
             [VERSION = { 4 |  5  | 6 | 7 } ]
```

Essential Parameters

DSSIZE: Extent Size for Data Storage

DSSIZE is the count of blocks or cylinders to be assigned to the file's Data Storage logical extent. This value must be specified. Block values must be followed by a "B" (for example, "5000B").

The number can be taken directly from the Space Requirements report produced by the ADACMP utility. If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If a small number of records is being loaded now and a larger number of records is to be added later, the ADACMP report value should be increased in proportion to the total records to be added; otherwise, the space allocation for Data Storage (the original and four additional extents) may not be large enough to accommodate the records to be added. The file must then be unloaded and reloaded (or reordered) to increase the Data Storage space allocation. For more information, see the section *LOAD File Space Allocation* in the ***LOAD Data and Space Requirements*** section.

FILE: File Number, File Type

FILE specifies the Adabas file number and file type to be assigned to the file.

The number specified must not be currently assigned to another file in the database, unless that file was first deleted using the KEEPFDT parameter (see ADADBS DELETE function). The number must not be greater than the maximum file number defined for the database; for a checkpoint, security, or trigger, or system file, the number must be 5000 or lower. File numbers may be assigned in any sequence.

The file type is optional and is used to indicate that the file is an Adabas system file or an Adabas LOB file. One of the following keywords may be specified:

CHECKPOINT	Adabas checkpoint system file
LOB	Adabas *LOB file*
SECURITY	Adabas security system file
SYSFILE	Adabas system file
TRIGGER	Adabas trigger system file

Notes:

1. An existing checkpoint system file created using the ADADEF utility cannot be overwritten.
2. The security system file is required if Adabas Security is to be used.
3. In an Adabas Transaction Manager (ATM) database, SYSFILE numbers 5 and 6 are reserved for the ATM nucleus. For Adabas version 7.1, these file numbers cannot be changed. The file numbers are more flexible in subsequent versions of Adabas.

4. Use the following parameters to load the ATM system files on an ATM database (ADARUN DTP=TM): `ADALOD LOAD FILE=5,SYSFILE ,ADALOD LOAD FILE=6,SYSFILE`
5. If CHECKPOINT, SECURITY, or TRIGGER is specified, the contents of //DDEBAND are ignored.
6. No //DDEBAND data set need be supplied if you are loading an empty LOB file.
7. CHECKPOINT, SECURITY, or SYSFILE files can be deleted only by the ADADBS DELETE function running as the only Adabas user; deleting a system file terminates Adabas when deletion is completed.
8. Adabas allows a maximum of eight (8) system files.
9. If a *LOB file* is being loaded, the parameters ADAMFILE, ANCHOR, LOWNERID, NUMREC, SKIPREC, and UQDE cannot be specified in the ADALOD LOAD run.

MAXISN: Maximum ISN Count

The MAXISN parameter is required. Specify the maximum number of ISN mappings in the address converter (AC). ADALOD determines the number of ISN mappings to allow space for in the AC using the calculation:

```
(MAXISN - MINISN) + 1
```

There is no default value.

The MAXISN and MINISN values you specify are used to calculate the initial number of AC blocks to allocate during the ADALOD execution. Depending on the size of RABNs in the database (which is determined by the ADADEF DEFINE parameter RABNSIZE), each RABN requires 3 or 4 bytes in the AC. In addition, the block size of each AC block depends upon the device type of the Associator. So the number of AC blocks that should be allocated is affected by the number of ISN mappings allowed, the RABN size, and the block size of the Associator device.

To calculate the number of AC blocks that must be allocated, ADALOD uses the following calculation and rounds up to the nearest integer:

```
(#-of-ISN-mappings X RABN-size)/device-blocksize
```

For example, assume the RABN size for the database is set to "3" (the ADADEF DEFINE RABNSIZE parameter) and that the block size of the device on which the Associator resides is 2544 bytes. If MAXISN=1000 and MINISN=1, ADALOD calculates that the actual number of ISNs to be mapped as (1000 -1)+1 (`MAXISN - MINISN +1`), or 1000. It then multiplies 1000 by three (the RABN size), to get 3000 bytes. Finally, it divides 3000 by 2544 (the block size of the device), resulting in a value of roughly 1.18, which it rounds up to two. So ADALOD determines that two AC blocks should be allocated for this ADALOD run. (Note that on the corresponding ADAREP report, the "MAX-ISN Expected" value would not be listed as 1000; instead it is listed as the actual number of ISNs that would fit into two AC blocks – in this case about 1694.)

If more than *(MAXISN - MINISN) + 1* records are to be loaded, and if NOACEXTENSION is *not* specified, ADALOD increases the MAXISN value and allocates an additional AC extent.

MAXISN does not specify the maximum number of records that can be loaded into the file. The maximum number of records that Adabas permits in a file depends on the ISNSIZE parameter, which specifies whether ISNs in the file are 3 bytes or 4 bytes long. (If ISNSIZE=3, Adabas permits up to 16,777,215 records. If ISNSIZE=4, Adabas permits up to 4,294,967,294 records.)

SORTSIZE: Sort Size

SORTSIZE specifies the space available for the sort data set or data sets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by a "B"). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference* documentation for more information on estimating the sort space.

TEMPSIZE: Temporary Storage Size

TEMPSIZE specifies the size of the temp data set for the file. The Temp size equals the total of TEMP space required for each descriptor in the file; see the section *LOAD File Space Allocation* in the ***LOAD Data and Space Requirements*** section for more information. The size can be either in cylinders or blocks (followed by a "B").

Optional Parameters and Subparameters

ACRABN/ AC2RABN/DSRABN/ NIRABN/ UIRABN: Starting RABN

Causes space allocation for the address converter (ACRABN), secondary address converter (AC2RABN), Data Storage (DSRABN), the normal index (NIRABN), or the upper index (UIRABN) to begin at the specified RABN.

ADAMFILE: File to Be Loaded with ADAM Option

ADAMFILE specifies the file is to be loaded using the ADAM option.

If this parameter is specified, the Data Storage RABN for each input record is calculated using a randomizing algorithm, the result of which is based on the value of the ADAM descriptor in each record. See the ADAMER utility description for additional information about using the ADAM option. If ADAMFILE is specified, ADAMDE must also be specified.

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the ADAMFILE parameter cannot be specified in the ADALOD LOAD run.

ADAMDE: ADAM Key

ADAMDE specifies the field to be used as the ADAM key.

The ADAM descriptor must be defined in the field definition table (FDT). The descriptor must have been defined with the UQ option, and *cannot*

- be a sub-, super-, hyper-, collation, or phonetic descriptor;
- be a multiple-value field;
- be a field within a periodic group;
- be variable length;
- specify the null suppression (NU) option.

If the ISN of the record is to be used as the ADAM key, ADAMDE=ISN must be specified.

This parameter must be specified when the ADAM option has been selected for the file being loaded with the ADAMFILE parameter.

ADAMOFLOW: Overflow Area Size for ADAM File

ADAMOFLOW is the size of the Data Storage area to be used for ADAM file overflow. The ADAMOFLOW value applies only if the ADAM option has been selected for the file being loaded (see ADAMFILE parameter).

ADALOD will choose a prime number which is less than DSSIZE minus ADAMOFLOW (in blocks). This prime number is used to compute the Data Storage RABN for each record. If a record does not fit into the block with the computed RABN, it is written to the next free RABN in the overflow area.

ADAMPARM: Bit Truncation for ADAM File

ADAMPARM specifies the number of bits to be truncated from the ADAM descriptor value before it is used as input to the ADAM randomizing algorithm. A value in the range 1-255 may be specified. If this parameter is omitted, a value of 0 bits (no truncation) will be used.

This parameter achieves a type of record clustering, with nearly equal ADAM keys. ADAMPARM can be specified only when the ADAMFILE parameter has also been specified.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

If insufficient space can be obtained according to the placement parameters DSRABN, NIRABN, or UIRABN, only the first extent will be made there and the rest (until the fifth extent) will be made elsewhere. But if the placement parameter ACRABN is used with ALLOCATION=FORCE, the complete space has to be available there; otherwise, the utility terminates with an error.

ANCHOR: Expanded Component/ Anchor File

ANCHOR defines the base (anchor) file for either an existing or a new expanded file. If the file defined by ANCHOR is the same as that defined by the FILE parameter, the loaded file

becomes the physical base (anchor) file for a new expanded logical file. Otherwise, the FILE file becomes a new component of the expanded file defined by ANCHOR.

If ANCHOR specifies a file that is not part of an expanded file, the LOAD operation defines this file and the file specified by the FILE parameter as a new expanded file. It also sets the NOACEXTENSION indicator for the file specified by ANCHOR.

If ANCHOR specifies the anchor file of an already existing expanded file, the LOAD operation adds the file specified by FILE to the expanded file.

Note: When loading a new file to an existing expanded file, you must have exclusive update use of the anchor file as well as the file being added. This can be achieved by locking the anchor file for utility use.

Both the file specified by ANCHOR and the file specified by FILE must have the same field definition table (FDT) structure. The maximum record length (MAXRECL parameter) and any file security definitions must also be the same.

If ANCHOR is specified, the MINISN and NOACEXTENSION parameters must also be specified. Coupled files or multiclient files cannot be part of expanded files.

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the ANCHOR parameter cannot be specified in the ADALOD LOAD run.

ASSOPFAC: Associator Padding Factor

ASSOPFAC defines the padding factor to be used for each Associator block. If not specified, the default padding factor is 10.

The value specified represents the percentage of each Associator block (padding area) that is not to be used during the loading process. The padding area is reserved for use when additional entries must be added to the block for new descriptor values or new ISNs for existing values, thereby avoiding the overhead caused by relocating overflow entries into another block.

A value in the range 1-90 may be specified. The number of bytes contained in an Associator block, minus the number of bytes reserved for padding, must be larger than the largest descriptor value contained in the file, plus 10 bytes.

A small padding factor (1-10) should be specified if little or no descriptor updating is planned. A larger padding factor (10-50) should be specified if a large amount of updating including addition of new descriptor values (or new ISNs) is planned.

ASSOVOLUME: Associator Extent Volume

Note: The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME specifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated. If the requested number of blocks cannot be found on the

specified volume, ADALOD retries the allocation while disregarding the ASSOVOLUME parameter.

Note: If there are five or more blocks of unused ASSO space on the specified volume, ADALOD allocates these blocks; if this is not enough space, it ends with ERROR-060. If there are no free blocks remaining on the specified volume, ADALOD tries to allocate space on another volume.

If ACRABN, UIRABN, or NIRABN is specified, ADALOD ignores the ASSOVOLUME value when allocating the corresponding extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADALOD's default allocation rules.

BASEFILE: Base File Number

BASEFILE specifies the file number of the *base file* associated with the *LOB file* you are loading. This parameter is only used when loading LOB files.

For more information, read *Getting Started with Large Object (LB) Fields*, in *Adabas DBA Tasks Manual*.

DATAFRM: Overwrite ADAM Data Storage

DATAFRM controls overwriting of an ADAM file's Data Storage during loading. DATAFRM=YES (the default) forces ADALOD to reformat the Data Storage area when the file is loaded; DATAFRM=NO prevents reformatting, and is recommended when loading relatively few records because the load operation may run significantly faster. Specifying NO, however, assumes that the Data Storage area was previously formatted with the ADAFRM utility specifying FROMRABN.

Caution: Specify DATAFRM=NO with care. If the primary Data Storage area was incorrectly formatted, later file processing could cause errors and unpredictable results.

DATAPFAC: Data Storage Padding Factor

DATAPFAC is the padding factor to be used for each Data Storage physical block. A percentage value in the range 1-90 may be specified. If not specified here, the default padding factor is 10.

A small padding factor (1-10) should be specified if little or no record expansion is expected. A larger padding factor (10-50) should be specified if a large amount of updating is planned that will expand the logical records.

The percentage value specified represents the portion of each Data Storage block (padding area) to be reserved during the loading process for later record expansion. The padding area is used when any given logical record within the block requires additional space as the result of record updating, thereby avoiding the overhead that would be needed to relocate the record to another block.

Since records loaded into a file can be different lengths, the padding factor cannot be exactly the percentage specified in each block. Adabas balances the size of the padding area for the different record lengths to the extent that at least 50 bytes remain in a block.

Example:

A block size is 1000 bytes; the padding factor is 10%. The space available for loading records (block size - padding-area) is therefore 900 bytes.

After loading some records, 800 bytes of the block have been used. The next record is 170 bytes long. This record cannot be loaded into the current block because less the 50 bytes would remain in the block after the record was loaded. Therefore, the record is loaded into the next block.

The current block remains filled to 800 bytes. The difference between 800 and 900 bytes (that is, -100 bytes) is used for balancing.

Suppose the next record had been 150 bytes instead of 170 bytes, and assume that the cumulative balancing value at that point in time is a negative number of bytes. The 150-byte record would be loaded because 50 bytes would remain in the block after the record was loaded (1000 - 950).

However, 50 bytes of the padding area would have been used (900 - 950) leaving +50 bytes for balancing.

For files loaded with the ADAM option, a new record is loaded into its calculated Data Storage block if space is available in the block (including the padding area). Records that cannot be stored in their calculated block are stored in another block (in this case, the padding area is not used).

DATAVOLUME: Data Storage Extent Volume

Note: The value for DATAVOLUME must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) is to be allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADALOD retries the allocation while disregarding the DATAVOLUME value.

If DSRABN is specified, DATAVOLUME is ignored for the related file. If DATAVOLUME is not specified, the Data Storage space is allocated according to ADALOD's default allocation rules.

DSDEV: Data Storage Device Type

DSDEV specifies the device type on which the file's Data Storage is to be loaded. There is no default value; if DSDEV is not specified, an arbitrary device type is used.

DSREUSE: Data Storage Reusage

DSREUSE indicates whether Data Storage space which becomes available is to be reused. The default is YES.

ETID: Multiclient File Owner ID

The ETID parameter assigns a new owner ID to all records being loaded into a multiclient file. It specifies the user ID identifying the owner of the records being loaded. The owner ID assigned to the records is taken from the user profile of the specified user ID.

The ETID parameter must be specified if the file is to be loaded as a multiclient file (see the **LOWNERID parameter discussion**) and the input file contains no owner IDs; that is, the input file was not unloaded from a multiclient source file.

ETID is optional if the input file was unloaded from a multiclient source file. In this case, the loaded records keep their original owner IDs.

The ETID parameter must not be specified when loading a non-multiclient file.

Note: If the ETID parameter is used, the ADALOD utility requires an active nucleus. The nucleus will translate the ETID value into the internal owner ID value.

IGNFDT: Ignore Old FDT

When a file is deleted using the ADADBS DELETE function with the KEEPFDT parameter, the field definition table (FDT) remains in the Associator. When the file is again reloaded and IGNFDT is not specified, ADALOD compares the file's old FDT with the new one (security information is not compared). If both FDTs are identical, ADALOD loads the file and replaces the old FDT with the new FDT. If the FDTs are not identical, the old FDT is kept and the ADALOD operation ends with an error message.

Specifying the IGNFDT parameter causes ADALOD to ignore any existing (old) FDT for the file; no comparison is made. The new FDT replaces the old FDT, and ADALOD loads the file.

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index of the file is loaded in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is not specified, ADALOD obtains the default value from the sequential input file. If the input file was created using

- ADACMP, the default value is NO.
- ADAULD, the value of the file at the time of the unload is taken as the default.

ISNREUSE: ISN Reusage

ISNREUSE indicates whether or not an ISN freed as the result of deleting records may be reassigned to a new record. The default is NO.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE indicates whether ISNs in the file are 3 or 4 bytes long. The default is 3 bytes.

LIP: ISN Buffer Pool Size

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 2000 bytes.

LIP can be used to decrease the number of address converter I/Os during loading when the USERISN=YES and the user-supplied ISNs are unsorted. Optimum performance is obtained if LIP specifies a buffer size large enough to hold all ISNs to be processed.

The length of one input record is ISNSIZE + RABNSIZE + 1. Thus the entry length is at least 7 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 9 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

LOBFILE: LOB File Number

LOBFILE specifies the file number of the *LOB file* associated with the *base file* you are loading. This parameter is only used when loading base files.

For more information, read *Getting Started with Large Object (LB) Fields*, in *Adabas DBA Tasks Manual*.

LOWNERID: Internal Owner ID Length for Multiclient File

The LOWNERID parameter specifies the length of the internal owner ID values assigned to each record for multiclient files. Valid length values are 0-8. If the LOWNERID parameter is not specified, its default value is the length of the owner IDs in the input file.

The specified or default value of the LOWNERID parameter determine whether a file is to be loaded as a multiclient or a non-multiclient file. If the effective LOWNERID value is zero, the file is loaded as a normal, non-multiclient file; if it is nonzero, the file is loaded as a multiclient file.

In combination with the ETID parameter, the LOWNERID parameter can be used to

- reload a non-multiclient file as a multiclient file;
- increase/decrease the length of the owner ID for the file; or
- remove the owner ID from the records of a file.

The following table shows the possible combinations of the LOWNERID parameter and the owner ID length in the input file.

LOWNERID Parameter Setting	Owner ID Length Value in Input File	
	0	2
0	Keep as a non-multiclient file	Convert to a non-multiclient file
1	Set up multiclient file (ETID)	Decrease owner ID length
2	Set up multiclient file (ETID)	Keep owner ID length
3	Set up multiclient file (ETID)	Increase owner ID length
(not specified)	Keep as a non-multiclient file	Keep as a multiclient file

When loading a multiclient file (the specified or default value of LOWNERID is non-zero), the ETID parameter can be specified to assign a new owner ID to all records being loaded. If the input file already contains owner IDs and ETID is omitted, all records keep their original owner IDs.

Where the table indicates the ETID parameter in the "Owner ID Length...0" column, the ETID parameter is mandatory, as there are no owner IDs given in the input file.

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the LOWNERID parameter cannot be specified in the ADALOD LOAD run.

LWP: Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADALOD run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the sort data set, a smaller LWP value should be specified when loading relatively small files.

The minimum work pool size depends on the sort data set's device type:

Sort Device	Minimum LWP	Minimum LWP
	Bytes	Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

MAXDS/ MAXNI/ MAXUI: Maximum Secondary Allocation

Specifies the maximum number of blocks per secondary extent allocation for Data Storage (MAXDS), normal index (MAXNI), or upper index (MAXUI). The value specified must be in blocks (for example, MAXNI=8000B) and cannot be more than 65535B. If no limit is specified, no limit is assumed (the default).

MAXISN2: Allocate Secondary Address Converter RABNs to Account for Maximum Secondary ISNs

The MAXISN2 parameter is optional, regardless of whether or not spanned records exist in the ADALOD input file or not. Use this parameter to specify the desired size of the secondary address converter (AC2) in ISNs. The secondary address converter is used to map secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADALOD determines the number of ISN mappings for which to allow space in the secondary AC using the following method:

1. If a secondary AC is not yet allocated, one secondary AC block is allocated.
2. If the current MAXISN2 setting is greater than or equal to four times the MAXISN value (the current maximum primary ISNs expected), the same algorithm is used to determine the number of secondary ISNs as is used to allocate additional primary ISNs.
3. If none of the above conditions is met, then secondary AC space is allocated as the smaller of the following two calculations:

- The product of 10 times the old MAXISN2 setting (10 x oldMAXISN2)
- The sum of the old MAXISN2 setting and the MAXISN setting (oldMAXISN2 + MAXISN)

There is no default value.

MAXRECL: Maximum Compressed Record Length

MAXRECL specifies the maximum compressed record length permitted for the file. The default is the maximum length supported by the device type being used.

MINISN: Lowest ISN Count

This parameter specifies the lowest number of ISNs that can be assigned in the file. The default is 1.

The main purpose of MINISN is to assign the low end of the ISN range for a component file of an Adabas expanded file. MINISN is required when ANCHOR is specified for an expanded file.

Use MINISN to avoid wasting Associator space in files where all records are assigned ISNs significantly greater than 1. For example, a savings bank uses account numbers as ISN numbers, and the lowest account number is 1,000,001. Specifying MINISN = 1 000 001 stops Adabas from allocating address converter space for ISNs 1-999 999, which would be unused. For more information, see the description of the MAXISN parameter.

MIXDSDEV: Data Storage Mixed Device Types

MIXDSDEV allows the allocation of secondary Data Storage extents on different device types, and therefore with different block lengths. If MIXDSDEV is not specified (the default), Data Storage extents for the specified file must all be on the same device type.

NAME: File Name

NAME is the name to be assigned to the file. This name appears, along with data pertaining to this file, on the Database Status Report produced by the ADAREP utility. The maximum number of characters permitted is 16. The default name assigned is TESTFILE.

If the file name contains special characters or embedded blanks, the name must be enclosed within apostrophes ('...'), which themselves must be doubled if one is included in the name; for example, 'JAN''S FILE'.

NISIZE: Normal Index Size

NISIZE specifies the number of blocks or cylinders to be assigned to the normal index. A block value must be followed by a "B" (for example, "5500B").

If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If the NISIZE parameter is omitted:

- ADALOD determines the space allocation for the normal index based on a sampling of records taken from the input data set. Since this calculation requires additional CPU time and I/O operations, Software AG recommends setting this parameter if the size is known so that no estimation is performed.

- and INDEXCOMPRESSION=YES is set, the index size estimation made by ADALOD does not consider the index compression as it has no knowledge of the rate of compression to be expected. ADALOD may thus allocate a larger index than necessary.

If a small number of records is being loaded and a larger number of records is to be added later, the NISIZE parameter should be set to increase the Normal Index to accommodate the total record amount. For more information, see the section *LOAD File Space Allocation* in the ***LOAD Data and Space Requirements*** section.

NOACEXTENSION: Limit Address Converter Extents

If NOACEXTENSION is specified, the MAXISN defined for this file cannot be increased in the future. No additional address converter (AC) extents will be created. NOACEXTENSION applies mainly to component files comprising Adabas expanded files; if ANCHOR is specified, NOACEXTENSION must also be specified.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Limit Number of Records to Be Loaded

NUMREC specifies the limit on the number of records to be loaded. If NUMREC is specified, ADALOD stops after processing the specified number of records (unless an end-of-file condition on the input data set ends ADALOD operation before that time). This option is most often used to create a subset of a file for test purposes. If this parameter is omitted, all input records are processed.

If the input data set contains more records than specified by NUMREC, ADALOD processes the number of records specified by NUMREC and then ends with condition code 4.

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the NUMREC parameter cannot be specified in the ADALOD LOAD run.

PGMREFRESH: Program-Generated File Refresh

PGMREFRESH specifies whether a user program is allowed to perform a refresh operation on the file being loaded. If PGMREFRESH is specified, a refresh can be made using an E1 command, or an equivalent call to the nucleus.

REPLICATOR: Load an Adabas Replicator System File

The `REPLICATOR` parameter is an Event Replicator for Adabas parameter for an Event Replicator Server database. It is a file type parameter. Use this parameter to load the Replicator system file into the Event Replicator Server database.

The Replicator system file stores the Event Replicator initialization parameters. When it is loaded into the Event Replicator database, it can be read during Event Replicator Server startup.

You can modify the initialization parameters in the Replicator system file using the Adabas Event Replicator Subsystem. If the Replicator system file cannot be found at startup, the Event Replicator initialization parameters are read from the DDKARTE statements of the Event Replicator Server startup job.

The REPLICATOR parameter may not be specified when loading a file on a non-Event Replicator database. The contents of DD/EBAND are ignored when loading a Replicator system file. For more information about Adabas system files, read about the FILE parameter of the ADALOD LOAD function, elsewhere in this section.

The REPLICATOR parameter may not be specified in the same ADALOD LOAD as the RPLTARGETID, or any of its associated parameters.

RESTART: Restart Interrupted ADALOD Execution

RESTART forces an interrupted ADALOD run to be restarted, beginning with the last *restart point* reached before the interruption. The *restart point* is the latest point of execution that can be restored from the Temp data set.

If ADALOD is interrupted by a defined error condition, ADALOD issues a message indicating whether or not a restart is possible.

When restarting the ADALOD operation, the following parameters may be changed:

- TEMPSIZE can be increased to make the temp data set larger. Note, however, that the temp data set content contains information necessary for the restart operation, and therefore *must not be changed* ;
- The SORTSIZE and SORTDEV parameters and the sort data set can be changed.

No other parameters can be changed. The DDEBAND/EBAND and DDFILEA/FILEA data sets must remain the same.

RPLDBSI: Before Image for Data Storage

The RPLDSBI parameter is an Event Replicator for Adabas parameter for an Adabas database that turns on the collection of before images of data storage during an update command to the file. Parameter RPLDSBI may only be specified if the RPLTARGETID parameter is also specified. Specify RPLDSBI to turn on the collection of data storage before images during an update. For more information about how this setting is used in Adabas database processing during replication, read *Adabas Nucleus Replication Setup* and *Detailed Adabas Nucleus Processing* in the *Event Replicator for Adabas Concepts Manual* documentation.

The RPLDSBI parameter may not be specified in the same ADALOD LOAD run as the REPLICATOR parameter.

RPLKEY: Primary Key for Replication

The RPLKEY parameter is an Event Replicator for Adabas parameter for an Adabas database that specifies the primary key for replication. This parameter may only be specified if the RPLTARGETID parameter is also specified. For more information about how this primary key is used in Adabas database processing during replication, read *Adabas Nucleus Replication Setup*

and *Detailed Adabas Nucleus Processing* in the *Event Replicator for Adabas Concepts Manual* documentation.

The RPLKEY parameter may not be specified in the same ADALOD LOAD run as the REPLICATOR parameter.

RPLLOAD: Replicate Load Data

The ADALOD LOAD RPLLOAD parameter is an Event Replicator for Adabas parameter that can be specified for the Adabas database files when using replication. It is only allowed for ADALOD LOAD if replication is already turned on for a database.

Note: The version of Event Replicator specified in an ADALOD LOAD utility job that specifies RPLLOAD=YES must match the version of Event Replicator used by the Event Replicator database. In addition, the Adabas version used by the Event Replicator database must be greater than or equal to the Adabas version used in an ADALOD LOAD utility job that specifies RPLLOAD=YES.

The RPLLOAD parameter indicates whether or not data, and possibly the FCB/FDT, loaded to the Adabas database via the ADALOD LOAD utility will be replicated to the Event Replicator Server. Valid values are "YES", "FILE", and "NO"; the default is "NO":

- When RPLLOAD=YES is specified, ADALOD LOAD replicates data to the Event Replicator Server database that it loads to the Adabas database.
- When RPLLOAD=FILE is specified, ADALOD LOAD replicates the FCB/FDT and data to the Event Replicator Server database. When the data is replicated to an Adabas destination, the file is first allocated on the target database. The DESTINATION parameters DREPLICATEUTI for the target destination and DAREPLICATEUTI for the target file must be set to YES to use this option.
- When RPLLOAD=NO is specified, ADALOD LOAD does not replicate its load data to the Event Replicator Server database.

Caution: If ADALOD LOAD ends abnormally (due to insufficient space, for example), updates made to the file before the abnormal ending cannot be backed out; there is no automated recovery for the updated data or for the replicated data. Software AG therefore recommends that you perform an ADASAV SAVE on the file before you run ADALOD LOAD.

RPLTARGETID: Replication Target ID

The RPLTARGETID parameter is an Event Replicator for Adabas parameter for an Adabas database that specifies the Event Replicator target ID used when the Adabas file data is replicated.

The RPLTARGETID parameter may not be specified in the same ADALOD LOAD as the REPLICATOR parameter. For more information about how this target ID is used in Adabas database processing during replication, read *Adabas Nucleus Replication Setup* in the *Event Replicator for Adabas Concepts Manual* documentation.

Note: Replication may not be turned on for an Adabas system file, a ciphered file, or a file that allows spanned data storage records (i.e. a file compressed with the ADACMP COMPRESS option SPAN).

RPLUPDATEONLY: Allow Only Event Replicator Processing Updates

The `RPLUPDATEONLY` parameter is an Event Replicator for Adabas parameter for an Adabas database that can be used in the ADALOD LOAD function to indicate whether an Adabas database file may be updated only by the Event Replicator Server as part of Adabas-to-Adabas replication or by other means as well. This parameter is optional. Valid values are "YES" or "NO". A value of "YES" indicates that the file can only be updated via Event Replicator processing; a value of NO indicates that the file can be updated by any normal means, including Event Replicator processing.

If the file is a new file, the default for this parameter is "NO".

However, if the file specified in the ADALOD LOAD function is an existing file, there is no default for this parameter. If no value is specified for the `RPLUPDATEONLY` parameter in the ADALOD LOAD function for an existing file, the value used previously for the file is used.

SKIPREC: Number of Records to Be Skipped

SKIPREC specifies the number of input records to be skipped before beginning load processing. The default is 0 (no records are skipped).

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the SKIPREC parameter cannot be specified in the ADALOD LOAD run.

SLOG: Load an Adabas SLOG File

The `SLOG` parameter is an Event Replicator for Adabas parameter for an Event Replicator Server database. It is a file type parameter. Use this parameter to load the SLOG system file into the Event Replicator database.

An SLOG file is an Adabas system file used only by the Event Replicator Server to store subscription logging data. The `SLOG` parameter may not be specified when loading a file on a non-Event Replicator database. The contents of DD/EBAND are ignored when loading an SLOG file.

SORTDEV: Sort Device Type

ADALOD uses the sort data set to sort descriptor values. The SORTDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEMPDEV: Temporary Storage Device Type

ADALOD uses the temp data set to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UISIZE: Upper Index Size

UISIZE specifies the number of blocks or cylinders to be assigned to the upper index. A block value must be followed by a "B" (for example, "5500B").

If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If the UISIZE parameter is omitted:

- ADALOD determines the space allocation for the upper index based on a sampling of records taken from the input data set. Since this calculation requires additional CPU time and I/O operations, Software AG recommends setting this parameter if the size is known so that no estimation is performed.
- and INDEXCOMPRESSION=YES is set, the index size estimation made by ADALOD does not consider the index compression as it has no knowledge of the rate of compression to be expected. ADALOD may thus allocate a larger index than necessary.

If a small number of records is being loaded and a larger number of records are to be added later, the UISIZE parameter should be set to increase the upper index to accommodate the total record amount. For more information, see the section *LOAD File Space Allocation* in the ***LOAD Data and Space Requirements*** section.

UQDE: Unique Descriptors

UQDE defines one or more descriptors as unique. Each descriptor specified must contain a different value in each input record. If a non-unique value is detected during ADALOD processing, ADALOD terminates with an error message.

If the unique descriptor (UQ) option was specified with the ADACMP utility, the UQDE parameter here is not required.

Adabas prevents a descriptor defined with the unique descriptor (UQ) option from being updated with an add or update command if the update would cause a duplicate value for the descriptor.

Note: For Adabas expanded files, ADALOD can only detect unique descriptor violations within the component file. If an identical value exists for a unique descriptor in one of the other component files, ADALOD cannot detect it. You must therefore ensure that unique descriptor values remain unique throughout an expanded file.

Note: If a *LOB file* is being loaded (read about the **FILE parameter**), the UQDE parameter cannot be specified in the ADALOD LOAD run.

USERISN: User ISN Assignment

USERISN=YES indicates that the USERISN option for the loaded file is to be in effect, and that the ISN for each new record is being supplied by the user in the input data. If USERISN=NO, Adabas assigns the ISN for each new record.

If USERISN is not specified, a default setting is assumed that depends on the input file itself. If the input file was created by ADACMP with the USERISN option or by ADAULD from a file having the USERISN option, the default for ADALOD operation is USERISN=YES; otherwise, the default is USERISN=NO. Specifying USERISN here overrides the existing default value.

Note: Adabas 5.2 files initially loaded with the USERISN option do not require USERISN=YES to again be specified when the files are reloaded; ADALOD assumes the default as described above. However, Adabas 5.1 files initially loaded with the USERISN option *must* have USERISN=YES specified whenever they are reloaded.

VERSION: Input Data Format

Originally, this parameter specified the Adabas version of the output (ADACMP) data sets to be loaded into Adabas.

Because ADALOD determines the version of the sequential input data set itself, this parameter is ignored. It is available only for compatibility with old ADALOD jobs.

Examples

Example 1:

```
ADALOD  LOAD FILE=6,MAXISN=20000,DSSIZE=20,ASSOPFAC=15,
ADALOD       DATAPFAC=15,TEMPSIZE=20,SORTSIZE=10
```

File 6 is to be loaded. The number of records initially permitted for the file is 20,000. 20 cylinders are to be allocated for Data Storage. The Associator and Data Storage block padding factors are both 15 percent. The temp and sort data sets are 20 and 10 cylinders, respectively.

Example 2:

```
ADALOD  LOAD FILE=7,MAXISN=350000,ASSOPFAC=5,MINISN=100001
ADALOD       DATAPFAC=15,DSSIZE=100,USERISN=YES
ADALOD       TEMPSIZE=200,SORTSIZE=100
```

File 7 is to be loaded. The number of records initially allocated for the file is 250,000, and the minimum is 100,001. The Associator padding factor is 5 percent. The Data Storage padding factor is 15 percent. 100 cylinders are to be allocated for Data Storage. ISNs are contained in the input. The temp and sort data sets are equal to 200 and 100 cylinders, respectively.

Example 3:

```
ADALOD  LOAD FILE=8,ADAMFILE,ADAMDE='AK'
ADALOD       ADAMPARM=4,ADAMOFLOW=5,UQDE='AK',MINISN=1
ADALOD       MAXISN=10000,DSSIZE=20,ASSOPFAC=5,DATAPFAC=5
ADALOD       TEMPSIZE=10,SORTSIZE=5
```

File 8 is to be loaded as an ADAM file. Field AK is the ADAM key. 4 bits are to be truncated from each value of AK before using the value to calculate the Data Storage RABN for the record. The size of the ADAM overflow area is 5 cylinders. The field AK is defined as a unique descriptor. The maximum number of records initially allocated for the file is 10,000. Twenty (20) cylinders are to be allocated to Data Storage, from which the five ADAM overflow cylinders are taken. The padding factor for both the Associator and Data Storage is five percent. The sizes of the Temp and Sort data sets are ten and five cylinders, respectively.

Example 4:

```
ADALOD  LOAD FILE=9,NAME=INVENTORY,MAXISN=5000
ADALOD       DSSIZE=2000B,DSRABN=30629,NISIZE=300B,UISIZE=50B
ADALOD       MAXDS=1000B,MAXNI=50B,MAXUI=1B
ADALOD       INDEXCOMPRESSION=YES
ADALOD       ASSOPFAC=20,DATAPFAC=10
ADALOD       TEMPSIZE=10,SORTSIZE=5,UQDE='U1,U2'
```

File 9 is to be loaded. The text name for the file is INVENTORY. The initial space allocation for the file is for 5,000 records. 2,000 blocks are to be allocated for Data Storage, beginning with RABN 30,629. 300 blocks are to be allocated for the normal index. 50 blocks are to be allocated to the upper index. The maximum allocations per secondary extent for Data Storage, normal index and upper index are 1000 blocks, 50 blocks, and 1 block respectively. The index is to be compressed. The padding factor for the Associator is 20 percent. The padding factor for Data Storage is 10 percent. The sizes of the temp and sort data sets are 10 and 5 cylinders respectively. Descriptors U1 and U2 are defined as unique descriptors.

Example 5:

```
ADALOD  LOAD FILE=2,SECURITY
ADALOD       DSSIZE=20B,MAXISN=2000,NISIZE=20B,UISIZE=5B
ADALOD       TEMPSIZE=10,SORTSIZE=5
```

File 2 is to be loaded as an Adabas security file. The DDEBAND contents are ignored. Space is allocated for Data Storage (20 blocks), for the address converter (2000 ISNs), the normal index (20 blocks), and the upper index (5 blocks). The temp size is 10 cylinders, and the sort area size is 5 cylinders.

LOAD Data and Space Requirements

The following general information describes data requirements for LOAD operation, and how ADALOD LOAD allocates space. For more information about space allocation, refer to the *Adabas DBA Reference* documentation.

Input Data for LOAD Operations

Compressed data records produced by the ADACMP or ADAULD utility may be used as input to ADALOD. If output from an ADAULD utility run made with the MODE=SHORT option is used as ADALOD input, any descriptor information will be removed from the FDT, and no index will exist for the file.

LOAD File Space Allocation

ADALOD allocates space for the normal index (NI), upper index (UI), address converter (AC), Data Storage, and the temp area for the file being loaded.

Index Space Allocation

If the NISIZE and/or the UISIZE parameters are supplied, allocation is made using the user-supplied values. If these parameters are not supplied, ADALOD allocates space for these indexes based on a sampling of the values present for each descriptor.

Descriptor values are sampled as follows:

1. ADALOD reads the compressed input, stores the records into Data Storage, extracts each value for each descriptor and writes these values to the temp data set. Each temp block contains values for one descriptor only. At the end of this processing phase, the following information is present:

   - number of values for each descriptor
   - number of bytes required for each descriptor
   - temp RABNs used for each descriptor

   For unique descriptors, the NI space requirement is equal to the temp size used. For non-unique descriptors, the number of duplicate values must be determined. Each duplicate value's space requirement must be estimated and then subtracted from the number of bytes required. The result is the NI size required for the duplicate descriptor.

   The number of duplicate values is determined by reading up to 16 temp blocks containing values for a single descriptor. These values are sorted to determine how many are duplicates. The resulting count of duplicate values is multiplied by the factor:

$$\frac{\text{total number of values}}{\text{number of values in the sample}}$$

The result is the estimated number of identical descriptor values present in the entire file for this descriptor. This space requirement is subtracted from the temp size estimate.

2. The upper index (UI) size is computed after all normal index (NI) and temp sizes are available.
3. The NI and UI sizes are each multiplied by the result of:

$$\frac{\text{(MAXISN - MINISN) + 1}}{\text{number of records being loaded}}$$

For example, if 10000 records require 10 blocks of UI space and 500 blocks of NI space with MINISN = 1 (the default), the specification of MAXISN = 60000 causes 60 UI blocks and 3000 NI blocks to be allocated:

$$10 \times \frac{60000}{10000} = 60 \text{ UI Blocks}$$

$$500 \times \frac{60000}{10000} = 3000 \text{ NI Blocks}$$

However, this calculation is not made if USERISN=YES is in effect.

By setting MAXISN appropriately, it is therefore possible to increase the size allocation for files in which a small number of records are being loaded and for which a much larger number of records are to be added subsequently.

If the NISIZE and UISIZE parameters have been specified, the space allocation is made using unassigned Associator RABNs. If the NIRABN and/or the UIRABN parameters are supplied, space allocation is made at the user-specified RABN.

Address Converter Space Allocation

The address converter allocation is based on the MAXISN and MINISN values for the file. ADALOD allocates the blocks needed to contain the number of bytes calculated by the formula:

```
RABNSIZE x((MAXISN - MINISN)+ 1)
```

If the ACRABN parameter has been specified, ADALOD allocates the address converter beginning with the user-specified block number; otherwise, it uses unassigned Associator RABNs.

Data Storage Space Allocation

Data Storage allocation is based upon the value specified with the DSSIZE parameter. If the DSRABN parameter has been specified, the allocation is made beginning with the user-specified block number; otherwise, unassigned Data Storage RABNs are used.

If there are different device types in the database, Data Storage allocation can be forced on a specified device type by specifying DSDEV. The MIXDSDEV parameter permits Data Storage allocation on different device types, assuming the device types can store records with the length specified by MAXRECL.

Temp Area Space Allocation

For each descriptor, ADALOD generates a list of the values and ISNs of the records containing the value, and writes this information to the Temp data set. The space required for descriptor information is equal to the sum of the space required for each descriptor. The space needed for each descriptor can be calculated using the following formula:

```
SP = N x  NPE x  NMU x  (L + 4)
```

where

SP	is the space required for the descriptor (in bytes).
N	is the number of records being loaded.
NPE	is the average number of occurrences, if the descriptor is contained in a periodic group. If not in a periodic group, NPE equals 1.
NMU	is the average number of occurrences, if the descriptor is a multiple-value field. If not a multiple-value field, NMU equals 1.
L	is the average length (after compression) of each value for the descriptor.

Example:

A file containing 20,000 records is being loaded. The file contains two descriptors (AA and CC). Descriptor AA has 1 value in each record and the average compressed value length is 3 bytes. Descriptor CC has an average of 10 values in each record and the average compressed value length is equal to 4 bytes.

Field Definitions:

```
01,AA,5,U,DE
01,CC,12,A,DE,MU
```

- Space requirement for AA.

```
SP = 20,000 •  1 •  (3 + 4)
SP = 140,000 bytes
```

- Space requirement for CC.

```
SP = 20,000 •  10 •  (4 + 4)
SP = 1,600,000 bytes
```

- Total space requirement = *1,740,000 bytes* .

The number of cylinders required may be calculated by dividing the number of blocks required by the number of blocks per cylinder.

For a model 3380 device type:

$$\text{Blocks required} = \frac{1{,}740{,}000}{7476 \text{ bytes per block}} = 232 +, \text{ or } 233 \text{ blocks}$$

$$\text{Cylinders required} = \frac{233 \text{ blocks}}{90 \text{ blocks per cylinder}} = 2 +, \text{ or } 3 \text{ cylinders}$$

Associator Updating by LOAD

ADALOD then sorts the descriptor values collected in the input phase and enters the sorted values into the normal index and upper index. If the allocated index space is not enough for the normal index or upper index, ADALOD allocates up to four additional extents.

Each additional extent allocated is equal to about 25 percent of the total current space allocated to the index. If insufficient space is available for the additional extent or if the maximum number of allocated extents has been reached, ADALOD terminates with an error message.

Loading Expanded Files

An expanded file is made up of a series of normal Adabas physical files. The number sequence of the files within the expanded file is arbitrary. The first file may be file 53; the second, file 127; the third, 13, and so on. ISNs assigned to each component file must be unique; no two files can contain the same ISN. The ISN range over all files must be in ascending order; however, there can be gaps in the sequence.

The total number of records in an expanded-file chain cannot exceed 4,294,967,294.

The sequence of physical component files that build an expanded logical file is defined by the ANCHOR parameter, which defines the first component file (anchor) in the sequence. The anchor file is loaded just as any other Adabas file; each additional component file must be loaded with the ANCHOR parameter referring to the anchor file. ADALOD inserts the new physical file into the existing expanded file chain according to the range of ISNs assigned to the added file. Each added component file must also specify the NOACEXTENSION parameter when being loaded to prevent Adabas from assigning new ISNs to a component file.

ADALOD processes only the anchor file and the single physical (component) files that compose an expanded file, and not the complete expanded file itself.

Loading Data into an Expanded File

To load data (for example, several million records) into different physical files, the input data must first be divided into several DDEBAND/EBAND input files. The DDEBAND/EBAND file data may be mapped into the component files using the SKIPREC and NUMREC parameters; however, one-to-one mapping without skipping or limits is recommended. This avoids the need to read records that will not be used later, and thus improves performance.

Examples:

The following examples, which show parts of one or more jobs for loading an expanded file, illustrate the mapping of DDEBAND/EBAND file data into component files:

```
//DDEBAND    DD DSN=LOAD.DATA.FILE1,...
//DDKARTE    DD *
ADALOD LOAD FILE=40,NAME='XXX_Part1'
ADALOD      MINISN=1,MAXISN=10000000,NOACEXTENSION
ADALOD      NUMREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .

//DDEBAND    DD DSN=LOAD.DATA.FILE1,...
//DDKARTE    DD *
ADALOD LOAD FILE=41,NAME='XXX_Part2',ANCHOR=40
ADALOD      MINISN=10000001,MAXISN=20000000,NOACEXTENSION
ADALOD      NUMREC=10000000,SKIPREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .

//DDEBAND    DD DSN=LOAD.DATA.FILE2,...
//DDKARTE    DD *
ADALOD LOAD FILE=35,NAME='XXX_Part2',ANCHOR=40
ADALOD      MINISN=20000001,MAXISN=30000000,NOACEXTENSION
```

```
ADALOD      NUMREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .
```

Loading Multiclient Files

Note: A multiclient file cannot be made part of an expanded file, and an expanded file cannot be converted to a multiclient file.

A multiclient file stores records for multiple users or groups of users. It divides the physical file into multiple logical files by attaching an owner ID to each record. Each user can access only the subset of records that is associated with the user's owner ID.

For any installed external security package such as RACF or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID. The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user.

The ADALOD LOAD function uses the LOWNERID and ETID parameters to support the migration of an application from a standard to a multiclient environment. The parameters work together to define owner IDs and determine whether a file is a multiclient file.

LOWNERID specifies the length of the internal owner ID values assigned to each record for multiclient files. In combination with the ETID parameter, the LOWNERID parameter can be used to reload a standard file as a multiclient file, change the length of the owner ID for the file, or remove the owner ID from the records of a file.

If the LOWNERID parameter is not specified, the length of the owner ID for the input file (if any) remains the same.

ETID assigns a new owner ID to all records being loaded into a multiclient file, and must be specified if the input file contains no owner IDs; that is, the input file was not unloaded from a multiclient source file.

Examples of Loading/Updating Multiclient Files

```
ADALOD LOAD FILE=20,LOWNERID=2,NUMREC=0
```

Creates file 20 as a multiclient file. The length of the internal owner ID is two bytes, but no actual owner ID (ETID) is specified. No records are actually loaded in the file (NUMREC=0).

```
ADALOD LOAD FILE=20,LOWNERID=2,ETID=USER1
```

Creates file 20 as a multiclient file, load all supplied records, and assign them to user USER1. The length of the internal owner ID is two bytes.

```
ADALOD UPDATE FILE=20,ETID=USER2
```

Performs a mass update to add records to file 20, a multiclient file. Load all the new records and assign them to USER2.

98 UPDATE: Add/Delete Records

- Essential Parameters 543
- Optional Parameters and Subparameters 544
- Examples 549
- Formats for Specifying ISNs 550
- UPDATE Data and Space Requirements 552
- Mass Updates of Expanded Files 554

Caution: If ADALOD UPDATE ends abnormally (due to insufficient space, for example), updates made to the file before the abnormal ending cannot be backed out. Software AG therefore recommends that you perform ADASAV SAVE on the file before you run ADALOD UPDATE.

The UPDATE function adds or deletes a large number of records (ISNs) to or from an existing file. A single UPDATE operation can both add and delete ISNs.

Records to be added must be in compressed (ADACMP or ADAULD output) form and be in the DDEBAND/EBAND input data set.

ISNs to be deleted must be specified by either or both of the DDISN and DELISN parameters.

Notes:

1. The UPDATE function cannot be used with an Adabas system file if the Adabas nucleus is active, and cannot be used to change the checkpoint or security files.
2. A multiclient file cannot be made part of an expanded file, and an expanded file cannot be converted to a multiclient file.

```
ADALOD UPDATE   FILE = file-number
                SORTSIZE = size
                TEMPSIZE = size
                [DDISN]
                [DELISN =  isn-list ]
                [DSREUSE = { YES | NO } ]
                [ETID =  multiclient-file-owner-id ]
                [ISNREUSE = { YES | NO } ]
                [LIP = { isn-pool-size |  2000 } ]
                [LWP = { work-pool-size |  1048576 } ]
                [MAXISN =  number
                    [ACRABN = starting-rabn ]
                    [ASSOVOLUME = 'Associator-extent-volume' ] ]
                [MAXISN2 = number
                    [AC2RABN = starting-rabn]
                    [ASSOVOLUME = 'Associator-extent-volume' ] ]
                [NOUSERABEND]
                [NUMREC = number]
                [PASSWORD = 'password ' ]
                [RESTART]
                [RPLLOAD = { YES | NO } ]
                [SKIPREC = {number |  0 } ]
                [SORTDEV = {device-type  | ADARUN-device  } ]
                [TEMPDEV = {device-type  | ADARUN-device  } ]
                [TEST]
                [USERISN = { YES | NO } ]
```

Essential Parameters

FILE: File Number

FILE specifies the number of the file to be updated. If a component file of an Adabas expanded file is specified, only that component file is updated; the other component files must be updated in separate UPDATE operations.

SORTSIZE: Sort Size

SORTSIZE is the number of blocks or cylinders available for the sort data set.

TEMPSIZE: Temporary Storage Size

TEMPSIZE is the number of blocks or cylinders available for the temp data set.

Optional Parameters and Subparameters

ACRABN: Starting RABN for Address Converter

ACRABN causes additional space allocation for the address converter to begin at the specified RABN. ACRABN is effective only if MAXISN specifies an increase for the file's address converter.

AC2RABN: Starting RABN for Secondary Address Converter

AC2RABN causes additional space allocation for the secondary address converter to begin at the specified RABN. AC2RABN is effective only if MAXISN2 specifies an increase for the file's secondary address converter. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored. There is no default value.

ASSOVOLUME: Associator Extent Volume

Note: The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME is effective only if MAXISN or MAXISN2 specify an increase for the file's address converter.

ASSOVOLUME specifies the volume on which the file's address converter extents is to be allocated. If the requested number of blocks cannot be found on the specified volume, ADALOD retries the allocation while disregarding the ASSOVOLUME parameter.

If ACRABN or AC2RABN is specified, ADALOD ignores the ASSOVOLUME value when allocating the address converter extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADALOD's default allocation rules.

DDISN: Read ISNs to be Deleted from Sequential Data Set

If DDISN is specified, ISNs to be deleted are read from the DDISN/ISN sequential data set. If both the DDISN and DELISN parameters are specified, the ISNs from the two lists are merged. The DDISN/ISN data set must have variable or variable blocked records. See the section **Formats for Specifying ISNs** for more information.

When the UPDATE function is executed, all ISNs are first read and stored in the ISN pool in the order they occur. The size of the ISN pool (specified by LIP) must be large enough to store all data read from DDISN/ISN. The records are then sorted in ascending order. Overlapping ranges and duplicate ISNs are not allowed. ISNs not found during processing are ignored.

When deleting ISNs from an Adabas expanded file, you can specify the complete ISN list for all component files; the UPDATE function automatically selects only the ISNs that are appropriate for the component file being processed.

Note: This parameter cannot be specified in an ADALOD UPDATE operation on a *LOB file*.

DELISN: ISNs to be Deleted

DELISN specifies a list of the ISNs of records to be deleted. If both DDISN and DELISN are specified, the ISNs from the two lists are merged. A range list may be specified as:

```
DELISN=10-80,90,100-110 ↩
```

Overlapping ranges and duplicate ISNs are not allowed. You can specify, at most, 32 single ISNs or ISN ranges. When deleting ISNs from an Adabas expanded file, you can specify the complete list for all component files. The UPDATE function selects the appropriate ISNs from the list and deletes them from the component file.

Note: This parameter cannot be specified in an ADALOD UPDATE operation on a *LOB file*.

DSREUSE: Data Storage Reusage

DSREUSE indicates whether or not Data Storage space that becomes available as a result of a record deletion is to be reused.

This parameter is in effect for the execution of the UPDATE function only. The permanent setting of DSREUSE is not changed. That permanent setting is the default if this value is not specified.

ETID: Multiclient File Owner ID

The ETID parameter assigns a new owner ID to all records being added to an existing multiclient file. The owner ID is automatically adjusted to the length for owner IDs specified by LOWNERID when the multiclient file was last loaded. If no ETID is specified, all loaded records keep their owner IDs specified on the input source.

The ETID parameter must be specified if the existing file is multiclient and the input file was not unloaded from a multiclient file. ETID must not be specified if the existing file is a non-multiclient file.

Note: If the ETID parameter is used, the ADALOD utility requires an active nucleus. The nucleus will translate the ETID value into the internal owner ID value.

ISNREUSE: ISN Reusage

ISNREUSE indicates whether the ISN for a deleted record can be reassigned to a new record.

This ISNREUSE setting is in effect only during execution of the UPDATE function. The permanent ISNREUSE setting is unchanged. The permanent setting is the default if this value is not specified.

LIP: ISN Work Pool Size

LIP specifies the size of the work pool for containing ISNs to be deleted. Four bytes per ISN and eight bytes per ISN range are required in this pool. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 2000 bytes.

LWP: Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADALOD run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the Sort data set, a smaller LWP value should be specified when updating relatively small files.

The minimum work pool size depends on the sort data set's device type:

Sort Device	Minimum LWP	Minimum LWP
	Bytes	Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

MAXISN: Highest ISN to be Allocated to the File

The MAXISN parameter may be used to specify a new ISN setting for the file. This parameter should be used if the current record count plus the number of ISNs (records) to be added exceeds the current MAXISN setting. The specified larger value determines the additional space required for the address converter, and causes ADALOD to allocate a new extent. A smaller MAXISN value causes no change in the address converter space.

Note: The MAXISN setting for a file cannot be increased if the file was last loaded with NOACEXTENSION active.

The MAXISN setting should be increased by an amount suitable for all planned expansion; this avoids using up the address converter extent too quickly, and alleviates the need to either unload and reload the file or run the ADAORD REORFASSO utility because the maximum number of address converter extents has been allocated.

With the optional ACRABN parameter, the beginning of the new address converter extent can be set to a specific RABN number. See the ACRABN parameter description for more information.

If the MAXISN parameter is omitted, ADALOD allocates new address converter extents only if the old MAXISN value is exceeded.

MAXISN2: Highest ISN to be Allocated in the Secondary Address Converter for the File

The MAXISN2 parameter is optional, regardless of whether or not spanned records exist in the ADALOD input file or not. Use this parameter to specify the desired size of the secondary address converter (AC2) in ISNs. The secondary address converter is used to map secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADALOD determines the number of ISN mappings for which to allow space in the secondary AC using the following method:

1. If a secondary AC is not yet allocated, one secondary AC block is allocated.
2. If the current MAXISN2 setting is greater than or equal to four times the MAXISN value (the current maximum primary ISNs expected), the same algorithm is used to determine the number of secondary ISNs as is used to allocate additional primary ISNs.
3. If none of the above conditions is met, then secondary AC space is allocated as the smaller of the following two calculations:
   - The product of 10 times the old MAXISN2 setting (10 x oldMAXISN2)
   - The sum of the old MAXISN2 setting and the MAXISN setting (oldMAXISN2 + MAXISN)

There is no default value.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Limit Number of Records to Be Added

NUMREC limits the number of records to be added. If NUMREC is specified, ADALOD processing terminates after adding the number of records specified (unless an end-of-file condition on the input data set has already caused ADALOD termination). If this parameter is omitted, *all* input records are added.

If the input data set contains more records than specified by NUMREC, ADALOD adds the number of records specified by NUMREC and then terminates with condition code 4.

Note: This parameter cannot be specified in an ADALOD UPDATE operation on a *LOB file*.

PASSWORD: File Password

If the file to be updated is password-protected, the parameter must be used to provide a valid password. There is no default for PASSWORD.

RESTART: Restart Interrupted ADALOD Execution

RESTART forces an interrupted ADALOD run to be restarted, beginning with the last *restart point* reached before the interruption. The *restart point* is the latest point of execution that can be restored from the Temp data set.

If ADALOD is interrupted by a defined error condition, ADALOD issues a message indicating whether or not a restart is possible.

When restarting the ADALOD operation, the following parameters may be changed:

- TEMPSIZE can be increased to make the temp data set larger. Note, however, that the temp data set contents *must not be changed* because it contains information necessary for the restart operation;
- The SORTSIZE and SORTDEV parameters and the sort data set can be changed.

No other parameters can be changed. The DDEBAND/EBAND, DDFILEA/FILEA and DDISN/ISN data sets must remain the same.

RPLLOAD: Replicate Update Data

The ADALOD UPDATE RPLLOAD parameter is an Event Replicator for Adabas parameter that can be specified for the Adabas database files when using replication. It is only allowed for ADALOD UPDATE if replication is already turned on for a database.

Note: The version of Event Replicator specified in an ADALOD UPDATE utility job that specifies RPLLOAD=YES must match the version of Event Replicator used by the Event Replicator database. In addition, the Adabas version used by the Event Replicator database must be greater than or equal to the Adabas version used in an ADALOD UPDATE utility job that specifies RPLLOAD=YES.

The `RPLLOAD` parameter indicates whether or not inserts and deletes to the Adabas database via the ADALOD UPDATE utility will be replicated to the Event Replicator Server. Valid values are "YES" and "NO"; the default is "NO".

When RPLLOAD=YES is specified, ADALOD UPDATE inserts and deletes to the Adabas database will be replicated to the Event Replicator database. When RPLLOAD=NO is specified, ADALOD UPDATE inserts and deletes are *not* replicated.

Caution: If ADALOD UPDATE ends abnormally (due to insufficient space, for example), updates made to the file before the abnormal ending cannot be backed out; there is no automated recovery for the updated data or for the replicated data. Software AG therefore recommends that you perform an ADASAV SAVE on the file before you run ADALOD UPDATE.

SAVEDREC: Save Deleted Records on a Sequential File

SAVEDREC indicates that deleted records are to be written to a sequential data set. The format of the data set is identical to that created by the ADAULD utility with the MODE=SHORT option.

Note: This parameter cannot be specified in an ADALOD UPDATE operation on a *LOB file*.

SKIPREC: Number of Records to Be Skipped

SKIPREC is the number of input records to be skipped before beginning to process updates. The default is 0 (no records are skipped).

Note: This parameter cannot be specified in an ADALOD UPDATE operation on a *LOB file*.

SORTDEV: Sort Device Type

ADALOD uses the sort data set to sort descriptor values. The SORTDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEMPDEV: Temporary Storage Device Type

ADALOD uses the temp data set to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this data set. This parameter is required only if the device type to be used is different from the standard device type assigned to Temp by the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

USERISN: User ISN Assignment

USERISN=YES indicates that the USERISN option for the file is to be in effect, and that the ISN for each new record is being supplied by the user in the input data. If USERISN=NO, Adabas assigns the ISN for each new record.

The specified USERISN setting is effective only while the UPDATE function is executing. The permanent USERISN setting is not changed, and is the default if this parameter is not specified.

When performing an ADALOD UPDATE function on a file with a hyperdescriptor for which the hyperexit changed the ISNs of descriptor values, USERISN=YES is no longer required for the add/load operation.

When adding records *from* a non-USERISN=YES file, the ADALOD parameter USERISN=NO must be specified and the file to be updated must have the USERISN option. This feature is useful for Adabas Text Retrieval (TRS).

Examples

Example 1:

```
ADALOD  UPDATE FILE=6,MAXISN=18000
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

Records are to be added to file 6. The MAXISN for the file is to be increased to 18,000.

Example 2:

```
ADALOD  UPDATE FILE=7,TEMPSIZE=10,
ADALOD         ETID=USER3,SORTSIZE=5
```

Records with user's owner ID of USER3 are to be added to multiclient file 7.

Example 3:

```
ADALOD  UPDATE FILE=8,DELISN=1000-1999,5000-5999
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

The records with ISNs 1,000 to 1,999 and 5,000 to 5,999 are to be deleted from file 8. If an input data set is provided, records are to be added.

Example 4:

```
ADALOD  UPDATE FILE=6
ADALOD         DDISN,SAVEDREC
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

Records are to be deleted from file 6. The ISNs of the records to be deleted are contained in an input data set. The deleted records are to be saved on an output data set.

Example 5:

```
ADALOD  UPDATE FILE=6,DDISN,LIP=20K,SKIPREC=500
ADALOD         TEMPSIZE=5,SORTSIZE=10
```

Records are to be added and deleted from file 6. The ISNs which identify the records to be deleted are contained in an input data set (DDISN). The size of the ISN pool is set to 20K. The first 500 records on the input data set are to be skipped.

Formats for Specifying ISNs

There are two formats for specifying ISNs in the DDISN or ISN data set. The first format can be used in all cases where only 31-bit ISNs are specified. A record can contain a mix of single ISNs and ranges of ISNs.

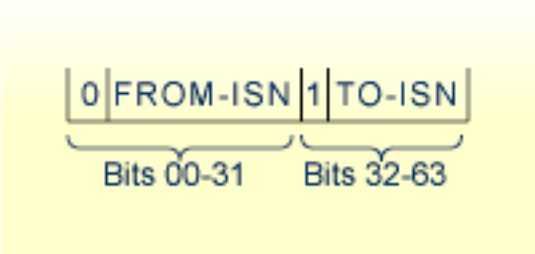
The second format supports 32-bit ISNs and can only be used with Adabas version 6 and above. Each record can specify *either* single ISNs (indicated by X'00000000' in the first fullword) *or* ranges of ISNs (indicated by X'FFFFFFFF' in the first fullword).

If the first fullword in a record contains a value other than X'00000000' or X'FFFFFFFF', it is assumed to be the 31-bit format. The DDISN/ISN data set can contain records in both formats.

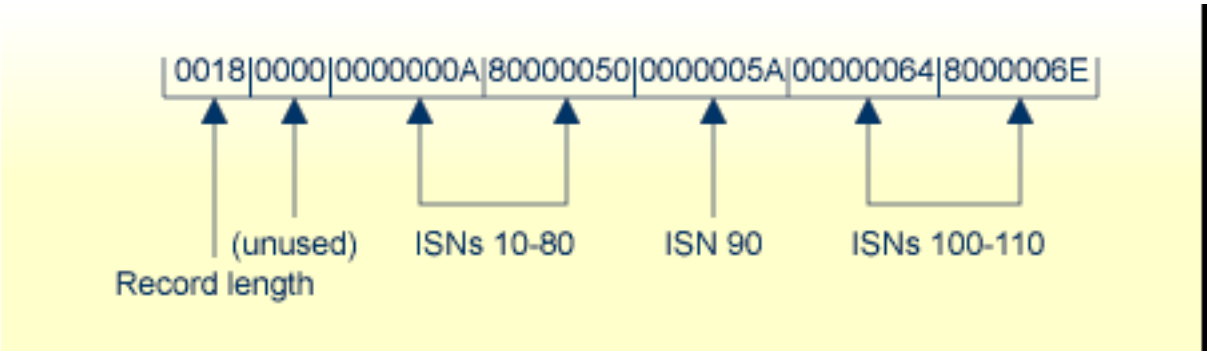
Format 1: 31-Bit Format

A single ISN requires 4 bytes. Set the high-order bit to 0 and specify the ISN in bits 01-31:

A range of ISNs requires 8 bytes. In the first four bytes, specify the first ISN in the range as a single ISN; in the next four bytes, set the high-order bit to 1 and specify the last ISN:

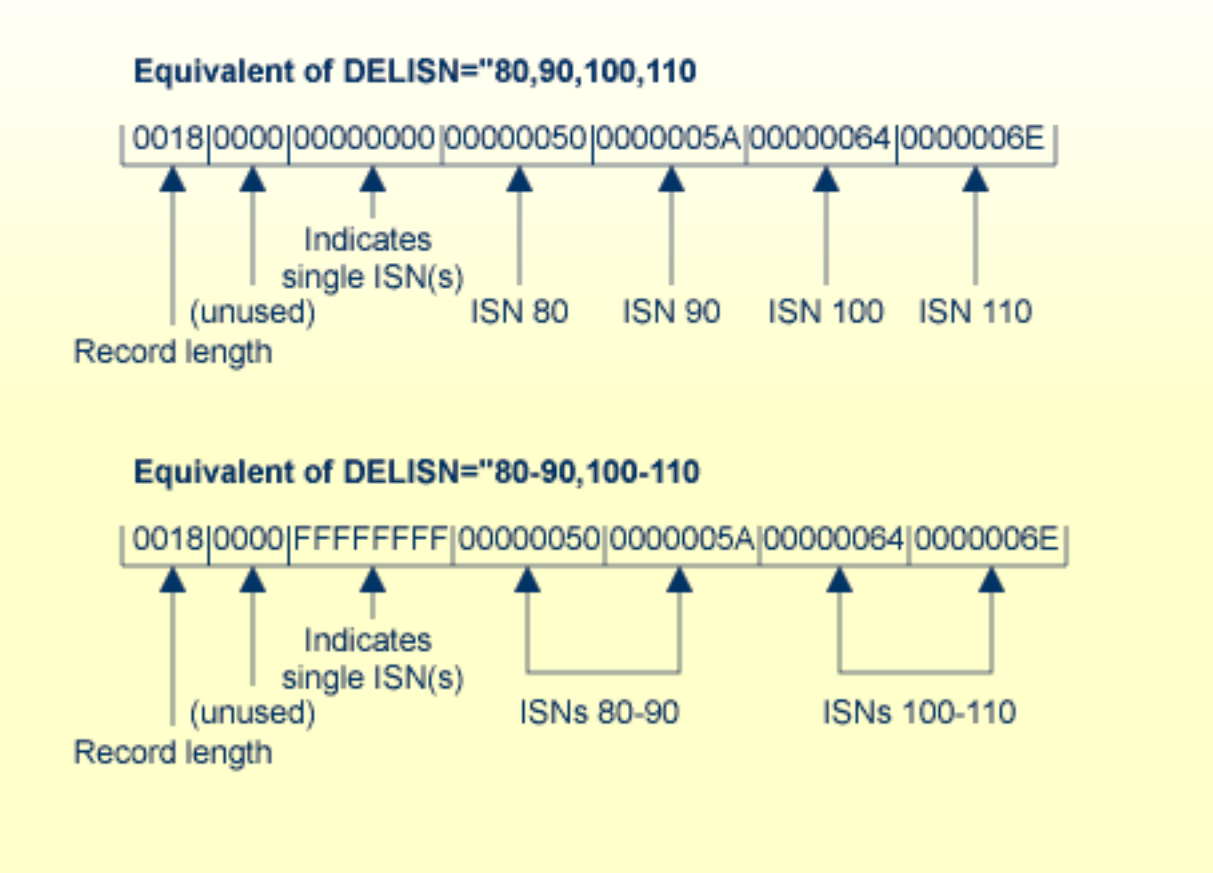


The following example shows a variable-length record containing the equivalent of DELISN=10-80,90,100-110:



Format 2: 32-Bit Format

In the 32-bit format, the first fullword in each record indicates whether the record contains single ISNs or ranges of ISNs. To indicate single ISNs, put zero in the first fullword (X'00000000'); to indicate ranges of ISNs, put -1 (X'FFFFFFFF'). In the following example, the first record contains single ISNs; the second record contains ranges. The two records are identical except for the indicator in the first fullword.



UPDATE Data and Space Requirements

The following general information describes data requirements for UPDATE operation, and how ADALOD UPDATE allocates space. For more information about space allocation, refer to the *Adabas DBA Reference* documentation.

Input Data for UPDATE Operations

Records to be added must be in the form of compressed data records produced by the ADACMP or ADAULD utility. The field definitions used for the ADACMP run must agree with the

definitions for the file to which the records will be added as contained in the field definition table (FDT).

Note: Records being added to a ciphered file must already be encrypted using the same cipher code as was used for the records already in the file.

The ISNs of records to be deleted may be provided with the DELISN parameter and/or in an input data set. If provided in an input data set, each ISN must be provided as a 4-byte binary number. The data set must have the record format VARIABLE BLOCKED. If desired, all ISNs to be added to or deleted from an Adabas expanded file can be specified; the UPDATE function selects the appropriate ISNs for the component file being processed.

UPDATE Space Allocation

If records are to be added and a larger MAXISN value has been specified, an additional address converter extent will be allocated by ADALOD. The size of the new extent is based on the difference between the new MAXISN and the previous MAXISN setting. If either insufficient space is available for the new extent or the maximum number of extents has already been allocated, processing ends with an error message.

If an additional Data Storage extent is required, ADALOD allocates an additional extent equal to approximately 25 percent of the total size of the Data Storage extents currently allocated to the file. As for the address converter, processing ends with an error message if either sufficient space is not available for the added extent or the maximum number of extents has already been allocated.

Generating UPDATE Descriptor Information

When adding records, ADALOD UPDATE generates a list of all descriptor values and the corresponding ISNs of the new records, and writes this information to the temp data set.

Associator Updating with UPDATE

Before processing the input, ADALOD UPDATE copies the file's existing normal index to the temp data set, but removes the descriptor values of any ISNs to be deleted.

ADALOD sorts the information written to temp during the input phase and merges the sorted values with the current normal index. The normal index is reordered during this process, and the Associator block padding factor is reestablished for each block. A new upper index is then created.

Empty space in partially filled blocks resulting from descriptor updating is reused. This can increase the number of empty blocks at the end of the index. Although one or more normal index and/or upper index extents may become empty as the result of the reorder process, ADALOD does not condense, delete, or change the size of these extents.

If new free space is needed for the normal index or upper index, ADALOD allocates an additional extent (or extents). Each additional extent allocated is equal to approximately 25 percent of the total current space allocated to the index. If insufficient space is available for the additional extent or if the maximum number of extents has already been allocated, ADALOD terminates with an error message.

Mass Updates of Expanded Files

Using ADALOD UPDATE for a mass update to an expanded file, records must be added to or deleted from each component file individually. However, each component file can be processed using the same ADALOD commands.

When deleting a record with DELISN or DDISN, the complete list of ISNs to be deleted from all component files can be supplied. ADALOD automatically selects only the ISN values from the specified range that is appropriate for the component file currently being processed.

The same is true when adding new records with USERISN=YES.

When new expanded file records are being added with USERISN=NO but no free ISN is found, the loader cannot allocate a new address converter extent since the ISN range cannot be increased (NOACEXTENSION is active for all component files). Instead, ADALOD creates the index as though end-of-file had been reached. The remaining records not loaded may be added later to another component file using the SKIPREC parameter.

ADALOD does not check for unique descriptor values across component file boundaries.

Example:

The following is an example for performing a mass update to an expanded file (only the relevant parts of the complete jobs are shown):

```
              .
              .
//DDEBAND    DD DSN=MOREDATA.LOAD.PART1-2,...
//DDKARTE    DD *
ADALOD UPDATE FILE=40,USERISN=YES
ADALOD         DELISN=9000001-9500000,12000001-14000000
ADALOD         SORTSIZE=...,TEMPSIZE=...
              .
              .
//DDEBAND    DD DSN=MOREDATA.LOAD.PART1-2,...
//DDKARTE    DD *
ADALOD LOAD    FILE=41,USERISN=YES
ADALOD         DELISN=9000001-9500000,12000001-14000000
ADALOD         SORTSIZE=...,TEMPSIZE=...
              .
              -
```

99 Loader Storage Requirements and Use

Static Storage

Static	Type*	Size
Modules ADARUN, ADALOD	A	approximately 180 kilobytes

Dynamic Storage

Dynamic	Type*	Size
Sort work pool	A	LWP
General work pool	A	6 • (Associator block size)
I/O buffer for Associator	A	Associator block size
ISN pool	A	LIP
I/O buffer for Data Storage	A	Data Storage block size
AC bitmap	A	4K bytes
I/O buffer for temp	A	temp block size
DVT splitting	A	temp block size • number of descriptors
Internal descriptor table	A	number of descriptors • 74
I/O buffer for DDEBAND/EBAND	O	DDEBAND/EBAND block size
I/O buffer for DDOLD/OLD	O	DDOLD/OLD block size
I/O buffer for DDISN/ISN data	O	DDISN/ISN block size
I/O buffer if records must be written to temp overflow	O	DDFILEA/FILEA block size

** Type A is always used; type O is used only if needed.*

100 TEMP Data Set Space Usage

- Sequential TEMP Data Set 680

ADALOD uses the TEMP data set to store the following information:

- restart information;
- Data Storage RABN/ISN for each record to be deleted (UPDATE only);
- contents of the normal index at the start of the operation (UPDATE only);
- descriptor values obtained from the input data set;
- ADAM overflow area (ADAM files only).

Sequential TEMP Data Set

If the TEMP data set is filled while collecting descriptor values from the input data set, ADALOD temporarily writes the remaining descriptors to the sequential temp file DD/FILEA (if specified in the JCL). The descriptors are later read back in when the new index is built.

If actually called, DD/FILEA makes ADALOD operation considerably slower than specifying a TEMP data set that is large enough to hold all descriptor values. The DD/FILEA TEMP data set should normally be used only as a safety net to ensure adequate space for all descriptors during ADALOD operation. Specifying the DD/FILEA TEMP file therefore avoids an ADALOD abend caused by a temp area overrun.

Notes:

1. ADALOD writes only descriptor values from the DD/EBAND input file to DD/FILEA.
2. The normal TEMP data set must be large enough to hold all values for each single descriptor.
3. The estimated number of TEMP blocks for ADALOD UPDATE may need to be adjusted upwards if forward index compression is in effect for the file since the NI is written to the TEMP data set without forward compression.

If you are running ADALOD UPDATE only to delete ISNs, the size of the TEMP data set must be calculated as the sum of the calculations in the following three steps:

1. Part 1 of TEMP data set contains:
   - RABN 1 : information for restart
   - RABN 2 : FCB of the file
   - RABN 3 - n : DETAB (count of DEs * 116). For example, one element is 116 bytes. The count of DEs is the number of DEs in the FDT.
2. Part 2 of the TEMP data set contains the list of ISNs to be deleted, plus the belonging DS-RABNs, 8 bytes per element. Calculate:

```
(8 * to-be-deleted-ISNs) / (TEMPBLKSIZE - 16) = needed-blocks
```

3. Part 3 contains the complete NI:

```
count NI-BLKS * ASSOBLKSIZE) / (TEMPBLKSIZE - 16) = needed blocks
```

101 ADALOD Space/Statistics Report

During LOAD or UPDATE operation, ADALOD prints a report on the message output data set (DDDRUCK for z/OS and z/VM systems, SYS009 for VSE systems, or SYSOUT for BS2000). The report shows the following information:

- ADALOD function executed (LOAD or UPDATE), and the database/file affected;
- Estimated NI/UI sizes (shown for the LOAD function only if the NI/UISIZE parameters were not specified);
- Available and used file space, by Adabas component (shown for the LOAD function only);
- Current RABNs assigned for the file (shown for the LOAD function only);
- For spanned record files, the number INPUT RECORDS PROCESSED includes all physical record segments;
- File processing statistics (records processed and system storage used).

Example of the ADALOD LOAD report:

```
  PARAMETERS:ADALOD  LOAD FILE...
        .
        .

  FUNCTION TO BE EXECUTED:

LOAD FILE NUMBER    7  (MYOWNFILE)
INTO DATABASE    0013  (MYBESTDB)

AVAILABLE SPACE:

(LOAD function only)

        I FILE I  DEV   I  NUMBER OF  I      FROM      TO    I
        I LAY- I  TYPE  I   BLOCKS    I      RABN      RABN  I
```

```
        I OUT  I        I             I                       I
        I------I--------I-------------I-----------------------I
        I ASSO I  3380  I      2695   I       137      2831  I
        I DATA I  3380  I      1339   I         3      1341  I
        ------------------------------------------------------


ESTIMATED NORMAL INDEX SIZE = 37 BLOCKS
ESTIMATED UPPER INDEX SIZE = 8 BLOCKS

TOP ISN = 773, MAX ISN EXPECTED = 1335

I FILE  I  DEV   I  LIST  I   ALLOC    I    FROM      TO  I   UNUSED  I
I LAY-  I  TYPE  I  TYPE  I   SPACE    I    RABN    RABN  I   SPACE   I
I OUT   I        I        I  (BLOCKS)  I                  I  (BLOCKS) I
-------------------------------------------------------------------------
I ASSO  I  3380  I   AC   I         2  I      137     138 I          0 I
I ASSO  I  3380  I   UI   I         8  I      139     146 I          0 I
I ASSO  I  3380  I   NI   I        37  I      147     183 I         15 I
I DATA  I  3380  I   DS   I        60  I        3      62 I         48 I
-------------------------------------------------------------------------

   PROCESSING STATISTICS


            773 INPUT RECORDS PROCESSED
             14 BLOCKS USED ON TEMP-DATASET (0%)
              0 BLOCKS USED ON SORT PART 1  (0%)
              0 BLOCKS USED ON SORT PART 2  (0%)
          51824 BYTES OF STORAGE USED TO STORE RECORDS
```

102 JCL/JCS Requirements and Examples

- Collation with User Exit 564
- BS2000 564
- z/OS 568
- z/VM 570
- VSE 572

This section describes the job control information required to run ADALOD with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Note: When running with the optional Recovery Aid (RLOG), all temporary data sets must also be cataloged in the job control.

Collation with User Exit

If a collation user exit is to be used during ADALOD execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

```
ADARUN CDXnn= exit-name
```

where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference* documentation for more information.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/ tape	Stores descriptor values if the temp data set is too small

Data Set	Link Name	Storage	More Information
Sort area	DDSORTR1	disk	With large files, split the sort area across two volumes [1]
Sort area	DDSORTR2	disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
Compressed data	DDEBAND	disk/ tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/ tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/ tape	Deleted records, if any [3]
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADALOD parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		ADALOD report, see also *Messages and Codes*
ADALOD messages	SYSLST/ DDDRUCK		*Messages and Codes*

Notes:

1. Performance can be improved when sorting large files if the sort data set either occupies two volumes, or if two sort data sets are specified. Both data sets must be on the same device type (SORTDEV parameter), and each must be exactly half the size specified by the SORTSIZE parameter.
2. Four bytes per ISN, REC-FORM=VB, BUFF-LEN as in sequential file description, REC-SIZE maximum equals BUFF-LEN - 4. (In ISP format, REC-FORM is RECFM; BUFF-LEN is BLKSIZE; and REC-SIZE is LRECL.)
3. REC-FORM=VB, BUFF-LEN as in sequential file description, REC-SIZE maximum equals BUFF-LEN - 4. (In ISP format, REC-FORM is RECFM; BUFF-LEN is BLKSIZE; and REC-SIZE is LRECL.)

ADALOD JCL Example (BS2000)

Load File

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/ASS-SYSLST L.LOD.LOAD
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
```

```
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD LOAD FILE=1
ADALOD NAME= TESTFILE-1
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=50
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADALOD LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/SYSFILE SYSLST=L.LOD.LOAD
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/FILE CMP.AUS,LINK=DDEBAND

/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD LOAD FILE=1
ADALOD NAME= TESTFILE-1
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=50
/LOGOFF NOSPOOL
```

Update

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/DELETE-FILE LOD.ISN
/SET-JOB-STEP
/CREATE-FILE LOD.ISN,PUB(SPACE=(48,48))
/SET-JOB-STEP
/DELETE-FILE LOD.OLD
/SET-JOB-STEP
/CREATE-FILE LOD.OLD,PUB(SPACE=(480,48))
```

```
/SET-JOB-STEP
/ASS-SYSLST L.LOD.LOAD
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/SET-FILE-LINK DDEBAND,CMP.AUS
/SET-FILE-LINK DDISN,LOD.ISN
/SET-FILE-LINK DDOLD,LOD.OLD
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD UPDATE FILE=1,DDISN,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=50
ADALOD DELISN=100 199,230,301 399
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```

/.ADALOD LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A L O D MASS UPDATE
/REMARK *
/SYSFILE SYSLST=L.LOD.UPDA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/FILE CMP.AUS,LINK=DDEBAND
/FILE LOD.ISN,LINK=DDISN ,SPACE=(48,48)
/FILE LOD.OLD,LINK=DDOLD ,SPACE=(480,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD UPDATE FILE=1,DDISN,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=50
ADALOD DELISN=100 199,230,301 399
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	Required only if Adabas nucleus is not active
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/ tape	Stores descriptor values if the temp data set is too small
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, split the sort area across two volumes [1]
Recovery log (RLOG)	DDRLOGR1	disk	Required for the recovery log option
Compressed data	DDEBAND	disk/ tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/ tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/ tape	Deleted records, if any [3]
ADARUN parameters	DDCARD	reader	*Operations*
ADALOD parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	ADALOD report, see also *Messages and Codes*
ADALOD messages	DDDRUCK	printer	*Messages and Codes*

Notes:

1. Performance can be improved when sorting large files if the sort data set either occupies two volumes, or if two sort data sets are specified. When using two volumes, each volume must be exactly half the size specified by the SORTSIZE parameter. If two data sets are used, both must be on the same device type (SORTDEV parameter).
2. Four bytes per ISN, RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.
3. RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.

ADALOD JCL Examples (z/OS)

Refer also to ADALODE, ADALODA, ADALODM, and ADALODV in the JOBS data set for additional ADALOD examples on loading an ADAM file or the Adabas demo files.

Load File

Refer to ADALOD in the JOBS data set for this example.

```
//ADALOD    JOB
//*
//*    ADALOD: LOAD FILE
//*
//LOD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <=== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <=== SORT
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDEBAND <=== INPUT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADALOD LOAD FILE=1
ADALOD NAME='TESTFILE-1'
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=100
/*
```

Update

Refer to ADALODMU in the JOBS data set for this example.

```
//ADALODMU  JOB
//*
//*      ADALOD: MASS UPDATE
//*
//LOD       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD              <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1         <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1         <=== DATA
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1         <=== TEMP
```

```
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1         <=== SORT
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDEBAND        <=== INPUT
//DDISN     DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDISN          <=== ISNS TO DEL
//DDOLD     DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.DDOLD, <=== DEL REC
//            SPACE=(TRK,(100,20),RLSE),UNIT=DISK,VOL=SER=VOLvvv
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADALOD UPDATE FILE=1,LWP=400K,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=100
ADALOD DELISN=100-199,230,301-399
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDATARn	disk	
Work	DDWORKR1 DDWORKR4	disk	
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/ tape	Stores descriptor values if temp data set is too small.
Sort area	DDSORTR1	disk	With large files, split sort area across two volumes [1]
Sort area	DDSORTR2	disk	
Recovery log (RLOG)	DRLOGR1	disk	Required for the recovery log option
Compressed data	DDEBAND	disk/ tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/ tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/ tape	Deleted records, if any [3]
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADALOD parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	ADALOD report, see also *Messages and Codes*
ADALOD messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

Notes:

1. Performance can be improved when sorting large files if the sort data set either occupies two volumes, or if two sort data sets are specified. Both data sets must be on the same device type (SORTDEV parameter), and each must be exactly half the size specified by the SORTSIZE parameter.
2. Four bytes per ISN, RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.
3. RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.

ADALOD JCL Examples (z/VM)

Load File

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDEBAND,DSN=FILE001.LODD001,MODE=A
DATADEF DDPRINT,DSN=ADALOD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADALOD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNLOD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=FILE001.LODC001,MODE=A
ADARUN
```

Contents of RUNLOD CONTROL A1:

```
ADARUN PROG=ADALOD,DEVICE=dddd,DB=yyyyy
```

Contents of FILE015 LODC001 A1:

```
ADALOD LOAD FILE=1
ADALOD      NAME='TESTFILE-1'
ADALOD      MAXISN=50000,DSSIZE=10
ADALOD      TEMPSIZE=100,SORTSIZE=50
```

Update

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDEBAND,DSN=ADALOD.LODD015,MODE=A
DATADEF DDISN,DSN=ADALOD.ISN,MODE=A
DATADEF DDOLD,DSN=ADABASVv.OLDISN,MODE=A
DATADEF DDPRINT,DSN=ADALOD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADALOD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNLOD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=UPDATE.CONTROL,MODE=A
ADARUN
```

Contents of RUNLOD CONTROL A1:

```
ADARUN PROG=ADALOD,DEVICE=dddd,DB=yyyyy
```

Contents of UPDATE CONTROL A1:

```
ADALOD UPDATE FILE=1,DDISN,SAVEDREC

ADALOD        TEMPSIZE=100,SORTSIZE=50
ADALOD        DELISN=100-199,230,301-399
```

VSE

Data Set	Symbolic	Storage	Logical Unit	More Information
Associator	ASSORn	disk		[1]
Data Storage	DATARn	disk		[1]
Work	WORKR1	disk	[1]	Required for inactive nucleus
Compressed data	EBAND	tape disk	SYS010 [1]	
Recovery log (RLOG)	RLOGR1	disk		Required for the recovery log option
Temp area	TEMPR1	disk	[1]	
Temp overflow (optional)	FILEA	tape disk	SYS012 [1]	Stores descriptor values if the temp data set is too small.

Data Set	Symbolic	Storage	Logical Unit	More Information
Sort area	SORTR1	disk		With large files, split sort area across two volumes [2]
ISNs to be deleted	ISN	tape disk	SYS016 [1]	ISNs to be deleted
Deleted records	OLD	tape disk	SYS014 [1]	Deleted ISNs
ADALOD messages	--	printer	SYS009	ADALOD report, see also *Messages and Codes*
ADARUN messages	--	printer	SYSLST	*Messages and Codes*
ADARUN parameters	-- CARD CARD	reader tape disk	SYSRDR SYS000 [1]	
ADALOD parameters	-	reader	SYSIPT	

Notes:

1. Any programmer logical unit may be used.
2. Performance can be improved when sorting large files if the sort data set occupies two volumes. When using two volumes, each volume must be exactly half the size specified by the SORTSIZE parameter. If two data sets are used, both must be on the same device type (SORTDEV parameter).

ADALOD JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures (PROCs).

Load File

Refer to member ADALOD.X for this example.

```
* $$ JOB JNM=ADALOD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADALOD
*       SAMPLE FILE LOAD
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYSTEN,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'DEMO.FILE'
// MTC REW,SYS010
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
```

```
ADALOD LOAD FILE=1
ADALOD NAME='TESTFILE-1'
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

Update

Refer to member ADALODMU.X for this example.

```
* $$ JOB JNM=ADALODMU,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADALODMU
*       MASS UPDATE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,DISK,VOL=DISK01,SHR
// ASSGN SYS014,DISK,VOL=DISK02,SHR
// ASSGN SYS016,DISK,VOL=DISK03,SHR
// DLBL EBAND,'FILE.INPUT',,SD
// EXTENT SYS010,DISK01,1,0,sssss,nnnnn
// DLBL OLD,'FILE.OLD',,SD
// EXTENT SYS014,DISK02,1,0,sssss,nnnnn
// DLBL ISN,'FILE.ISN',,SD
// EXTENT SYS016,DISK03,1,0,sssss,nnnnn
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADALOD UPDATE FILE=1,LWP=400K,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=100
ADALOD DELISN=100-199,230,301-399
/*
/&
* $$ EOJ
```

XII ADAMER Utility: ADAM Estimation

This chapter covers the following topics:

Functional Overview
Estimate ADAM Access Requirements
ADAMER Output Report Description
JCL/JCS Requirements and Examples

103 Functional Overview

The ADAMER utility produces statistics that indicate the number of Data Storage accesses required to find and read a record when using an ADAM descriptor. This information is used to determine

- whether usage of the ADAM option would reduce the number of accesses required to retrieve a record using an ADAM descriptor as opposed to the standard Adabas accessing method;
- the amount of Data Storage space required to produce an optimum distribution of records based on the randomization of the ADAM descriptor.

The input data for ADAMER is a data set containing the compressed records of a file produced by the ADACMP or ADAULD utility.

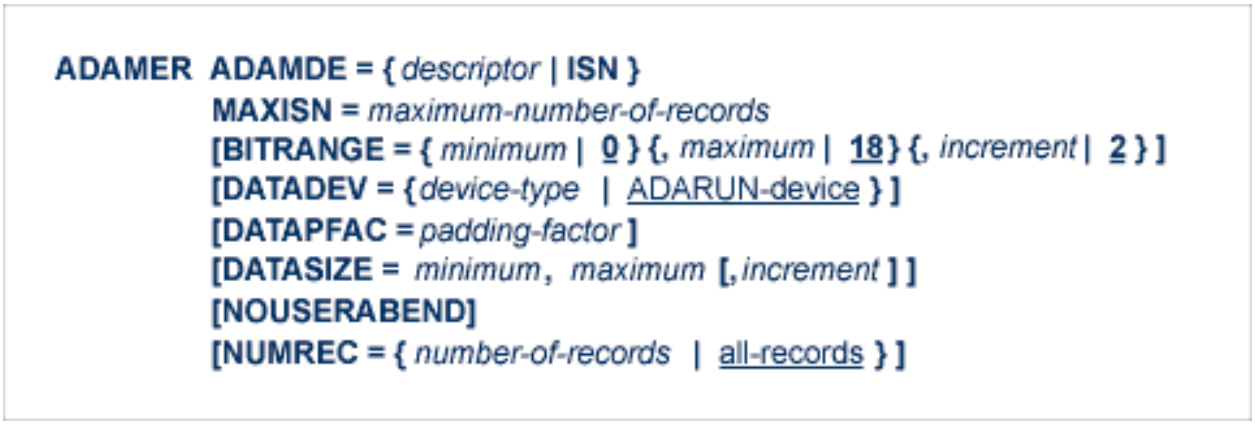
The field to be used as the ADAM descriptor is specified with the ADAMDE parameter. A multiple value field or a field contained within a periodic group may not be used. The ISN assigned to the record may be used instead of a descriptor as the basis for randomization (ADAMDE=ISN parameter).

The ADAM descriptor must contain a different value in each record, since the file cannot be successfully loaded with the ADAM option of the ADALOD utility if duplicate values are present for the ADAM descriptor. The ADAMER utility requires a descriptor field defined as unique (UQ), but does not check for unique values; checking for unique descriptor values is done by the ADALOD utility when loading the file as an ADAM file.

The BITRANGE parameter may be used to specify that a given number of bits are to be truncated from each ADAM descriptor value before the value is used as input to the randomization algorithm. This permits records containing ADAM descriptor values beginning with the same value (for example, 40643210, 40643220, 40643344) to be loaded into the same physical block in Data Storage. This technique can be used to optimize sequential reading of the file when using the ADAM descriptor to control the read sequence, or to remove insignificant information such as a check digit.

104 Estimate ADAM Access Requirements

- Essential Parameters 580
- Optional Parameters 580
- Examples 582



This chapter describes the syntax and parameters of the ADAMER utility.

Essential Parameters

ADAMDE: ADAM Key

Specifies the descriptor to be used as the ADAM key. If ISN is specified, ADAMER uses the ISN of each input record as input for the randomization algorithm.

The ADAM descriptor must be found in the field definition table (FDT) and be defined as a unique descriptor (UQ). It cannot be a sub-, super-, hyper-, collation, or phonetic descriptor. The descriptor also cannot specify the NU option, cannot be an MU field or a field within a periodic group, and cannot be a variable-length field.

MAXISN: Highest ISN to be Allocated for the File

The total number of records expected to be contained in the file.

MAXISN should include the number of records to be originally loaded plus the number of records that are likely to be added to the file.

Optional Parameters

BITRANGE: Bit Truncation for ADAM Key

The minimum, maximum, and incremental number of bits to be truncated from each ADAM descriptor value before the value is used as input to the ADAM randomization algorithm. Bits are always truncated from the rightmost portion of the compressed value.

A maximum of 20 different bit truncations is permitted for each ADAMER execution.

Example:

The following specification results in the truncation of 0 bits, 2 bits, and 4 bits for each Data Storage size for which statistics are provided.

```
BITRANGE=0,4,2 ↩
```

If this parameter is omitted, a default BITRANGE equal to 0,18,2 is used.

DATADEV: Data Storage Device Type

The device type to be used for Data Storage. If DATADEV is not specified, the device type specified by the ADARUN DEVICE parameter is the default.

DATAPFAC: Data Storage Padding Factor

The Data Storage padding factor to be used for the file. The number specified represents the percent of each Data Storage physical block that is not to be used during initial file loading. A value in the range 1-90 may be specified.

If this parameter is omitted, a padding factor of 10 percent is used during ADAMER execution.

DATASIZE: Data Storage Sizes for ADAM Estimates

The Data Storage sizes, in cylinders, for which ADAM statistics are to be provided. A maximum of four Data Storage sizes can be calculated per ADAM execution. The minimum and maximum values may be specified without the increment. ADAMER calculates two increments to produce a report based on all four values.

Example:

The following specification results in statistics for Data Storage sizes of 100, 125, 150, and 175 cylinders.

```
DATASIZE=100,175,25
```

If DATASIZE is omitted, ADAMER provides statistics for four Data Storage sizes as follows:

Size 1:	The first 100 input records are read and the Data Storage size requirement is based on the ADAM descriptor values present in these records and the value specified for MAXISN. The resulting Data Storage size is used as Data Storage Size 1.
Size 2:	Data Storage Size 1 x 1.33.
Size 3:	Data Storage Size 2 x 1.33.
Size 4:	Data Storage Size 3 x 1.33.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC: Maximum Number of Records to Read

The maximum number of records to be read from the input file. If NUMREC is not specified, *all* records are read.

Examples

Example 1:

```
ADAMER ADAMDE=CC,
ADAMER DATADEV=3350,DATASIZE=50,110,20,
ADAMER DATAPFAC=10,MAXISN=225000,BITRANGE=2,6,1
```

The ADAM descriptor is CC. Model 3350 device type is to be used for Data Storage. Statistics for Data Storage sizes of 50, 70, 90, and 110 cylinders are to be provided. Data Storage padding factor of 10 percent is to be used. The planned number of records for the file is 225,000. For each Data Storage size, statistics are to be provided for bit truncations of 2, 3, 4, 5, and 6 bits.

Example 2:

```
ADAMER ADAMDE=CD,DATADEV=3380,DATAPFAC=5,MAXISN=80000
```

The ADAM descriptor is CD. Model 3380 device type is to be used for Data Storage. Data Storage padding factor of 5 percent is to be used. The planned number of records for the file is 80,000. Default values are to be used for all other parameters.

105 ADAMER Output Report Description

The following entries appear on the report produced by ADAMER:

Field	Explanation
LOADISNS	Number of records contained in the input data set.
MAXISN	Total file records (see the MAXISN parameter description).
DATA DEVICE	Data Storage device type (see the DATADEV parameter description).
DATAPFAC	Data Storage padding factor (see DATAPFAC parameter description).

The following fields appear under "AVERAGE NUMBER OF EXCPs":

Field	Explanation
Data Storage SIZE	See the DATASIZE parameter description. The number of cylinders is rounded up to the nearest integer.
BIT-PARM	See the BITRANGE parameter description.
FOR LOADISNS	The average number of I/Os required to find and read a record when the ADAM descriptor is used. This result assumes that the number of records in the file is equal to the number of records contained in the input data set.
DISK USAGE	The percentage of Data Storage space occupied after initial loading of the file. This result assumes that the number of records to be loaded is equal to the number of records contained in the input data set.
FOR MAXISN	The average number of I/Os required to find and read a record when using the ADAM descriptor. This result assumes that the number of records in the file is equal to the value specified with the MAXISN parameter.
DISK USAGE	The percentage of Data Storage space occupied after initial loading of the file. This result assumes that the number of records to be loaded is equal to the number of records specified with the MAXISN parameter.

Using the information contained on the ADAMER report, the user can determine

- the optimum balance between access and Data Storage space requirements; and
- the optimum number of bits that should be truncated from each ADAM descriptor value so that records containing similar beginning values are loaded into the same physical block. This is necessary only if optimization of sequential reading is desired.

106 JCL/JCS Requirements and Examples

- BS2000 586
- z/OS 587
- z/VM 588
- VSE 589

This section describes the job control information required to run ADAMER with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Input data	DDEBAND	tape/ disk	Output of ADACMP or ADAULD utility
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAMER parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAMER messages/report	SYSLST/ DDDRUCK		*Messages and Codes*

ADAMER JCL Example (BS2000)

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A M E R ALL FUNCTIONS
/REMARK *
/ASS-SYSLST L.MER
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDEBAND,CMP.AUS
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAMER,DB=yyyyy,IDTNAME=ADABAS5B
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1
ADAMER MAXISN=10000
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAMER LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A M E R ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.MER
/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.AUS,LINK=DDEBAND
/EXEC (ADARUN,ADA.MOD)
```

```
ADARUN PROG=ADAMER,DB=yyyyy,IDTNAME=ADABAS5B
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1
ADAMER MAXISN=10000
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Input data	DDEBAND	tape/ disk	Output of ADACMP or ADAULD utility
ADARUN parameters	DDCARD	reader	*Operations*
ADAMER parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAMER messages/report	DDDRUCK	printer	*Messages and Codes*

ADAMER JCL Example (z/OS)

Refer to ADAMER in the JOBS data set for this example.

```
//ADAMER    JOB
//*
//*   ADAMER:
//*      ADAM ESTIMATION
//*
//MER      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.COMPR1 <=== COMPRESS
DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAMER,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAMER MAXISN=1000,ADAMDE=AA,BITRANGE=0,2,4
ADAMER DATADEV=eeee,DATAPFAC=10,DATASIZE=100,175,25
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Input data	DDEBAND	tape/ disk	Output of ADACMP or ADAULD utility
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAMER parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADAMER messages/report	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADAMER JCL Example (z/VM)

```
DATADEF DDEBAND,DSN=ADABASVv.BAND,MODE=A
DATADEF DDPRINT,DSN=ADAMER.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAMER.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNMER.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAMER.CONTROL,MODE=A
ADARUN
```

Contents of RUNMER CONTROL A1:

```
ADARUN PROG=ADAMER,DEVICE=dddd,DB=yyyyy
```

Contents of ADAMER CONTROL A1:

```
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1 ADAMERMAXISN=10000
```

VSE

File	Sym. Name	Storage	Logical Unit	More Information
Input data	EBAND	tape disk	SYS010 *	Output of ADACMP or ADAULD utility
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations*
ADAMER parameters	-	reader	SYSIPT	
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADAMER messages/report	-	printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADAMER JCS Example (VSE)

See ***Library and File Procedures for VSE Examples*** for a description of the VSE procedures (PROCs). Refer to member ADAMER.X for this example.

```
* $$ JOB JNM=ADAMER,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAMER
// OPTION LOG,PARTDUMP
*       ADAM ESTIMATION
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// DLBL EBAND,'EXAMPLE.DByyyyy.COMPR1',0,SD
// EXTENT SYS004
// ASSGN SYS004,DISK,VOL=DISK01,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAMER,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAMER MAXISN=1000,ADAMDE=AA,BITRANGE=0,2,4
ADAMER DATADEV=eeee,DATAPFAC=10,DATASIZE=100,175,25
/*
/&
* $$ EOJ
```

XIII ADAORD Utility: Reordering Functions

This chapter covers the following topics:

Functional Overview
REORASSO: Reorder Associator
REORDATA: Reorder Data Storage
REORDB: Reorder Database
REORFASSO: Reorder Associator for a Single File
REORFDATA: Reorder Data Storage for a Single File
REORFILE: Reorder File
RESTRUCTUREDB: Restructure Database
RESTRUCTUREF: Restructure Single Files
STORE: Store Files
JCL/JCS Requirements and Examples

107 Functional Overview

- Reorder Functions 594
- Restructure Functions 595
- Store Function 595
- Space Allocation 596
- Adabas 8 Considerations 596

The ADAORD utility can be used to:

- Reorder the Associator or Data Storage for a database or a single file in a database (REORASSO, REORFASSO, REORDATA, REORFDATA, REORDB, and REORFILE functions)
- Restructure a database or a single file in a database and store the resulting output files into an existing database (RESTRUCTUREDB, RESTRUCTUREF, and STORE functions).

Only one function may be executed during a given execution of ADAORD.

Parts of the database are overwritten during ADAORD execution. We therefore recommend that you back up the database (or file) using the ADASAV utility before running ADAORD functions.

In addition, all ADAORD functions except RESTRUCTUREF (file) require exclusive EXF control of the database files involved in the operation. RESTRUCTUREF requires EXU control; other users may access database files being used by RESTRUCTUREF, but only for reading. Note, however, that operations involving checkpoint, security, or files loaded using ADALOD's SYSFILE option require exclusive database control.

Note: When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

Reorder Functions

The REORASSO function physically reorders all Associator blocks for all files; the REORFASSO function reorders the Associator for a single file. This eliminates Associator space fragmentation and combines multiple address converter, normal and upper index, and Data Storage Space Table (DSST) component extents into a single logical extent for each component.

The REORDATA function reorders Data Storage for all files in the database; the REORFDATA function reorders Data Storage for a single file. This condenses extents containing only empty blocks, and also eliminates any Data Storage fragmentation caused by file deletion.

The REORDB function performs both the REORASSO and REORDATA functions in a single execution of ADAORD.

The REORFILE function performs both the REORFASSO and REORFDATA functions in a single execution of ADAORD. The records may be reordered in the logical sequence by a descriptor, by ISN, or in the current sequence.

The REORDATA, REORDB, REORFDATA and STORE functions do not reorder ADAM files. However, these functions can be used to relocate an ADAM file to different RABNs.

Restructure Functions

The RESTRUCTUREDB function unloads an entire database to a sequential data set; the RESTRUCTUREF function unloads one or more files to a sequential data set. This data set containing unloaded data can be used as input to the STORE function.

The RESTRUCTURE functions are used to relocate the database to a different physical device or a file or files to another device.

The format of the sequential data set produced by the RESTRUCTURE functions is independent of the database device type, and is *not* compatible with the format required by the ADALOD or ADASAV utilities. Therefore, the target database may be contained on a device type different from the source database.

The Associator and Data Storage are reordered as part of RESTRUCTURE/STORE processing.

When RESTRUCTUREDB/F restructures an ADAM file that uses the overflow area, and then STORE stores the restructured file in a database with a smaller DATA block size, an ADAORD ERROR-103 may occur. Use the ADAULD and ADALOD utilities to move ADAM files instead.

Store Function

The STORE function loads one or more files into an existing database using the DDFILEA output created by the RESTRUCTUREDB, RESTRUCTUREF, or REORDB function.

The Associator and Data Storage are reordered as part of RESTRUCTURE/STORE processing.

The STORE function does not reorder ADAM files. However, it can be used, in combination with other ADAORD functions, to relocate an ADAM file to different RABNs. When the RESTRUCTUREDB or RESTRUCTUREF functions restructure an ADAM file that uses the overflow area, and then STORE stores the restructured file in a database with a smaller DATA block size, an ADAORD ERROR-103 may occur. Use the ADAULD and ADALOD utilities to move ADAM files instead.

Space Allocation

ADAORD allocates the amount of space required by the *xx*SIZE or MAXISN and MAXISN2 parameters, if specified. Otherwise, ADAORD allocates space based on the current size of the file. Note that the *xx*RELEASE parameters affect the amount of space required.

If possible, space is allocated on the volume specified by the *xxxx*VOLUME parameter. If insufficient free space is available on the specified volume, ADAORD allocates the remainder of the required space on other volumes, according to its default rules of allocation.

An *xx*RABN parameter overrides the associated *xxxx*VOLUME parameter.

Adabas 8 Considerations

You can restructure databases and files from an Adabas version prior to Adabas 8 and store them in an Adabas 8 database using ADAORD STORE. However, you cannot store the restructured output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7). If you attempt this, the following warning will be generated and ADAORD will end with a CC=4:

```
*** Warning: The input data set is from V8 and will not be processed
```

108 REORASSO: Reorder Associator

- Optional Parameters and Subparameters 599
- Examples 603

The REORASSO function reorders the entire Associator. If a file is not explicitly specified, its related Associator information is reordered according to its existing definition. To reorder Associator information for specific files, use the REORFASSO function.

This function requires exclusive EXF control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

This is the syntax of the ADAORD REORASSO function:

Note: If the parameter MAXFILES or NEWDBID is specified, an active nucleus will terminate automatically at the end of the REORASSO function.

Optional Parameters and Subparameters

ACRABN: Starting RABN for Address Converter

The beginning RABN for the file's address converter extent. If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

AC2RABN: Starting RABN for Secondary Address Converter

The beginning RABN for the file's secondary address converter extent. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent. If the file contains no secondary address converter extents, this parameter is ignored.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOPFAC: Associator Padding Factor

The new Associator block padding factor. The number specified represents the percentage of each Associator block not to be used during the reorder process. A value in the range 1-90 may be specified. The remaining number of bytes after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current Associator padding factor in effect for the file is used.

ASSOVOLUME: Associator Extent Volume

Note: The value for the ASSOVOLUME parameter must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the corresponding file's Associator space (that is, the AC, NI, and UI extents) should be allocated. If the requested number of blocks cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default rules of allocation.

If ACRABN, UIRABN, or NIRABN is specified, ADAORD ignores the ASSOVOLUME value when allocating the corresponding extent type.

If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADAORD's default allocation rules.

DBINDEXCOMPRESSION: Compress Database Indexes

DBINDEXCOMPRESSION indicates whether the indexes of files are rebuilt in compressed or uncompressed form. It applies to all files for which no INDEXCOMPRESSION parameter is specified.

DBINDEXCOMPRESSION can be used to build compressed or uncompressed indexes for all files of the database, making it unnecessary to specify index compression for each file.

FILE: File Number

The file number to which the following parameters apply. Each specified file and its parameters should be on a separate ADAORD statement following the ADAORD REORASSO function statement.

For any file whose number is not specified, current Associator block padding factor and MAXISN value are retained, and all Associator space allocations remain the same.

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index for the file is rebuilt in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is *not* specified

- but the DBINDEXCOMPRESSION parameter is specified for the database as a whole, the default is the database value.
- and DBINDEXCOMPRESSION is also *not* specified, the default is the current compression form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LPB: Prefetch Buffer Size

Specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXFILES: Maximum Number of Files

MAXFILES specifies the maximum number of files that can be loaded into the database. The minimum value for this parameter is 3. The highest value permitted is 5000 or one less than the ASSOR1 block size, whichever is lower. For example, 2003 is the highest MAXFILES value for a database whose ASSOR1 is stored on a 3380 DASD.

If this parameter is omitted, the current value for MAXFILES is retained.

When MAXFILES is specified, the nucleus terminates after the ADAORD REORASSO function is completed.

MAXISN: Highest ISN Permitted for the File

MAXISN specifies the highest ISN that can be allocated for the file. This value must be greater than the current TOPISN value displayed in the ADAREP database report.

ADAORD uses the specified value to calculate the address converter space required. If this parameter is omitted, the current MAXISN value for the file is retained.

MAXISN2: Highest Secondary ISN Permitted for the File

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. This value must be greater than the current TOP AC2 ISN value displayed in the ADAREP database report. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADAORD uses the specified value to calculate the space required in the secondary address converter for the file. If this parameter is omitted, the current MAXISN2 value for the file is retained. If the file contains no secondary address converter extents, this parameter is ignored.

NEWDBID: Database Identifier

NEWDBID is the ID to be assigned to the database. A value in the range 1-65535 may be used. For systems using Online System Security, the value 999 is reserved. If this parameter is omitted, the current database ID is retained.

When NEWDBID is specified, the nucleus terminates after the ADAORD REORASSO function is completed.

NEWDBNAME: Database Name

The name to be assigned to the database. The name assigned may be from 1 to 16 characters. If this parameter is omitted, the current database name is retained.

If the database name contains special characters or embedded blanks, the name must be enclosed within apostrophes ('...'), which themselves must be doubled if included in the name; for example, 'JAN''S DB'.

NIRABN: Starting RABN for Normal Index

NIRABN specifies the beginning RABN number for the normal index extent. If this parameter is omitted, ADAORD assigns the starting RABN.

NIRELEASE: Release Unused Normal Index Blocks

Specifying NIRELEASE releases unused normal index (NI) blocks belonging to the specified file. If NIRELEASE is not specified, ADAORD allocates *at least* the number of NI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

NISIZE: Normal Index Size

NISIZE is the number of blocks or cylinders to be allocated for the normal index. If the value is blocks, it must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

RAID: Action to Follow Determination That File Does Not Exist

The RAID parameter instructs ADAORD to ignore any FILE parameters that refer to a file that does not exist in the database.

If RAID is not specified (the default), ADAORD terminates with an error message when it encounters a FILE parameter referring to a file that does not exist in the database.

The RAID parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

RPLUPDATEONLY: Allow Only Event Replicator Processing Updates

The `RPLUPDATEONLY` parameter can be used in the ADAORD REORASSO function to indicate whether this Adabas database file may be updated only by the Event Replicator Server as part of Adabas-to-Adabas replication or by other means as well. This parameter is optional. Valid values are "YES" or "NO". A value of "YES" indicates that the file can only be updated via Event Replicator processing; a value of NO indicates that the file can be updated by any normal means, including Event Replicator processing. There is no default; if no value is specified for the `RPLUPDATEONLY` parameter in the ADAORD REORASSO function, the value used previously for the file is used.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables *cannot* be tested: only the syntax of the specified parameters can be tested.

UIRABN: Starting RABN for Upper Index

UIRABN is the beginning RABN number for the file's upper index extent. If this parameter is omitted, ADAORD assigns the starting RABN for each of these extents.

UIRELEASE: Release Unused Upper Index Blocks

Specifying UIRELEASE releases unused upper index (UI) blocks belonging to the specified file. If UIRELEASE is not specified, ADAORD allocates *at least* the number of UI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

UISIZE: Upper Index Size

UISIZE is the number of blocks or cylinders to be allocated for the upper index. If the value is blocks, it must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

Examples

Example 1:

```
ADAORD REORASSO ↩
```

The Associator is to be reordered.

Example 2:

```
ADAORD   REORASSO
ADAORD      MAXFILES=200
ADAORD      NEWDBID=6,NEWDBNAME=DATABASE-6
```

The Associator is to be reordered. A maximum of 200 files are permitted for the database. The database ID and name are to be 6 and DATABASE-6, respectively.

Example 3:

```
ADAORD   REORASSO
ADAORD      FILE=1,ACRABN=1000,NIRABN=2200,
ADAORD      FILE=2,MAXISN=500000,
ADAORD      FILE=4,ASSOPFAC=5
```

The Associator is to be reordered. The address converter allocation for file 1 is to begin with RABN 1,000. The normal index for file 1 is to begin with RABN 2,200. The MAXISN for file 2 is to be set to 500,000. The Associator block padding factor for file 4 is to be set to 5 percent. The Associator information for all other database files is reordered according to each file's current definition.

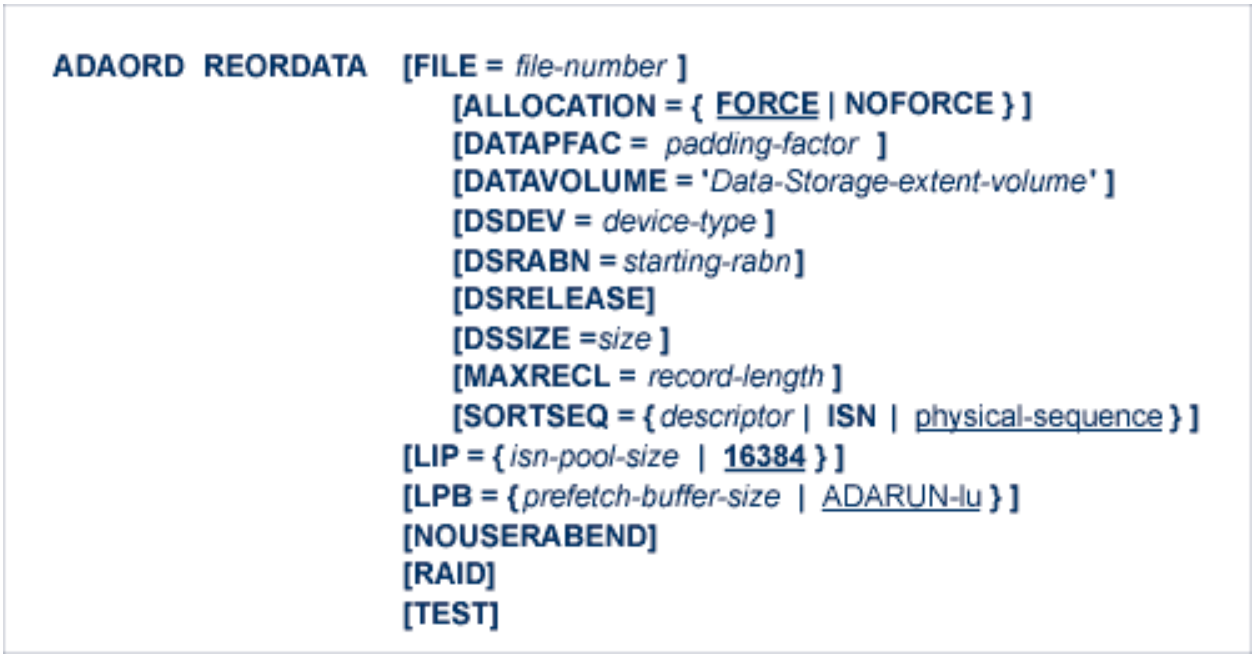
109 REORDATA: Reorder Data Storage

- Optional Parameters and Their Subparameters 606
- Examples 609

The REORDATA function reorders Data Storage for *all* files. Files not specified are reordered according to their existing definitions. The REORDATA function does not reorder ADAM files. However, it can be used to relocate an ADAM file to different RABNs.

This function requires exclusive EXF control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

This is the syntax of the ADAORD REORDATA function:



Optional Parameters and Their Subparameters

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameter DSRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameter fails, the utility retries the allocation without the placement parameter.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor, which is the percentage of each Data Storage block reserved for record expansion when the file is reordered. A value in the

range 1-90 may be specified (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). If this parameter is omitted, the current padding factor for the file is used.

DATAVOLUME: Data Storage Extent Volume

Note: The value for the DATAVOLUME parameter must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If DATAVOLUME is not specified, the Data Storage space is allocated based on the current size of the file. The DSRELEASE parameter also affects the amount of space required.

DSDEV: Data Storage Device Type

DSDEV is the file's Data Storage device type. The specified device type must already be defined to Adabas, normally when the database was created or by the ADADBS utility's ADD function.

If DSDEV is not specified, ADAORD attempts to allocate the file on the device type used before reordering.

DSRABN: Data Storage Starting RABN

The beginning RABN for the specified file's Data Storage extent. If this parameter is omitted, ADAORD assigns the starting RABN.

DSRELEASE: Release Unused Data Storage Blocks

Specifying DSRELEASE releases unused Data Storage (DS) blocks belonging to the specified file. If DSRELEASE is not specified, ADAORD allocates *at least* the number of DS blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

DSSIZE: Data Storage Size

DSSIZE is the number of blocks or cylinders to be allocated for the file's Data Storage (DS) logical extent. If the value is blocks, it must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the DATAPFAC padding factor used.

FILE: File Number

FILE is the file to which the following parameters apply. Each specified file and its parameters should be on a separate ADAORD statement following the ADAORD REORDATA statement.

For any file whose number is not specified, the file is reordered using the current physical sequence, and the current Data Storage padding factor and space allocation are retained.

LIP: ISN Buffer Pool Size

The LIP parameter can be used to decrease the number of Associator I/O operations when recreating the address converter. For best performance, specify a size that accepts all ISNs of the largest file to be processed.

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 16384 bytes (or 16K).

The length of one input record is ISNSIZE + RABNSIZE. Thus the entry length is at least 6 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 8 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

Note: When ADAORD is processing files that contain spanned records with secondary ISNs, a second LIP will be allocated to contain these ISNs.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXRECL: Maximum Compressed Record Length

Use the MAXRECL parameter to change the maximum record length, after compression, permitted in the file. Specifying MAXRECL has two effects:

The DATA data set for the file can be allocated only to devices that support the specified length.

If the file contains Data Storage records that exceed the specified length, ADAORD abends and prints ERROR-126 (Data Storage record too long).

If MAXRECL is not specified, the maximum compressed record length does not change.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

RAID: Action to Follow Determination That File Does Not Exist

The RAID parameter instructs ADAORD to ignore any FILE parameters that refer to a file that does not exist in the database.

If RAID is not specified (the default), ADAORD terminates with an error message when it encounters a FILE parameter referring to a file that does not exist in the database.

The RAID parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

SORTSEQ: Record Processing Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADAORD REORDATA
```

Data Storage for the entire database is to be reordered.

Example 2:

```
ADAORD   REORDATA
ADAORD              FILE=1,DSRABN=1000,DSSIZE=200B
ADAORD              FILE=4,SORTSEQ=AA
ADAORD              FILE=5,DATAPFAC=15
```

Data Storage is to be reordered. Data Storage for file 1 is to begin with RABN 1,000 with 200 blocks to be allocated. File 4 is to be reordered using descriptor AA for sequence control. The Data Storage block padding factor for file 5 is to be set to 15 percent. All other database files are reordered according to their existing definitions.

110 REORDB: Reorder Database

- Optional Parameters and Subparameters 614
- Examples 620

The REORDB function reorders the entire Associator and Data Storage for a database. Files that are not specified are reordered according to their existing definitions.

The REORDB function does not reorder ADAM files. However, it can be used to relocate an ADAM file to different RABNs.

This function requires exclusive EXF control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

This is the syntax of the ADAORD REORDB function:

```
ADAORD  REORDB    [DBINDEXCOMPRESSION = { YES | NO } ]
                  [FILE = file-number ]
                      [ACRABN = starting-rabn ]
                      [AC2RABN = starting-rabn ]
                      [ALLOCATION = { FORCE | NOFORCE } ]
                      [ASSOPFAC = padding-factor ]
                      [ASSOVOLUME = 'Associator-extent-volume ' ]
                      [DATAPFAC = padding-factor ]
                      [DATAVOLUME = 'Data-Storage-extent-volume' ]
                      [DSDEV = device-type ]
                      [DSRABN = starting-rabn]
                      [DSRELEASE]
                      [DSSIZE =size ]
                      [INDEXCOMPRESSION = { YES | NO } ]
                      [ISNSIZE = { 3 | 4 } ]
                      [MAXISN = highest-isn ]
                      [MAXISN2 = highest-isn ]
                      [MAXRECL = record-length ]
                      [NIRABN = starting-rabn]
                      [NIRELEASE]
                      [NISIZE = size ]
                      [SORTSEQ = {descriptor | ISN | physical-sequence } ]
                      [UIRABN = starting-rabn ]
                      [UIRELEASE]
                      [UISIZE = size ]
                  [LIP = {isn-pool-size | 16384 } ]
                  [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                  [MAXFILES = maximum-number-files ]
                  [NEWDBID = database-identifier ]
                  [NEWDBNAME = database-name]
                  [NOUSERABEND]
                  [RAID]
                  [TEST]
```

Note: If the parameter MAXFILES or NEWDBID is specified, the nucleus automatically terminates at the end of the REORDB function.

Optional Parameters and Subparameters

ACRABN: Starting RABN for Address Converter

The RABN with which the file's address converter extent is to begin. If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

AC2RABN: Starting RABN for Secondary Address Converter

The beginning RABN for the file's secondary address converter extent. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent. If the file contains no secondary address converter extents, this parameter is ignored.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOPFAC: Associator Padding Factor

The new Associator block padding factor. The number specified represents the percentage of each Associator block not to be used during the reorder process. A value in the range 1-90 may be specified. The remaining number of bytes after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current Associator padding factor in effect for the file is used.

ASSOVOLUME: Associator Extent Volume

Note: The value for the ASSOVOLUME parameter must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) should be allocated. If the requested number of blocks cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If ACRABN, UIRABN, or NIRABN is specified, ADAORD ignores the ASSOVOLUME value when allocating the corresponding extent type.

If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADAORD's default allocation rules.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor, which is the percentage of each Data Storage block reserved for record expansion when the file is reordered. A value in the range 1-90 may be specified (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). If this parameter is omitted, the current padding factor for the file is used.

DATAVOLUME: Data Storage Extent Volume

Note: The value for the DATAVOLUME parameter must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated. If the requested number of blocks requested with DSSIZE cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If DATAVOLUME is not specified, the Data Storage space is allocated according to ADAORD's default allocation rules.

DBINDEXCOMPRESSION: Compress Database Indexes

DBINDEXCOMPRESSION indicates whether the indexes of files are rebuilt in compressed or uncompressed form. It applies to all files for which no INDEXCOMPRESSION parameter is specified.

DBINDEXCOMPRESSION can be used to build compressed or uncompressed indexes for all files of the database, making it unnecessary to specify index compression for each file.

DSDEV: Data Storage Device Type

DSDEV is the file's Data Storage device type. The specified device type must already be defined to Adabas, normally when the database was created or by the ADADBS utility's ADD function.

If DSDEV is not specified, ADAORD attempts to allocate the file on the device type used before reordering.

DSRABN: Data Storage Starting RABN

The beginning RABN for the file's Data Storage extent. If this parameter is omitted, ADAORD assigns the starting RABN.

DSRELEASE: Release Unused Data Storage Blocks

Specifying DSRELEASE releases unused Data Storage (DS) blocks belonging to the specified file. If DSRELEASE is not specified, ADAORD allocates *at least* the number of DS blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

DSSIZE: Data Storage Size

DSSIZE is the number of blocks or cylinders to be allocated for the file's Data Storage (DS) logical extent. If the value is blocks, it must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the DATAPFAC padding factor.

FILE: File Number

The file to which the following parameters apply. Each specified file must be on a separate ADAORD statement following the ADAORD REORDB function statement, and must be immediately followed by the applicable parameters for the file.

For any file whose number is not specified, the current Associator and Data Storage block padding factors and MAXISN value are retained, and all Associator and Data Storage space allocations remain the same.

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index for the file is rebuilt in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is *not* specified

- but the DBINDEXCOMPRESSION parameter is specified for the database as a whole, the default is the database value.
- and DBINDEXCOMPRESSION is also *not* specified, the default is the current compression form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LIP: ISN Buffer Pool Size

The LIP parameter can be used to decrease the number of Associator I/O operations when recreating the address converter. For best performance, specify a size that accepts all ISNs of the largest file to be processed.

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 16384 bytes (or 16K).

The length of one input record is ISNSIZE + RABNSIZE. Thus the entry length is at least 6 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 8 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

Note: When ADAORD is processing files that contain spanned records with secondary ISNs, a second LIP will be allocated to contain these ISNs.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32,760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXFILES: Maximum Number of Files

MAXFILES specifies the maximum number of files that can be loaded into the database. The minimum value for this parameter is 3. The highest value permitted is 5000 or one less than the ASSOR1 block size, whichever is lower. For example, 2003 is the highest MAXFILES value for a database whose ASSOR1 is stored on a 3380 DASD.

If this parameter is omitted, the current value for MAXFILES is retained.

When MAXFILES is specified, the nucleus terminates after the ADAORD REORDB function is completed.

MAXISN: Highest ISN Permitted in the File

The highest ISN that can be allocated for the file. This value must be greater than the current TOPISN value displayed in the ADAREP database report.

ADAORD uses the specified value to calculate the address converter space required. If this parameter is omitted, the current MAXISN value for the file is retained.

MAXISN2: Highest Secondary ISN Permitted for the File

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. This value must be greater than the current TOP AC2 ISN value displayed in the ADAREP database report. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADAORD uses the specified value to calculate the space required in the secondary address converter for the file. If this parameter is omitted, the current MAXISN2 value for the file is retained. If the file contains no secondary address converter extents, this parameter is ignored.

MAXRECL: Maximum Compressed Record Length

Use the MAXRECL parameter to change the maximum record length, after compression, permitted in the file. Specifying MAXRECL has two effects:

- The file's DATA data set is allocated only to devices that support the specified length.
- If the file contains Data Storage records that exceed the specified length, ADAORD abends and prints the ERROR-126 message (Data Storage record too long).

If MAXRECL is not specified, the maximum compressed record length does not change.

NEWDBID: Database Identifier

NEWDBID is the ID to be assigned to the database. A value in the range 1-65,535 may be used. For systems using Adabas Online System Security, the value 999 is reserved. If this parameter is omitted, the current database ID is retained.

When NEWDBID is specified, the nucleus terminates after the ADAORD REORDB function is completed.

NEWDBNAME: Database Name

NEWDBNAME specifies the name to be assigned to the database. The name can contain up to 16 characters. If the name contains special characters or embedded blanks, it must be enclosed in apostrophes ('...'); for example, 'JAN''S DB'. If this parameter is omitted, the current database name is retained.

NIRABN: Starting RABN for Normal Index

NIRABN specifies the RABN with which the file's normal index extent is to begin. If this parameter is omitted, ADAORD assigns the starting RABN.

NIRELEASE: Release Unused Normal Index Blocks

Specifying NIRELEASE releases unused normal index (NI) blocks belonging to the specified file. If NIRELEASE is not specified, ADAORD allocates *at least* the number of NI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

NISIZE: Normal Index Size

NISIZE specifies the number of blocks or cylinders to be allocated for the file's normal index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

RAID: Action to Follow Determination That File Does Not Exist

The RAID parameter instructs ADAORD to ignore any FILE parameters that refer to a file that does not exist in the database.

If RAID is not specified (the default), ADAORD terminates with an error message when it encounters a FILE parameter referring to a file that does not exist in the database.

The RAID parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

SORTSEQ: File Processing Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UIRABN: Starting RABN for Upper Index

UIRABN defines the beginning RABN for the Associator's upper index extent for the file. If this parameter is omitted, ADAORD assigns the starting RABN.

UIRELEASE: Release Unused Upper Index Blocks

UIRELEASE releases unused upper index (UI) blocks belonging to the file. If UIRELEASE is not specified, ADAORD allocates *at least* the number of UI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

UISIZE: Upper Index Size

UISIZE specifies the number of blocks or cylinders to allocate for the upper index. A block count must be followed by a "B" (for example, "2000B"). If UISIZE is omitted, ADAORD allocates space in proportion to an increase or decrease in the ASSOPFAC padding factor.

Examples

Example 1:

```
ADAORD   REORDB
```

The Associator and Data Storage are to be reordered. No changes are to be made to the current database parameters.

Example 2:

```
ADAORD   REORDB
ADAORD          MAXFILES=200
ADAORD          NEWDBID=6,NEWDBNAME=DATABASE-6
```

The Associator and Data Storage are to be reordered. A maximum of 200 files are permitted for the database. The database ID and name are to be 6 and DATABASE-6, respectively. If the nucleus is active during the REORDB operation, it will be stopped following the operation (NEWDBID was specified).

Example 3:

```
ADAORD   REORDB
ADAORD          FILE=1,ACRABN=1000,NIRABN=2200,SORTSEQ=ISN
ADAORD          FILE=2,MAXISN=500000
ADAORD          FILE=4,ASSOPFAC=5,DATAPFAC=20,DSSIZE=5,DSRABN=1
```

The Associator and Data Storage are to be reordered. The address converter allocation for file 1 is to begin with RABN 1,000. The normal index allocation for file 1 is to begin with RABN 2,200.

The Data Storage portion of file 1 is to be reordered in ascending ISN sequence. The MAXISN for file 2 is to be set to 500,000. The following assignments are made for file 4: the Associator block padding factor is to be changed to 5 percent, the Data Storage block padding factor is set to 20 per cent, and a new DSSIZE of 5 cylinders is assigned starting at RABN 1. All other files are reordered according to their existing definitions.

Example 4:

```
ADAORD REORDB
ADAORD FILE=66
ADAORD DSRELEASE
ADAORD NIRELEASE
ADAORD UIRELEASE
ADAORD RAID
```

1. ADAORD reorders the entire database.
2. ADAORD releases all unused storage from the Data Storage, normal index, and upper index of file 66.

3. However, if file 66 does not exist in the database, ADAORD does not terminate with an error message; rather, ADAORD ignores this condition and proceeds.

Example 5:

```
ADAORD REORDB
ADAORD DBINDEXCOMPRESSION=YES
ADAORD FILE=1
ADAORD FILE=2,INDEXCOMPRESSION=NO
ADAORD FILE=3
```

All files are reordered and rebuilt with compressed indexes, except for file 2, which is rebuilt with an uncompressed index.

111 REORFASSO: Reorder Associator for a Single File

- Essential Parameter 625
- Optional Parameters 625
- Examples 780

The REORFASSO function reorders the Associator for a single file. Associator information for unspecified files is not reordered.

This function requires exclusive EXU control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

This is the syntax of the ADAORD REORFASSO function:

```
ADAORD REORFASSO FILE = file-number
                        [ACRABN = starting-rabn ]
                        [AC2RABN = starting-rabn ]
                        [ALLOCATION = { FORCE | NOFORCE } ]
                        [ASSOPFAC = padding-factor  ]
                        [ASSOVOLUME = 'Associator-extent-volume ' ]
                        [INDEXCOMPRESSION = { YES | NO } ]
                        [ISNSIZE = { 3 | 4 } ]
                        [MAXISN = highest-isn ]
                        [MAXISN2 = highest-isn ]
                        [NIRABN = starting-rabn ]
                        [NIRELEASE]
                        [NISIZE = size ]
                        [PASSWORD = password ]
                        [UIRABN = starting-rabn ]
                        [UIRELEASE]
                        [UISIZE = size ]
                    [EXCLUDE = file-list ]
                    [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                    [NOUSERABEND]
                    [TEST]
```

Essential Parameter

FILE: File Number

FILE specifies the file to be processed, and to which the parameters that follow in the statement sequence apply. Several files and their related parameters may be specified within one REORFASSO operation; see the examples at the end of this section. If a component file of an Adabas expanded file is specified, only that file's Associator is reordered; this has no adverse effect on the other component files.

Optional Parameters

ACRABN: Starting RABN for Address Converter

ACRABN specifies the file's starting address converter RABN. If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

AC2RABN: Starting RABN for Secondary Address Converter

The beginning RABN for the file's secondary address converter extent. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent. If the file contains no secondary address converter extents, this parameter is ignored.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOPFAC: Associator Padding Factor

ASSOPFAC defines the new Associator block padding factor, which is the percentage of each Associator block *not* used during the reorder process. Specify a value in the range 1-90. The number of bytes free after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current padding factor in effect for the file is used.

ASSOVOLUME: Associator Extent Volume

Note: The value for the ASSOVOLUME parameter must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) should be allocated. If the requested number of blocks cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If ACRABN, UIRABN, or NIRABN is specified, ADAORD ignores the ASSOVOLUME value when allocating the corresponding extent type.

If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADAORD's default allocation rules.

EXCLUDE: Exclude Specified Files from Reorder

EXCLUDE lists the numbers of the files to be excluded from REORDER processing; that is, the files that are not to be reordered.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

Files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index for the file is rebuilt in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is not specified, the default is the current form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32,760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXISN: Highest ISN to Be Allocated

MAXISN is the highest ISN which may be allocated for the file. This value must be greater than the current TOPISN value displayed in the ADAREP database report.

ADAORD uses the specified value to calculate the address converter space required. If this parameter is omitted, the current MAXISN value for the file remains in effect.

MAXISN2: Highest Secondary ISN Permitted for the File

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. This value must be greater than the current TOP AC2 ISN value displayed in the ADAREP database report. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADAORD uses the specified value to calculate the space required in the secondary address converter for the file. If this parameter is omitted, the current MAXISN2 value for the file is retained. If the file contains no secondary address converter extents, this parameter is ignored.

NIRABN: Starting RABN for Normal Index

NIRABN is the starting RABN to be used for the normal index. If NIRABN is omitted, ADAORD assigns the starting RABN.

NIRELEASE: Release Unused Normal Index Blocks

Specifying NIRELEASE releases unused normal index (NI) blocks belonging to the specified file. If NIRELEASE is not specified, ADAORD allocates *at least* the number of NI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

NISIZE: Normal Index Size

NISIZE specifies the number of blocks or cylinders to be allocated for the file's normal index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UIRABN: Starting RABN for Upper Index

UIRABN is the starting RABN for the upper index. If this parameter is omitted, ADAORD assigns the starting RABN.

UIRELEASE: Release Unused Upper Index Blocks

Specifying UIRELEASE releases unused upper index (UI) blocks belonging to the specified file. If UIRELEASE is not specified, ADAORD allocates *at least* the number of UI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

UISIZE: Upper Index Size

UISIZE specifies the number of blocks or cylinders to be allocated for the file's upper index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

Examples

Example 1:

```
ADAORD   REORFASSO   FILE=9
ADAORD                          ASSOPFAC=5
ADAORD                          FILE=23
ADAORD                          UIRABN=3151,UISIZE=50B
ADAORD                          NIRABN=3201
```

The Associator for files 9 and 23 is to be reordered; Associator data for other files is not changed.

The Associator padding factor is set to 5% for file 9. For file 23, the following Associator changes are being made: the new upper index starting RABN is 3151, with a new upper index size of 50 blocks. The new normal index starting RABN is 3201; the normal index size remains the same as before.

Example 2:

```
ADAORD   REORFASSO   FILE=104
ADAORD                          ASSOPFAC=5,NISIZE=5B,UISIZE=2B
ADAORD                          ACRABN=10000,NIRABN=10510,UIRABN=10515
ADAORD                          FILE=105
```

The Associator for files 104 and 105 is to be reordered; Associator information for all other files is unchanged.

For file 104, the Associator padding factor is to be set to 5. The sizes of the normal index and upper index are to be 5 blocks and 2 blocks respectively. The starting RABN for the address converter is to be 10000. The starting RABN for the normal index is to be 10510. The starting RABN for the upper index is to be 10515. Information for file 105 is reordered according to the file's existing definition.

112 REORFDATA: Reorder Data Storage for a Single File

- Essential Parameter 632
- Optional Parameters 633
- Examples 636

The REORFDATA function reorders Data Storage for a single file. Data Storage for unspecified files is not reordered.

The REORFDATA function does not reorder ADAM files. However, it can be used to relocate an ADAM file to different RABNs.

This function requires exclusive EXU control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

This is the syntax of the ADAORD REORFDATA function:

```
ADAORD REORFDATA FILE = file-number
                        [ALLOCATION = { FORCE | NOFORCE } ]
                        [DATAPFAC =  padding-factor  ]
                        [DATAVOLUME = 'Data-Storage-extent-volume' ]
                        [DSDEV = device-type ]
                        [DSRABN = starting-rabn]
                        [DSRELEASE]
                        [DSSIZE =size ]
                        [MAXRECL = record-length ]
                        [PASSWORD = password ]
                        [SORTSEQ = { descriptor | ISN | physical-sequence } ]
                    [EXCLUDE = file-list ]
                    [LIP = { isn-pool-size  | 16384 } ]
                    [LPB = { prefetch-buffer-size  | ADARUN-lu } ]
                    [NOUSERABEND]
                    [TEST]
```

Essential Parameter

FILE: File Number

FILE specifies the file to be processed, and to which the parameters that follow in the statement sequence apply. Several files and their related parameters may be specified within one REORFDATA operation; see the examples at the end of this section.

Optional Parameters

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameter DSRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameter.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameter fails, the utility retries the allocation without the placement parameter.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor, which is the percentage of each Data Storage block reserved for record expansion when the file is reordered. A value in the range 1-90 may be specified (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). If this parameter is omitted, the current padding factor for the file is used.

DATAVOLUME: Data Storage Extent Volume

Note: The value for the DATAVOLUME parameter must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If DATAVOLUME is not specified, the Data Storage space is allocated according to ADAORD's default allocation rules.

DSDEV: Data Storage Device Type

DSDEV is the file's Data Storage device type. The specified device type must already be defined to Adabas, normally when the database was created or by the ADADBS utility's ADD function.

If DSDEV is not specified, ADAORD attempts to allocate the file on the device type used before reordering.

DSRABN: Data Storage Starting RABN

DSRABN is the beginning RABN for the file's Data Storage extent. If this parameter is omitted, ADAORD assigns the starting RABN.

DSRELEASE: Release Unused Data Storage Blocks

Specifying DSRELEASE releases unused Data Storage (DS) blocks belonging to the file. If DSRELEASE is not specified, ADAORD allocates *at least* the number of DS blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

DSSIZE: Data Storage Size

DSSIZE specifies the number of blocks or cylinders to be allocated for the Data Storage. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the DATAPFAC padding factor.

EXCLUDE: Exclude Specified Files from Reorder

EXCLUDE lists the numbers of the files to be excluded from REORDER processing; that is, the files that are not to be reordered.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

Files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

LIP: ISN Buffer Pool Size

The LIP parameter can be used to decrease the number of Associator I/O operations when re-creating the address converter. For best performance, specify a size that accepts all ISNs of the largest file to be processed.

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 16384 bytes (or 16K).

The length of one input record is ISNSIZE + RABNSIZE. Thus the entry length is at least 6 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 8 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

Note: When ADAORD is processing files that contain spanned records with secondary ISNs, a second LIP will be allocated to contain these ISNs.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32,760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXRECL: Maximum Compressed Record Length

Use the MAXRECL parameter to change the maximum record length, after compression, permitted in the file. Specifying MAXRECL has two effects:

- The DATA data set for the file can be allocated only to devices that support the specified length.
- If the file contains Data Storage records that exceed the specified length, ADAORD abends and prints the ERROR-126 message (Data Storage record too long).

If MAXRECL is not specified, the maximum compressed record length does not change.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

SORTSEQ: File Reordering Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables *cannot* be tested: only the syntax of the specified parameters can be tested.

Examples

Example 1:

```
ADAORD   REORFDATA   FILE=16
```

The Data Storage for file 16 is to be reordered. No other files are affected.

Example 2:

```
ADAORD   REORFDATA   FILE=246
ADAORD              DATAPFAC=5,DSSIZE=10,SORTSEQ=MZ
ADAORD              FILE=247
```

The Data Storage for files 246 and 247 is to be reordered. No other files' Data Storage will be re-ordered.

For file 246, the Data Storage padding factor is to be set to 5. Data Storage for file 247 is reordered according to the file's existing definition.

113 REORFILE: Reorder File

- Essential Parameter 639
- Optional Parameters 640
- Examples 645

The REORFILE function reorders the Associator and Data Storage for a single file. Associator and Data Storage for other files are not affected.

This function requires exclusive EXU control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

This is the syntax of the ADAORD REORFILE function:

```
ADAORD REORFILE FILE = file-number
                       [ACRABN = starting-rabn ]
                       [AC2RABN = starting-rabn ]
                       [ALLOCATION = { FORCE | NOFORCE } ]
                       [ASSOPFAC = padding-factor  ]
                       [ASSOVOLUME = 'Associator-extent-volume ' ]
                       [DATAPFAC =  padding-factor  ]
                       [DATAVOLUME = 'Data-Storage-extent-volume' ]
                       [DSDEV = device-type ]
                       [DSRABN = starting-rabn]
                       [DSRELEASE]
                       [DSSIZE =size ]
                       [INDEXCOMPRESSION = { YES | NO } ]
                       [ISNSIZE = { 3 | 4 } ]
                       [MAXISN = highest-isn ]
                       [MAXISN2 = highest-isn ]
                       [MAXRECL = record-length ]
                       [NIRABN = starting-rabn]
                       [NIRELEASE]
                       [NISIZE = size ]
                       [PASSWORD = password ]
                       [SORTSEQ = { descriptor | ISN | physical-sequence } ]
                       [UIRABN = starting-rabn ]
                       [UIRELEASE]
                       [UISIZE = size ]
                     [EXCLUDE = file-list ]
                     [LIP = { isn-pool-size | 16384 } ]
                     [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                     [NOUSERABEND]
                     [TEST]
```

Essential Parameter

FILE: File Number

FILE specifies the file to be processed, and to which the parameters that follow in the statement sequence apply. Several files and their related parameters may be specified within one REORFILE operation; see the examples at the end of this section. If a component file of an Adabas expanded file is specified, only that file's Associator and Data Storage are reordered; this has no adverse effect on the other component files.

Optional Parameters

ACRABN: Starting RABN for Address Converter

ACRABN is the beginning RABN for the file's address converter extent. If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

AC2RABN: Starting RABN for Secondary Address Converter

The beginning RABN for the file's secondary address converter extent. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent. If the file contains no secondary address converter extents, this parameter is ignored.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOPFAC: Associator Padding Factor

ASSOPFAC specifies the new Associator block padding factor. The number specified represents the percentage of each Associator block not to be used during the reorder process. A value in the range 1-90 may be specified. The remaining number of bytes after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current Associator padding factor in effect for the file is used.

ASSOVOLUME: Associator Extent Volume

Note: The value for the ASSOVOLUME parameter must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) should be allocated. If the requested number of blocks cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If ACRABN, UIRABN, or NIRABN is specified, ADAORD ignores the ASSOVOLUME value when allocating the corresponding extent type.

If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADAORD's default allocation rules.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor, which is the percentage of each Data Storage block reserved for record expansion when the file is reordered. A value in the range 1-90 may be specified (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). If this parameter is omitted, the current padding factor for the file is used.

DATAVOLUME: Data Storage Extent Volume

Note: The value for the DATAVOLUME parameter must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If DATAVOLUME is not specified, the Data Storage space is allocated according to ADAORD's default allocation rules.

DSDEV: Data Storage Device Type

DSDEV specifies the device type to be used for the file's Data Storage. The specified device type must already be defined to Adabas, normally when the database was created or by the ADADBS utility's ADD function.

If this parameter is not specified, ADAORD attempts to allocate the file on the device type used before reordering.

DSRABN: Data Storage Starting RABN

DSRABN specifies the beginning RABN for the file's Data Storage extent. If the DSRABN parameter is omitted, ADAORD assigns the starting RABN.

DSRELEASE: Release Unused Data Storage Blocks

Specifying DSRELEASE releases unused Data Storage (DS) blocks belonging to the file. If DSRELEASE is not specified, ADAORD allocates *at least* the number of DS blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

DSSIZE: Data Storage Size

DSSIZE specifies the number of blocks or cylinders to be allocated for the Data Storage. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD calculates the file extent size in proportion to any increase or decrease in the DATAPFAC padding factor.

EXCLUDE: Exclude Specified Files from Reorder

EXCLUDE lists the numbers of the files to be excluded from REORDER processing; that is, the files that are not to be reordered.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

Files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index for the file is rebuilt in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is not specified, the default is the current form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LIP: ISN Buffer Pool Size

The LIP parameter can be used to decrease the number of Associator I/O operations when re-creating the address converter. For best performance, specify a size that accepts all ISNs of the largest file to be processed.

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 16384 bytes (or 16K).

The length of one input record is ISNSIZE + RABNSIZE. Thus the entry length is at least 6 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 8 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

Note: When ADAORD is processing files that contain spanned records with secondary ISNs, a second LIP will be allocated to contain these ISNs.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

MAXISN: Highest ISN Permitted for the File

MAXISN is the highest ISN which may be allocated for the file. This value must be greater than the current TOPISN value displayed in the ADAREP database report.

ADAORD uses the specified value to calculate the address converter space required. If this parameter is omitted, the current MAXISN value for the file is retained.

MAXISN2: Highest Secondary ISN Permitted for the File

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. This value must be greater than the current TOP AC2 ISN value displayed in the ADAREP database report. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADAORD uses the specified value to calculate the space required in the secondary address converter for the file. If this parameter is omitted, the current MAXISN2 value for the file is retained. If the file contains no secondary address converter extents, this parameter is ignored.

MAXRECL: Maximum Compressed Record Length

Use the MAXRECL parameter to change the maximum record length, after compression, permitted in the file. Specifying MAXRECL has two effects:

- The DATA data set for the file can be allocated only to devices that support the specified length.
- If the file contains Data Storage records that exceed the specified length, ADAORD abends and prints ERROR-126 (Data Storage record too long).

If MAXRECL is not specified, the maximum compressed record length does not change.

NIRABN: Starting RABN for Normal Index

NIRABN is the beginning RABN for the normal index extent. If this parameter is omitted, ADAORD assigns the starting RABN.

NIRELEASE: Release Unused Normal Index Blocks

Specifying NIRELEASE releases unused normal index (NI) blocks belonging to the file. If NIRELEASE is not specified, ADAORD allocates *at least* the number of NI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

NISIZE: Normal Index Size

NISIZE specifies the number of blocks or cylinders to be allocated for the file's normal index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

SORTSEQ: Reorder Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UIRABN: Starting RABN for Upper Index

UIRABN is the beginning RABN for the file's upper index extent. If this parameter is omitted, ADAORD assigns the starting RABN.

UIRELEASE: Release Unused Upper Index Blocks

Specifying UIRELEASE releases unused upper index (UI) blocks belonging to the file. If UIRELEASE is not specified, ADAORD allocates *at least* the number of UI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

UISIZE: Index Size for Upper Index

UISIZE specifies the number of blocks or cylinders to be allocated for the upper index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size in proportion to any increase or decrease in the ASSOPFAC padding factor.

Examples

Example 1:

```
ADAORD   REORFILE FILE=16
```

Associator and Data Storage for file 16 are to be reordered. All current extent sizes and padding factors for the file are to be retained. No other files are reordered.

Example 2:

```
ADAORD   REORFILE    FILE=246
ADAORD              DATAPFAC=5,DSSIZE=10B,SORTSEQ=MZ
ADAORD              ASSOPFAC=20,MAXISN=5000
ADAORD              FILE=20
```

File 246 is to be reordered; the Associator and Data Storage for all other files remain unchanged. The Data Storage padding factor is to be set to 5. A new Data Storage size of 10 blocks is to be used. The Data Storage is to be reordered in the logical sequence of descriptor MZ. The new Associator padding factor is 20. The highest ISN which may be assigned is 5000. File 20 is to be reordered with no changes to its current parameters.

Example 3:

```
ADAORD   REORFILE    FILE=9
ADAORD              ASSOPFAC=5,DATAPFAC=15
ADAORD              FILE=23
ADAORD              DSRABN=24032
ADAORD              UIRABN=3151,UISIZE=50B
ADAORD              NIRABN=3201
```

File 9 is to be reordered. The Associator padding factor is set to 5% and the Data Storage padding factor to 15%.

File 23 will also be reordered, with a new starting Data Storage RABN of 24032. In addition, the following Associator changes are being made for file 23: the new upper index starting RABN is 3151, with a new upper index size of 50 blocks. The new normal index starting RABN is 3201; the size remains the same.

114 RESTRUCTUREDB: Restructure Database

- Optional Parameters and Subparameters 649
- Examples 799

The RESTRUCTUREDB function unloads an entire database to a sequential data set, which can be used as input to the STORE function to load the data into a new database. The target database may be located on a physical device type different from the source database. The Associator and Data Storage are reordered as part of RESTRUCTURE/STORE processing.

This function requires exclusive EXF control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

When the RESTRUCTUREDB function restructures an ADAM file that uses the overflow area, and then the STORE function stores the restructured file in a database with a smaller DATA block size, an ADAORD ERROR-103 may occur. Use the ADAULD and ADALOD utilities to move ADAM files, instead.

Note: You can restructure databases and files from an Adabas version prior to Adabas 8 and store them in an Adabas 8 database using ADAORD STORE. However, you cannot store the restructured output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7). If you attempt this, the following warning will be generated and ADAORD will end with a CC=4:

```
*** Warning: The input data set is from V8 and will not be processed
```

This is the syntax of the ADAORD RESTRUCTUREDB function:

```
ADAORD  { RESTRUCTUREDB | REDB }
                [DBASSODEV = {device-type  | ADARUN-device} ]
                [DBDATADEV = {device-type  | ADARUN-device} ]
                [DBINDEXCOMPRESSION = { YES | NO } ]
                [FILE = file-number ]
                    [ASSOPFAC = padding-factor  ]
                    [ASSODEV = {device-type  | DBASSODEV-value } ]
                    [DATADEV = {device-type | DBDATADEV-value } ]
                    [DATAPFAC =  padding-factor  ]
                    [INDEXCOMPRESSION = { YES | NO } ]
                    [ISNSIZE = { 3 | 4 } ]
                    [SORTSEQ = {descriptor | ISN | physical-sequence } ]
                [LPB = { prefetch-buffer-size  | ADARUN-lu  } ]
                [NOUSERABEND]
                [TEST]
```

Optional Parameters and Subparameters

ASSODEV: Associator Device Type

ASSODEV specifies the device type to be used in the new database for the file's ASSO data set. This parameter is required only when the device type to be used is different from the default device type. The default device type is specified by the DBASSODEV parameter; if DBASSODEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter. These parameters have no effect on the data written to the DDFILEA/ FILEA data set.

ASSOPFAC: Associator Padding Factor

ADAORD uses the ASSOPFAC value to calculate the space required to perform the STORE function for the specified file. Valid values are 1-90. The number of AC, NI, and UI blocks is calculated for the device type specified by ASSODEV and the padding factor specified by ASSOPFAC. These parameters have no effect on the data written to DDFILEA. If ASSOPFAC is not specified, the current padding factor for the file is used.

DATADEV: Data Storage Device Type

DATADEV specifies the device type to be used for the specified file's new DATA data set. This parameter is required only when the device type to be used is different from the default device type. The default device type is specified by the DBDATADEV parameter; if DBDATADEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter. These parameters have no effect on the data written to DDFILEA.

DATAPFAC: Data Storage Padding Factor

ADAORD uses DATAPFAC to calculate the space required to perform the STORE function for the specified file. Valid values are 1-90 (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). The number of Data Storage blocks is calculated for the device type specified by DATADEV and the padding factor specified by DATAPFAC. If DATAPFAC is not specified, the current padding factor for the file is used. These parameters have no effect on the data written to DDFILEA.

DBASSODEV: Default Associator Device Type

DBASSODEV specifies a default device type for the new ASSO data set. ADAORD uses the device type specified here to calculate the ASSO space requirements for each restructured file. If DBASSODEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter.

To override the default device type for a file, use the FILE and ASSODEV parameters. The DBASSODEV parameter has no effect on the data written to the DDFILEA/ FILEA data set.

DBDATADEV: Default Data Storage Device Type

DBDATADEV specifies a default device type for the new DATA data set. ADAORD uses the device type specified here to calculate the DATA space requirements for each restructured file. If DBDATADEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter.

To override the default device type for a file, use the FILE and DATADEV parameters. The DATADEV parameter has no effect on the data written to the DDFILEA/ FILEA data set.

DBINDEXCOMPRESSION: Calculate Index Sizes for Database

DBINDEXCOMPRESSION indicates for all files whether the index space calculation performed and displayed by ADAORD is based on compressed or uncompressed indexes. It applies to all files for which no INDEXCOMPRESSION parameter is specified.

DBINDEXCOMPRESSION can be used to calculate the sizes of compressed or uncompressed indexes for all files of the database, making it unnecessary to calculate the sizes for each file.

FILE: File Number

FILE specifies the file to which the following parameters apply. The records for all files not specified by this parameter are unloaded in physical sequence, by file.

If an Adabas checkpoint or security file is specified, do not specify the SORTSEQ parameter.

INDEXCOMPRESSION: Calculate Index Sizes for File

INDEXCOMPRESSION indicates for its associated file whether the index space calculation performed and displayed by ADAORD is based on a compressed or uncompressed index.

If INDEXCOMPRESSION is *not* specified

- but the DBINDEXCOMPRESSION parameter is specified for the database as a whole, the default is the database value.

- and DBINDEXCOMPRESSION is also *not* specified, the default is the current compression form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

SORTSEQ: File Processing Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADAORD RESTRUCTUREDB
```

The RESTRUCTUREDB function is to be executed. All files are to be unloaded in physical sequence.

Example 2:

```
ADAORD RESTRUCTUREDB FILE=146,SORTSEQ=MZ
ADAORD               FILE=151,SORTSEQ=TF
```

The RESTRUCTUREDB function is to be executed. File 146 is to be unloaded in the sequence of descriptor MZ. File 151 is to be unloaded in the sequence of descriptor TF. All other files are to be unloaded in physical sequence.

115 RESTRUCTUREF: Restructure Single Files

- Essential Parameter 655
- Optional Parameters 655
- Examples 658

The RESTRUCTUREF function unloads one or more files to a sequential data set, which can be used as input to the STORE function to load the files into another database. The target database may be located on a physical device type different from the originating database. The Associator and Data Storage are reordered as part of RESTRUCTURE/STORE processing.

RESTRUCTUREF requires EXU control; other users may access database files being used by RESTRUCTUREF, but only for reading. Note, however, that operations involving either the checkpoint or security files require exclusive database control. In addition, parts of the database are overwritten during ADAORD execution. We therefore recommend that you back up the database (or file) using the ADASAV utility before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this reorder function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

When the RESTRUCTUREF function restructures an ADAM file that uses the overflow area, and then the STORE function stores the restructured file in a database with a smaller DATA block size, an ADAORD ERROR-103 may occur. Use the ADAULD and ADALOD utilities to move ADAM files, instead.

Note: You can restructure databases and files from an Adabas version prior to Adabas 8 and store them in an Adabas 8 database using ADAORD STORE. However, you cannot store the restructured output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7). If you attempt this, the following warning will be generated and ADAORD will end with a CC=4:

```
*** Warning: The input data set is from V8 and will not be processed
```

This is the syntax of the ADAORD RESTRUCTUREF function:

```
ADAORD { RESTRUCTUREF | REF }
                FILE = file-number
                    [ASSODEV = {device-type | DBASSODEV-value } ]
                    [ASSOPFAC = padding-factor ]
                    [DATADEV = {device-type | DBDATADEV-value } ]
                    [DATAPFAC = padding-factor ]
                    [DBASSODEV = {device-type | ADARUN-device} ]
                    [DBDATADEV = {device-type | ADARUN-device} ]
                    [INDEXCOMPRESSION = { YES | NO } ]
                    [ISNSIZE = { 3 | 4 } ]
                    [PASSWORD = password ]
                    [SORTSEQ = {descriptor | ISN | physical-sequence } ]
                [LPB = { prefetch-buffer-size | ADARUN-lu } ]
                [NOUSERABEND]
                [TEST]
```

Essential Parameter

FILE: File Number

FILE specifies the file to be restructured. A separate ADAORD FILE statement must be provided for each file to be processed, followed by ADAORD statements containing the relevant parameters for that file.

If you specify a file that is either coupled or part of an expanded file, the related files are automatically added to the file list. A message indicating the files added appears in DDPRINT.

Optional Parameters

ASSODEV: Associator Device Type

ASSODEV specifies the device type to be used for the specified file's new ASSO data set. This parameter is required only when the device type to be used is different from the default device type. The default device type is specified by the DBASSODEV parameter; if DBASSODEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter. These parameters have no effect on the data written to the DDFILEA/ FILEA data set.

ASSOPFAC: Associator Padding Factor

ADAORD uses ASSOPFAC to calculate the space required to perform the STORE function for the specified file. Valid values are 1-90. The number of AC, NI, and UI blocks is calculated for

the device type specified by ASSODEV and the padding factor specified by ASSOPFAC. If ASSOPFAC is not specified, the current padding factor for the file is used. These parameters have no effect on the data written to DDFILEA.

DATADEV: Data Storage Device Type

DATADEV specifies the device type to be used for the specified file's new DATA data set. This parameter is required only when the device type to be used is different from the default device type. The default device type is specified by the DBDATADEV parameter; if DBDATADEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter. These parameters have no effect on the data written to DDFILEA.

DATAPFAC: Data Storage Padding Factor

ADAORD uses DATAPFAC to calculate the space required to perform the STORE function for the specified file. Valid values are 1-90 (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). The number of Data Storage blocks is calculated for the device type specified by DATADEV and the padding factor specified by DATAPFAC. If DATAPFAC is not specified, the current padding factor for the file is used. These parameters have no effect on the data written to DDFILEA.

DBASSODEV: Default Associator Device Type

DBASSODEV specifies a default device type for the new ASSO data set. ADAORD uses the device type specified here to calculate the ASSO space requirements for each restructured file. If DBASSODEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter.

To override the default device type for a file, use the FILE and ASSODEV parameters. The DBASSODEV parameter has no effect on the data written to the DDFILEA/ FILEA data set.

DBDATADEV: Default Data Storage Device Type

DBDATADEV specifies a default device type for the new DATA data set. ADAORD uses the device type specified here to calculate the DATA space requirements for each restructured file. If DBDATADEV is not specified, the default is the device type specified by the ADARUN DEVICE parameter.

To override the default device type for a file, use the FILE and DATADEV parameters. The DBDATADEV parameter has no effect on the data written to the DDFILEA/ FILEA data set.

INDEXCOMPRESSION: Calculate Index Sizes for File

INDEXCOMPRESSION indicates for its associated file whether the index space calculation performed and displayed by ADAORD is based on a compressed or uncompressed index.

If INDEXCOMPRESSION is *not* specified

- but a compression value is specified for the database as a whole, the default is the database value.
- and *no* compression value is specified for the database, the default is the current compression form of the file.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum size is 32,760 bytes. The default depends on the ADARUN LU parameter. ADAORD may reduce a specified LPB value if the LU value is too small.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message v is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

If the file is password-protected, use this parameter to specify the password.

SORTSEQ: File Processing Sequence

SORTSEQ determines the sequence in which the file is to be processed. If this parameter is omitted, the records are processed in physical sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. *Do not* use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not that the validity of values and variables.

Examples

Example 1:

```
ADAORD RESTRUCTUREF FILE=15                              ↩
```

The RESTRUCTUREF function is to be executed. File 15 is to be unloaded in physical sequence. No other files are to be unloaded.

Example 2:

```
ADAORD RESTRUCTUREF  FILE=25,SORTSEQ=KL
ADAORD               FILE=40,SORTSEQ=J3
```

The RESTRUCTUREF function is to be executed. Files 25 and 40 are to be unloaded. File 25 is to be unloaded in the sequence of descriptor KL. File 40 is to be unloaded in the sequence of descriptor J3. No other files are to be unloaded.

116 STORE: Store Files

- Optional Parameters and Subparameters 662
- Examples 668

The STORE function loads one or more files into an existing database using output produced by the RESTRUCTURE functions. The Associator and Data Storage are reordered as part of RESTRUCTURE/STORE processing.

If the ALLFILES parameter is specified, all files contained on the input data set are stored. If ALLFILES is not specified, only those files specified by FILE parameters are stored.

One or more files may be specified with FILE parameter statements, even when ALLFILES is also specified. The STORE function loads each file specified with a FILE statement according to the definition contained in any subparameters immediately following that file's FILE statement. All other files are loaded according to their existing definitions.

If existing files in the database are to be overwritten, the OVERWRITE parameter must be supplied.

This function requires exclusive EXF control of the database files involved in the operation. In addition, parts of the database are overwritten during ADAORD execution, so we recommend that you back up the database (or file) using the ADASAV utility first, before running ADAORD functions.

If the file specified for this function was originally loaded with ISNREUSE=YES active, this store function will reset the first unused ISN value in that file's control block (FCB) to the actual first unused ISN found in the address converter.

Notes:

1. The STORE function does not reorder ADAM files. However, it, in conjunction with other ADAORD functions, can be used to relocate an ADAM file to different RABNs.
2. Storing restructured ADAM files on a device with a smaller DATA blocking factor than before can result in utility ERROR 103 if the ADAM file previously used the overflow area. To relocate an ADAM file to a different device, use the ADAULD and ADALOD utilities.
3. Checkpoint and security files from Adabas version 5.1 or 5.2 cannot be stored due to internal structure changes to the files in version 5.3.
4. If ALLFILES is requested in the STORE function, be sure that the Associator (ASSO) data set is large enough to accommodate the all the files. Use the calculation `(file-count * 3) + 110` to verify this. Otherwise, an ERROR-068 is possible.
5. You can restructure databases and files from an Adabas version prior to Adabas 8 and store them in an Adabas 8 database using ADAORD STORE. However, you cannot store the restructured output of an Adabas 8 database or file in a database running with any prior Adabas version (for example, Adabas 7). If you attempt this, the following warning will be generated and ADAORD will end with a CC=4: `*** Warning: The input data set is from V8 and will not be processed`

This is the syntax of the ADAORD STORE function:

```
ADAORD STORE [ALLFILES]
             [CHECKPOINT]
             [FILE = file-number ]
                 [ACRABN = starting-rabn ]
                 [AC2RABN = starting-rabn ]
                 [ALLOCATION = { FORCE | NOFORCE } ]
                 [ASSOPFAC = padding-factor  ]
                 [ASSOVOLUME = 'Associator-extent-volume ' ]
                 [DATAPFAC =  padding-factor  ]
                 [DATAVOLUME = 'Data-Storage-extent-volume' ]
                 [DSDEV = device-type ]
                 [DSRABN = starting-rabn]
                 [DSRELEASE]
                 [DSSIZE =size ]
                 [INDEXCOMPRESSION = { YES | NO } ]
                 [ISNSIZE = { 3 | 4 } ]
                 [MAXISN = highest-isn ]
                 [MAXISN2 = highest-isn ]
                 [MAXRECL = record-length ]
                 [NIRABN = starting-rabn]
                 [NIRELEASE]
                 [NISIZE =  size ]
                 [UIRABN = starting-rabn ]
                 [UIRELEASE]
                 [UISIZE =  size ]
             [EXCLUDE = file-list ]
             [LIP = { isn-pool-size  |  16384 } ]
             [NOUSERABEND]
             [OVERWRITE]
             [REPLICATOR]
             [SECURITY]
             [SLOG]
             [TEST]
             [TRIGGER]
```

Optional Parameters and Subparameters

ACRABN: Starting RABN for Address Converter

ACRABN specifies the beginning RABN for the file's address converter extent. If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent.

When specifying the starting RABN for Associator extents, the space needed for the FCBs, FDTs, and DSST should also be considered.

AC2RABN: Starting RABN for Secondary Address Converter

The beginning RABN for the file's secondary address converter extent. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

If this parameter is omitted, ADAORD assigns the starting RABN. The space requested must be available in one extent. If the file contains no secondary address converter extents, this parameter is ignored.

ALLFILES: Select All Files for Storing

ALLFILES causes all files in the input data set to be stored in the database. If ALLFILES is not supplied, only those files are stored for which FILE parameters have been specified.

If the input data set contains files that are coupled or part of an expanded file and the related files are not in the data set, ERROR-138 is returned indicating an inconsistent file list. You must add the related files before the STORE function will execute successfully.

Note: If ALLFILES is requested in the STORE function, be sure that the Associator (ASSO) data set is large enough to accommodate the all the files. Use the calculation `(file-count * 3) + 110` to verify this. Otherwise, an ERROR-068 is possible.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOPFAC: Associator Padding Factor

ASSOPFAC specifies the new Associator block padding factor. The number specified represents the percentage of each Associator block not to be used during the reorder process. A value in the range 1-90 may be specified. The remaining number of bytes after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current Associator padding factor in effect for the file is used.

ASSOVOLUME: Associator Extent Volume

Note: The value for the ASSOVOLUME parameter must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which to allocate the file's Associator space (the AC, NI, and UI extents). If the requested number of blocks cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If ACRABN, UIRABN, or NIRABN is specified, ADAORD ignores the ASSOVOLUME value when allocating the corresponding extent type.

If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADAORD's default allocation rules.

CHECKPOINT: Store the Checkpoint File

If either ALLFILES is specified or the FILE parameter specifies the checkpoint file, CHECKPOINT stores the checkpoint file from the DDFILEA/FILEA tape in the database, making that file the new checkpoint file. The new checkpoint file must have the same file number as the old checkpoint file.

If the CHECKPOINT parameter is not specified, the checkpoint file on the FILEA/DDFILEA tape is not stored in the database, even though the checkpoint file was specified by a FILE parameter or the ALLFILES parameter was specified.

DATAPFAC: Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor, which is the percentage of each Data Storage block reserved for record expansion when the file is reordered. A value in the range 1-90 may be specified (see the ADALOD LOAD DATAPFAC parameter discussion for more information about setting the padding factor). If this parameter is omitted, the current padding factor for the file is used.

DATAVOLUME: Data Storage Extent Volume

Note: The value of the DATAVOLUME parameter must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADAORD allocates the remaining blocks on other volumes according to its default allocation rules.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If DATAVOLUME is not specified, the Data Storage space is allocated according to ADAORD's default allocation rules.

DSDEV: Data Storage Device Type

DSDEV specifies the device type to be used for the file's Data Storage. The specified device type must already be defined to Adabas, normally when the database was created or by the ADADBS utility's ADD function.

If this parameter is not specified, ADAORD attempts to allocate the file on the device type used before restructuring.

DSRABN: Data Storage Starting RABN

The beginning RABN for the Data Storage extent for the specified file. If this parameter is omitted, ADAORD assigns the starting RABN.

DSRELEASE: Release Unused Data Storage Blocks

Specifying DSRELEASE releases unused Data Storage (DS) blocks belonging to the specified file. If DSRELEASE is not specified, ADAORD allocates *at least* the number of DS blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

DSSIZE: Data Storage Size

DSSIZE specifies the number of blocks or cylinders to be allocated for the Data Storage. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD will compute the file extent size (in blocks) in proportion to an increase or decrease in the DATAPFAC padding factor used.

EXCLUDE: Exclude Specified Files from Store

EXCLUDE lists the numbers of the files to be excluded from STORE processing; that is, the files that are not to be stored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

The EXCLUDE parameter may be specified only if ALLFILES is also specified.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

FILE: File Number

FILE specifies the file to be stored. A separate statement must be provided for each file to be processed, followed by ADAORD statements containing the relevant parameters for that file.

If you specify a file that is coupled or part of an expanded file and you do not also specify the related files, ERROR-138 is returned indicating an inconsistent file list. You must add the related files before the STORE function will execute successfully.

INDEXCOMPRESSION: Compress File Index

INDEXCOMPRESSION indicates whether the index for the file is rebuilt in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is not specified, the default is the form of the file at the time of the corresponding restructure operation.

ISNSIZE: 3- or 4-Byte ISN

ISNSIZE specifies whether ISNs in the file are to be 3 or 4 bytes long. The default is the value currently used for the file; this value is stored in the file control block (FCB).

Note: It is not possible to change the ISNSIZE of a physically coupled file using ADAORD.

LIP: ISN Buffer Pool Size

The LIP parameter can be used to decrease the number of Associator I/O operations when re-creating the address converter. For best performance, specify a size that accepts all ISNs of the largest file to be processed.

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value ("2048") or in kilobytes as a value followed by a "K" ("2K"). The default for LIP is 16384 bytes (or 16K).

The length of one input record is ISNSIZE + RABNSIZE. Thus the entry length is at least 6 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 8 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

Note: When ADAORD is processing files that contain spanned records with secondary ISNs, a second LIP will be allocated to contain these ISNs.

MAXISN: Highest ISN Permitted for the File

MAXISN specifies the highest ISN which may be allocated for the file. This value must be greater than the current TOPISN value displayed in the ADAREP database report.

ADAORD uses the specified value to calculate the address converter space required. If this parameter is omitted, the current MAXISN value for the file is retained.

MAXISN2: Highest Secondary ISN Permitted for the File

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. This value must be greater than the current TOP AC2 ISN value displayed in the ADAREP database report. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

ADAORD uses the specified value to calculate the space required in the secondary address converter for the file. If this parameter is omitted, the current MAXISN2 value for the file is retained. If the file contains no secondary address converter extents, this parameter is ignored.

MAXRECL: Maximum Compressed Record Length

Use the MAXRECL parameter to change the maximum record length, after compression, permitted in the file. Specifying MAXRECL has two effects:

- The DATA data set for the file can be allocated only to devices that support the specified length.
- If the file contains Data Storage records that exceed the specified length, ADAORD abends and prints ERROR-126 (Data Storage record too long).

If MAXRECL is not specified, there are two possibilities for the default value:

- If the maximum compressed record length before the file was restructured was the default ADALOD MAXRECL value, then DATA is allocated to a device arbitrarily, and the new maximum record length is derived from the device type;
- Otherwise, the maximum compressed record length does not change.

NIRABN: Starting RABN for Normal Index

The beginning RABN for the file's normal index extent. If this parameter is omitted, ADAORD assigns the starting RABN.

NIRELEASE: Release Unused Normal Index Blocks

Specifying NIRELEASE releases unused normal index (NI) blocks belonging to the file. If NIRELEASE is not specified, ADAORD allocates *at least* the number of NI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

NISIZE: Normal Index Size

The number of blocks or cylinders to be allocated for the normal index. The specified value cannot be larger than the largest single contiguous RABN area available; specifying blocks (a number of blocks followed by a "B") is therefore recommended.

If this parameter is omitted, ADAORD computes the file extent size (in blocks) in proportion to an increase or decrease in the ASSOPFAC padding factor used.

If this parameter is omitted and the INDEXCOMPRESSION parameter is specified, the ADAORD index size calculation does not consider the change in index size because ADAORD has no knowledge of the compression rate to be expected. Thus, ADAORD may allocate an index smaller than required causing secondary index extent allocations; or larger than necessary.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 034 (with a dump) or user abend 035 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing File

If a file to be stored already exists in the database, ADAORD terminates with an error message unless the OVERWRITE parameter has been specified.

REPLICATOR: Store the Replicator System File

Use this parameter to store the Replicator system file from the DDFILEA/FILEA tape as the new Replicator system file for the Event Replicator Server database. The new Replicator system file must have the same file number as the old Replicator system file. This parameter can only be specified if the database is an Event Replicator Server database.

When the `REPLICATOR` parameter is not specified, the Replicator system file on the DD-FILEA/FILEA tape is not stored in the database, even if it is specified by a `FILE` or `ALLFILES` parameter.

SECURITY: Store the Security File

SECURITY stores the security file from the DDFILEA/ FILEA tape, making that file the new security file for the database. The new file must have the same number as the old security file.

If SECURITY is omitted, the security file on the FILEA/ DDFILEA tape is not stored in the database, even if it is specified by a FILE parameter or the ALLFILES parameter is specified.

SLOG: Store the SLOG System File

Use this parameter to store the SLOG system file from the DDFILEA/FILEA tape as the new SLOG file for the Event Replicator Server database. The new SLOG file must have the same file number as the old SLOG file. This parameter can only be specified if the database is an Event Replicator Server database.

When the `SLOG` parameter is not specified, the SLOG system file on the DDFILEA/FILEA tape is not stored in the database, even if it is specified by a `FILE` or `ALLFILES` parameter.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TRIGGER: Store the Trigger File

Specify the TRIGGER parameter to store the trigger file from the DDFILEA/FILEA tape as the new trigger file for the database. The new trigger file must have the same file number as the old trigger file.

When the TRIGGER parameter is *not* specified, the trigger file on the DDFILEA/FILEA tape is *not* stored in the database, even if it is specified by the FILE or ALLFILES parameter.

UIRABN: Starting RABN for Upper Index

UIRABN specifies the beginning RABN for the upper index extent of the file. If this parameter is omitted, ADAORD assigns the starting RABN.

UIRELEASE: Release Unused Upper Index Blocks

Specifying UIRELEASE releases unused upper index (UI) blocks belonging to the file. If UIRELEASE is not specified, ADAORD allocates *at least* the number of UI blocks that were allocated before the file was reordered.

Note: Adabas calculates the file extent size using any changed padding factor or block size values *before* the file is reordered.

UISIZE: Upper Index Size

UISIZE specifies the number of blocks or cylinders to be allocated for the upper index. A block count must be followed by a "B" (for example, "2000B").

If this parameter is omitted, ADAORD computes the file extent size (in blocks) in proportion to an increase or decrease in the ASSOPFAC padding factor used.

If this parameter is omitted and the INDEXCOMPRESSION parameter is specified, the ADAORD index size calculation does not consider the change in index size because ADAORD has no knowledge of the compression rate to be expected. Thus, ADAORD may allocate an index smaller than required causing secondary index extent allocations; or larger than necessary.

Examples

Example 1:

```
ADAORD STORE FILE=14,OVERWRITE                          ↩
```

File 14, as unloaded by one of the RESTRUCTURE or the REORDB functions, is to be stored into an existing database. If the file already exists, it is deleted before being stored.

Example 2:

```
ADAORD   STORE   FILE=1,OVERWRITE
ADAORD           FILE=2,OVERWRITE
ADAORD           FILE=3,OVERWRITE
```

Files 1, 2 and 3 are written to the existing database. Old files 1, 2 and 3 are deleted.

Example 3:

```
ADAORD   STORE   OVERWRITE,ALLFILES
ADAORD           FILE=1,ACRABN=1000,NIRABN=2200
ADAORD           FILE=2,MAXISN=500000
ADAORD
FILE=4,ASSOPFAC=5,DATAPFAC=20,DSSIZE=5B,DSRABN=1
```

All files unloaded by the RESTRUCTURE function are to be stored into an existing database. The address converter for file 1 is to begin with RABN 1000. The normal index for file 1 is to begin with RABN 2200. The MAXISN for file 2 is to be set to 500,000. The following assignments are made for file 4: the Associator block padding factor is set to 5 percent; the Data Storage block padding factor is set to 20 per cent; a new DSSIZE of 5 cylinders is assigned starting at RABN 1.

All other files contained in the input data set are restored with their default values. If a file already exists, it is deleted before the new file is stored.

Example 4:

```
ADAORD STORE ALLFILES,CHECKPOINT
ADAORD EXCLUDE=20,10
```

All files from the input data set (including the checkpoint file) are stored. However, files 10 and 20 are excluded; that is, not stored.

Example 5:

```
ADAORD STORE ALLOCATION=NOFORCE
ADAORD FILE=10
ADAORD DSRABN=12345
```

File 10 is stored in the database. Its data storage is allocated beginning at RABN 12,345. If this allocation is not possible because the space is occupied by another file, ADAORD retries the allocation anywhere in the database's data storage.

117 JCL/JCS Requirements and Examples

- BS2000 672
- z/OS 677
- z/VM 680
- VSE 683

This section describes the job control information required to run ADAORD with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

Note: When running with the optional Recovery Aid (ADARAI) for RESTRUCTUREx or STORE functions, all temporary data sets must also be cataloged in the job control.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Intermediate storage	DDFILEA	tape/ disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the ADARAI option
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAORD parameters	SYS/DTA/DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAORD messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADAORD JCL Examples (BS2000)

Reorder File Data Storage, Reorder File, Reorder Data, Reorder Database

In SDF Format:

```
/.ADAORD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A O R D REORDER FILE DATA, REORDER FILE, REORDER DATA
 /REMARK * REORDER DATABASE
 /REMARK *
 /DELETE-FILE ADAyyyyy.FILEA
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.ORD.DATA
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
 /SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
```

```
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD REORDATA FILE=1,DSSIZE=80,DATAPFAC=30
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAORD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A O R D REORDER FILE DATA, REORDER FILE, REORDER DATA
 /REMARK * REORDER DATABASE
 /REMARK *
 /SYSFILE SYSLST=L.ORD.DATA
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
 /FILE ADAyyyyy.FILEA ,LINK=DDFILEA ,SPACE=(4800,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD REORDATA FILE=1,DSSIZE=80,DATAPFAC=30
 /LOGOFF NOSPOOL
```

Reorder Associator

In SDF Format:

```
/.ADAORD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A O R D REORDER FILE ASSO, REORDER ASSO
 /REMARK *
 /DELETE-FILE ADAyyyyy.FILEA
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.ORD.REOR
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD REORFASSO
 ADAORD FILE=1,MAXISN=20000,NISIZE=300B
 ADAORD FILE=3,NISIZE=400B,ASSOPFAC=2
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAORD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A O R D REORDER FILE ASSO, REORDER ASSO
 /REMARK *
 /SYSFILE SYSLST=L.ORD.REOR
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.FILEA ,LINK=DDFILEA ,SPACE=(4800,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD REORFASSO
 ADAORD FILE=1,MAXISN=20000,NISIZE=300B
 ADAORD FILE=3,NISIZE=400B,ASSOPFAC=2
 /LOGOFF NOSPOOL
```

Restructure

In SDF Format:

```
/.ADAORD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A O R D RESTRUCTURE
 /REMARK *
 /DELETE-FILE ADAyyyyy.FILEA
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.ORD.REST
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDRLOGR1,ADAyyyyy.RLOGR1,SHARE-UPD=YES
 /SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD RESTRUCTUREF
 ADAORD FILE=1,DATADEV=dddd
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAORD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A O R D RESTRUCTURE
 /REMARK *
 /SYSFILE SYSLST=L.ORD.REST
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.RLOGR1 ,LINK=DDRLOGR1,SHARUPD=YES
 /FILE ADAyyyyy.FILEA ,LINK=DDFILEA ,SPACE=(4800,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD RESTRUCTUREF
 ADAORD FILE=1,DATADEV=dddd
 /LOGOFF NOSPOOL
```

Store

Tape Example In SDF Format:

```
/.ADAORD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A O R D STORE
 /REMARK *
 /DELETE-FILE ADAyyyyy.FILEA
 /SET-JOB-STEP
 /REMARK Here, a tape has already been initialized in TSOS by
 /REMARK  /START-INIT
 /REMARK  INIT T-C4,VSN=ADA001,UNIT=M0 or M1
 /REMARK  END
 /CREATE-FILE ADAyyyyy.FILEA,SUP=TAPE(DEVICE=TAPE-C4,VOL=ADA001)
 /SET-JOB-STEP
 /ASS-SYSLST L.ORD.STOR
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDRLOGR1,ADAyyyyy.RLOGR1,SHARE-UPD=YES
 /REMARK Do not set the BUFFER-SIZE option for this file
 /SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
 /REMARK
 /REMARK  Here, using TAPEREL=NO, the tape will remain
 /REMARK  mounted This must be done from the console.
 /REMARK  if the tape is to be dismounted, omit this
 /REMARK  parameter
 /REMARK
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
```

```
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
 ADAORD STORE
 ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234
 ADAORD MAXISN=200000
 /LOGOFF SYS-OUTPUT=DEL
```

Tape Example In ISP Format:

```
/.ADAORD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A O R D STORE
 /REMARK *
 /SYSFILE SYSLST=L.ORD.STOR
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.RLOGR1 ,LINK=DDRLOGR1,SHARUPD=YES
 /REMARK Here, a tape has already been initialized in TSOS by
 /REMARK  /START-INIT
 /REMARK  INIT T-C4,VSN=ADA001,UNIT=M0 or M1
 /REMARK  END
 /REMARK Do not set the BLKSIZE option for this file
 /FILE ADAyyyyy.FILEA ,LINK=DDFILEA ,DEVICE=TAPE-C4,VOLUME=ADA001,LABEL=STD
 /REMARK
 /REMARK  Here, using TAPEREL=NO, the tape will remain
 /REMARK  mounted This must be done from the console.
 /REMARK  if the tape is to be dismounted, omit this
 /REMARK  parameter
 /REMARK
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
 ADAORD STORE
 ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234
 ADAORD MAXISN=200000
 /LOGOFF NOSPOOL
```

Non-Tape Example In SDF Format:

```

/.ADAORD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A O R D STORE
 /REMARK *
 /DELETE-FILE ADAyyyyy.FILEA
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.ORD.STOR
 /ASS-SYSDTA *SYSCMD
```

```
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDRLOGR1,ADAyyyyy.RLOGR1,SHARE-UPD=YES
/SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
ADAORD STORE
ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234
ADAORD MAXISN=200000
/LOGOFF SYS-OUTPUT=DEL
```

Non-Tape Example In ISP Format:

```
/.ADAORD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A O R D STORE
 /REMARK *
 /SYSFILE SYSLST=L.ORD.STOR
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.RLOGR1 ,LINK=DDRLOGR1,SHARUPD=YES
 /FILE ADAyyyyy.FILEA ,LINK=DDFILEA ,SPACE=(4800,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAORD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAORD STORE
 ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234
 ADAORD MAXISN=200000
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	Not used for REORASSO or REORFASSO
Intermediate storage	DDFILEA	tape/ disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the ADARAI option
ADARUN parameters	DDCARD	reader	*Operations*
ADAORD parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAORD messages	DDDRUCK	printer	*Messages and Codes*

ADAORD JCL Examples (z/OS)

Reorder File Associator

```
//ADAORDA   JOB
//*
//*     ADAORD: REORDER FILE ASSO,
//*              REORDER ASSO
//*
//ORD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDFILEA   DD   DSN=&&DDFILEA,DISP=(,PASS), <===INTERMEDIATE
//          UNIT=SYSDA,VOL=SER=vvvvvv,SPACE=(CYL,NN)          STORAGE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAORD REORFASSO
ADAORD FILE=1,ASSOPFAC=15,MAXISN=10000
/*
```

Refer to ADAORDA in the JOBS data set for this example.

Reorder File Data Storage, Reorder File, Reorder Data, Reorder Database

```
//ADAORDD   JOB
//*
//*     ADAORD: REORDER DATA STORAGE
//*
//ORD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDFILEA   DD   DSN=DDFILEA,                        <=== INTERMEDIATE
//           UNIT=TAPE,VOL=SER=vvvvvv,DISP=(,PASS)            FILE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X

//DDCARD    DD   *
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
```

```
ADAORD REORDATA
ADAORD FILE=1,DSSIZE=80,DATAPFAC=30
/*
```

Refer to ADAORDD in the JOBS data set for this example.

Restructure

```
//ADAORDR   JOB
//*
//*     ADAORD: RESTRUCTURE
//*
//ORD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <== DATA
//DDFILEA   DD   DSN=FILEA,                          <== INTERMEDIATE
//           UNIT=TAPE,VOL=SER=vvvvvv,DISP=(,KEEP)   <== FILE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAORD RESTRUCTUREF
ADAORD FILE=1,DATADEV=eeee
/*
```

Refer to ADAORDR in the JOBS data set for this example.

Store

```
//ADAORDS   JOB
//*
//*     ADAORD: STORE INTO A DIFFERENT DATABASE
//*              AFTER ADAORD RESTRUCTURE
//*

//ORD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDFILEA   DD   DSN=FILE1,                          <=== INTERMEDIATE
//               UNIT=TAPE,VOL=SER=vvvvvv,DISP=OLD             STORAGE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
```

```
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy   <=== DBID
/*
//DDKARTE   DD   *
ADAORD STORE
ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234,MAXISN=200000
/*
```

Refer to ADAORDS in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Intermediate storage	DDFILEA	tape/ disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the ADARAI option
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAORD parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAORD messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAORD JCL Examples (z/VM)

Reorder Associator

```
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDFILEA,DSN=ADAORD.FILEA,MODE=A
DATADEF DDPRINT,DSN=ADAORD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAORD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNORD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=REORASSO.CONTROL,MODE=A
ADARUN
```

Contents of RUNORD CONTROL A1

```
ADARUN PROG=ADAORD,DEVICE=dddd,DB=yyyyy
```

Contents of REORASSO CONTROL A1

```
ADAORD REORFASSO
ADAORD    FILE=1,MAXISN=20000,ASSOPFAC=20
ADAORD    FILE=3,NISIZE=400B,ASSOPFAC=20
```

Reorder Data Storage

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDFILEA,DSN=ADAORD.FILEA,MODE=A
DATADEF DDPRINT,DSN=ADAORD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAORD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNORD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=REORDATA.CONTROL,MODE=A
ADARUN
```

Contents of RUNORD CONTROL A1

```
ADARUN PROG=ADAORD,DEVICE=dddd,DB=yyyyy
```

Contents of REORDATA CONTROL A1

```
ADAORD REORDATA
ADAORD    FILE=1,DSSIZE=80,DATAPFAC=30
```

Restructure

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDFILEA,DSN=ADAORD.FILEA,MODE=A
DATADEF DDRLOGR1,DSN=ADABVv.RLOGR1,VOL=RLOGV1
DATADEF DDPRINT,DSN=ADAORD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAORD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNORD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=RESTRUCT.CONTROL,MODE=A
ADARUN
```

Contents of RUNORD CONTROL A1

```
ADARUN PROG=ADAORD,DEVICE=dddd,DB=yyyyy
```

Contents of RESTRUCT CONTROL A1

```
ADAORD RESTRUCTREF
ADAORD    FILE=1,DATADEV=eeee
```

Store

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDRLOGR1,DSN=ADABVv.RLOGR1,VOL=RLOGV1
DATADEF DDFILEA,DSN=ADAORD.FILEA,MODE=A
DATADEF DDPRINT,DSN=ADAORD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAORD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNORD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ORDSTORE.CONTROL,MODE=A
ADARUN
```

Contents of RUNORD CONTROL A1

```
ADARUN PROG=ADAORD,DEVICE=dddd,DB=yyyyy
```

Contents of ORDSTORE CONTROL A1

```
ADAORD STORE
ADAORD    FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234
ADAORD    MAXISN=200000
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
Intermediate Storage	FILEA	tape disk	SYS010 *	
Recovery log (RLOG)	RLOGR1	disk		Required when using the ADARAI option
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAORD parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADAORD messages		printer	SYS009	*Messages and Codes*

* *Any programmer logical unit may be used.*

ADAORD JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures (PROCs).

Refer to the following members for these examples:

Example	Member
Reorder File Associator	ADAORDA.X
Reorder File Data Storage	ADAORDD.X
Restructure	ADAORDR.X
Store Files	ADAORDS.X

Reorder File Associator

```
* $$ JOB JNM=ADAORDA,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAORDA
*       REORDER THE ASSOCIATOR.
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,DISK,VOL=DISK01,SHR
// DLBL FILEA,'ADABAS.ADAvrs.TEMP'
// EXTENT SYS010,DISK01,1,0,sssss,nnnnn
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAORD REORFASSO
ADAORD FILE=1,ASSOPFAC=15,MAXISN=10000
/*
/&
* $$ EOJ
```

Reorder File Data Storage, Reorder File, Reorder Data, Reorder Database

```
* $$ JOB JNM=ADAORDD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAORDD
*       REORDER DATA STORAGE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE,DO
// PAUSE MOUNT SCRATCH TAPE ON TAPE cuu
// MTC REW,SYS010
// MTC WTM,SYS010,5
// MTC REW,SYS010
// TLBL FILEA,'ADABAS.ADAvrs.TEMP'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAORD REORDATA
ADAORD FILE=1,DSSIZE=80,DATAPFAC=30
/*
/&
* $$ EOJ
```

Restructure

```
* $$ JOB JNM=ADAORDR,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAORDR
*       RESTRUCTURE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE,D0
// PAUSE MOUNT SCRATCH TAPE ON TAPE cuu
// MTC REW,SYS010
// MTC WTM,SYS010,5
// MTC REW,SYS010
// TLBL FILEA,'ADABAS.ADAvrs.TEMP'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAORD RESTRUCTUREF
ADAORD FILE=1,DATADEV=eeee
/*
/&
* $$ EOJ
```

Store Files

```
* $$ JOB JNM=ADAORDS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAORDS
*       STORE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE,D0
// PAUSE MOUNT SCRATCH TAPE ON TAPE cuu
// MTC REW,SYS010
// MTC WTM,SYS010,5
// MTC REW,SYS010
// TLBL FILEA,'ADABAS.ADAvrs.TEMP'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAORD,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAORD STORE
ADAORD FILE=1,DSSIZE=80,DATAPFAC=30,DSRABN=1234,MAXISN=200000
/*
/&
* $$ EOJ
```

XIV ADAPLP Utility: Print Data Protection Records from PLOG/Work

This chapter covers the following topics:

Functional Overview
ADAPLP Syntax and Examples
JCL/JCS Requirements and Examples

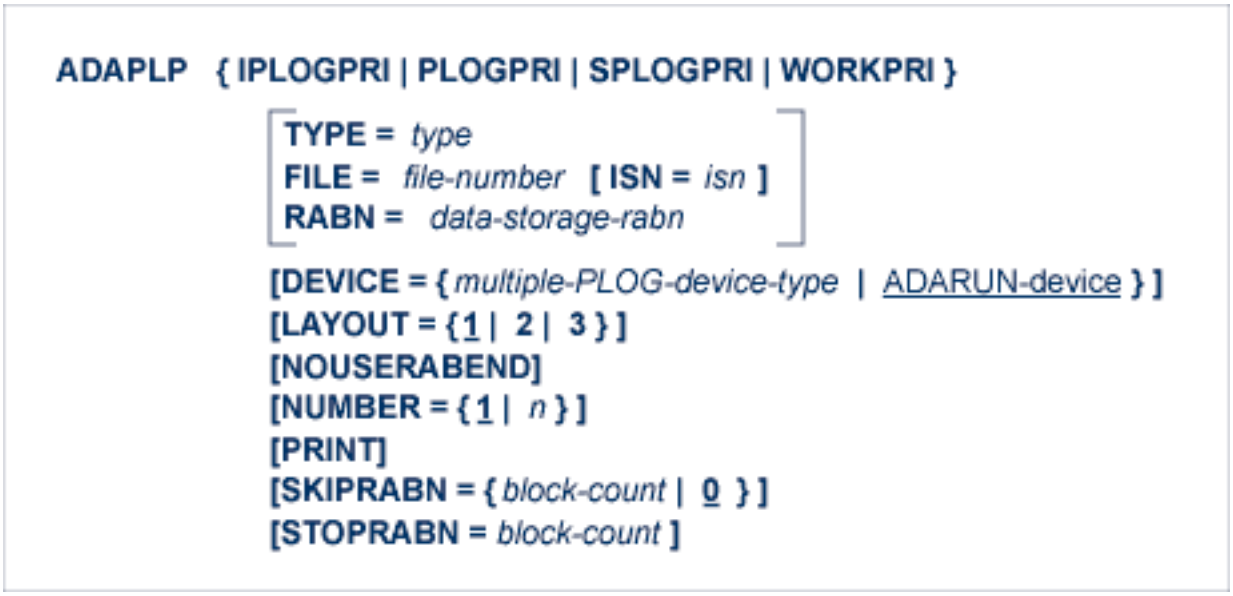
118 Functional Overview

The ADAPLP utility prints data protection records contained on the Adabas Work data set or the Adabas data protection log.

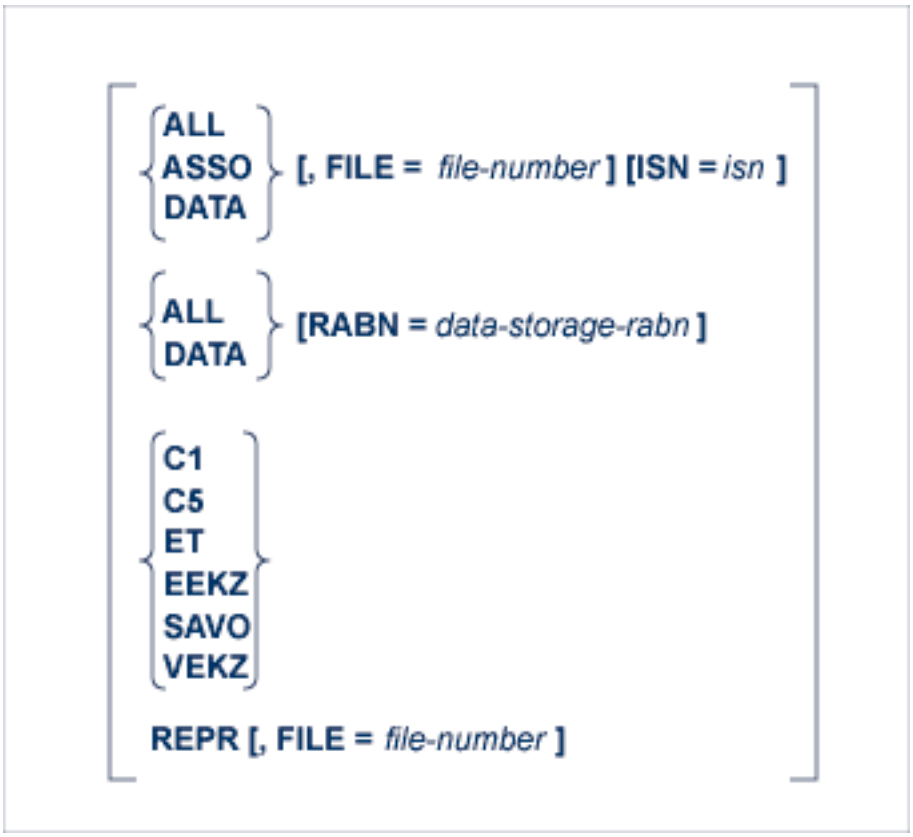
119 ADAPLP Syntax and Examples

- Optional Parameters and Subparameters 693
- Examples 696

The following diagram shows the ADAPLP syntax for specifying sequential intermediate (IPLOGPRI), multiple (PLOG..) or sequential (SPLOG..) protection logs, or Work data set printing:



where *type* is one of the following:



The IPLOGPRI function is used to print the sequential intermediate data sets created from the PLOG merge process. Input to ADAPLP IPLOGPRI must be a MERGINT1/MERGINT2 data set created by the ADARES utility and is specified in the DDPLOG DD JCL statement. Operation is similar to the SPLOGPRI function.

Optional Parameters and Subparameters

DEVICE: Device Type

DEVICE specifies device type on which the multiple protection data set to be printed is contained. This parameter is required only if the device type is different from the standard ADARUN device.

FILE: File for Which Data is to Be Printed

The FILE parameter can be used to limit printing to those protection records containing information about the specified Adabas file.

The FILE parameter cannot be specified with the RABN parameter, or when TYPE=C1, C5, ET, EEKZ, SAVO, or VEKZ is specified. Do not specify ISN with the RABN parameter.

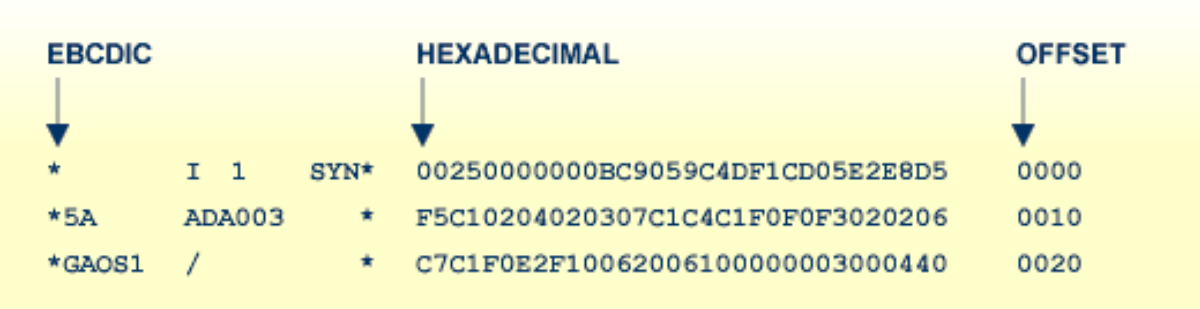
ISN: ISN for Which Data is to Be Printed

This parameter may be used to limit printing to the protection record identified by the specified ISN. The ISN parameter cannot be specified when the RABN parameter is specified, nor when TYPE=C1, C5, ET, EEKZ, SAVO, or VEKZ is specified.

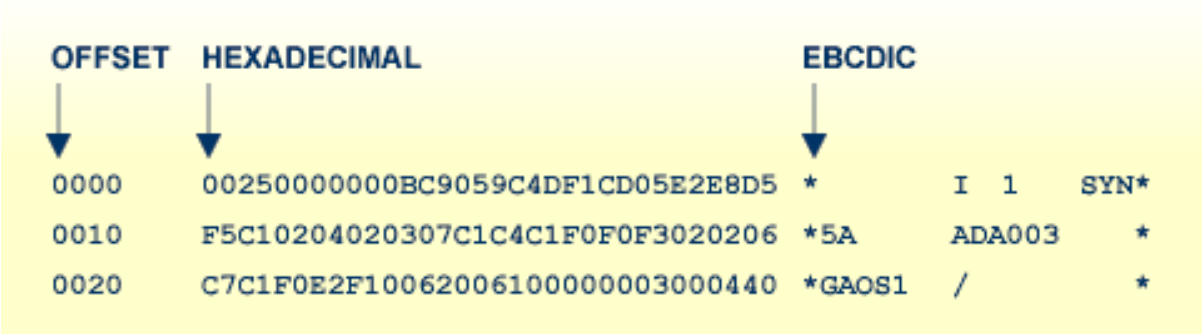
LAYOUT: Print Format

Controls the output format of the protection log record requested by the PRINT parameter. Specify either layout 1 (the default), 2, or 3:

LAYOUT=1 (the Default)



LAYOUT=2/3



LAYOUT=3 presents the same format as LAYOUT=2, and also includes an explanation of each PLOG record type.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMBER: Multiple Protection Log Data Set Number

NUMBER specifies the one of multiple (two through eight) protection log (PLOG) data sets to be printed. Specifying "2" selects the DD/PLOGR2 data set; specifying "3" selects the DD/PLOGR3 data set, etc. The default of "1" selects DD/PLOGR1.

PRINT: Print Entire Data Protection Record

The PRINT parameter prints the entire data protection log record. If this parameter is omitted, only the protection log record header is printed.

RABN: Print Only Updates for the Specified Data Storage Block

The RABN parameter can be used to track all updates to a particular Data Storage block that might be in error.

The parameter limits printing to the protection records that describe the before and after images of Data Storage records that have been removed from, updated in, or added to the specified Data Storage block.

The RABN parameter can be specified for TYPE=ALL (the default) or TYPE=DATA functions; that is, those that select data storage protection records.

SKIPRABN: Number of Blocks to Be Skipped

SKIPRABN specifies the number of blocks to be skipped before printing starts. Counting for the number of blocks to be printed (see STOPRABN parameter) begins after the number of blocks specified with this parameter have been skipped.

STOPRABN: Number of Blocks to Be Printed

STOPRABN limits the number of blocks to be printed. If this parameter is omitted, all blocks up to the end of the protection log are printed. In addition to the RABN count specified by STOPRABN, RABN 1 is also printed; therefore, the total number of printed RABNs is always one more than the value specified by STOPRABN.

TYPE: Type of Record to Be Printed

TYPE specifies the type of protection records to be selected for printing. The following values may be specified:

ALL	all protection records-the default
ASSO	Associator protection records
DATA	Data Storage protection records
C1	records resulting from Adabas C1 commands
C5	records resulting from Adabas C5 commands
EEKZx	records written at completion of a nucleus buffer flush
ET	records resulting from Adabas ET commands
REPR	Work data set records used by autorestart to repair the index
SAVO	online SAVE database/file records
VEKZ	records written at completion of update commands

Note: The number of protection records is reduced further by specifying the FILE, ISN, or RABN parameters.

Examples

Example 1:

```
ADAPLP WORKPRI PRINT,TYPE=ALL,STOPRABN=40
```

41 data protection records from the Adabas Work are to be printed.

Example 2:

```
ADAPLP WORKPRI PRINT,TYPE=ASSO,STOPRABN=10
```

11 Associator data protection blocks from the Adabas Work are to be printed.

Example 3:

```
ADAPLP PLOGPRI PRINT
```

All data protection blocks contained on one of multiple protection log data sets are to be printed.

120 JCL/JCS Requirements and Examples

- BS2000 698
- z/OS 701
- z/VM 704
- VSE 706

This section describes the job control information required to run ADAPLP with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	Required for WORKPRI.
Work	DDWORKR1 DDWORKR4	disk	Required for WORKPRI.
Sequential protection log	DDPLOG	tape/ disk	Required for SPLOGPRI or IPLOGPRI.
Multiple protection log	DDPLOGR1	disk	Required for PLOGPRI if NUMBER=1 (the default).
Multiple protection log	DDPLOGRn	disk	Required for PLOGPRI if NUMBER=n.
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAPLP parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAPLP messages	SYSLST DDDRUCK		*Messages and Codes*

ADAPLP JCL Examples (BS2000)

Print Sequential Protection Log

In SDF Format:

```
/.ADAPLP LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT SEQUENTIAL PROTECTION LOG
 /REMARK *
 /ASS-SYSLST L.PLP.SPLO
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDPLOG,ADAyyyyy.PLOG
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP SPLOGPRI PRINT
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAPLP LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT SEQUENTIAL PROTECTION LOG
 /REMARK *
 /SYSFILE SYSLST=L.PLP.SPLO
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.PLOG,LINK=DDPLOG
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP SPLOGPRI PRINT
 /LOGOFF NOSPOOL
```

Print Sequential Intermediate Protection Log

In SDF Format:

```
/.ADAPLP LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT SEQUENTIAL PROTECTION LOG
 /REMARK *
 /ASS-SYSLST L.PLP.SPLO
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDPLOG,ADAyyyyy.PLOG
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP IPLOGPRI PRINT
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAPLP LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT SEQUENTIAL PROTECTION LOG
 /REMARK *
 /SYSFILE SYSLST=L.PLP.SPLO
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.PLOG,LINK=DDPLOG
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP IPLOGPRI PRINT
 /LOGOFF NOSPOOL
```

Print One of Multiple Protection Log Data Sets

In SDF Format:

```
/.ADAPLP LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT MULTIPLE PROTECTION LOG
 /REMARK *
 /ASS-SYSLST L.PLP.PLOG
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOG
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP PLOGPRI PRINT
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAPLP LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT MULTIPLE PROTECTION LOG
 /REMARK *
 /SYSFILE SYSLST=L.PLP.PLOG
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.PLOG,LINK=DDPLOGR1
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP PLOGPRI PRINT
 /LOGOFF NOSPOOL
```

Print Work

In SDF Format:

```
/.ADAPLP LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT ADABAS WORK
 /REMARK *
 /ASS-SYSLST L.PLP.WORK
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
```

```
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP WORKPRI PRINT,TYPE=ASSO
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAPLP LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A P L P PRINT ADABAS WORK
 /REMARK *
 /SYSFILE SYSLST=L.PLP.WORK
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAPLP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPLP WORKPRI PRINT,TYPE=ASSO
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	Required for WORKPRI.
Work	DDWORKR1 DDWORKR4	disk	Required for WORKPRI.
Sequential protection log	DDPLOG	tape/ disk	Required for SPLOGPRI or IPLOGPRI.
Multiple protection log	DDPLOGR1	disk	Required for PLOGPRI if NUMBER=1 (the default).
Multiple protection log	DDPLOGRn	disk	Required for PLOGPRI if NUMBER=n.
ADAPLP messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations*
ADAPLP parameters	DDKARTE	reader	

ADAPLP JCL Examples (z/OS)

Refer to the JOBS data set for the following example jobs:

Job Member	Description
ADAPLP	Print protection log (from multiple data set PLOG)
ADAPLPS	Print protection log (from sequential PLOG)
ADAPLPW	Print Adabas Work

These jobs are listed in the following sections.

Print One of Multiple Protection Log Data Sets

```
//ADAPLP    JOB
//*
//*   ADAPLP: PROTECTION LOG PRINT (FROM MULTIPLE PLOG)
//*
//PLP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDPLOGR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR1 <=== PLOG
DATASET
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAPLP,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAPLP PLOGPRI
/*
```

Print Sequential Protection Log

```
//ADAPLPS    JOB
//*
//*   ADAPLP: PROTECTION LOG PRINT (FROM SEQUENTIAL PLOG)
//*
//PLP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
```

```
//DDPLOG    DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOG,  <=== PLOG DATASET
//                          UNIT=TAPE,VOL=SER=PLOG5
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X

//DDCARD    DD   *
ADARUN PROG=ADAPLP,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAPLP SPLOGPRI
/*
```

Print Sequential Intermediate Protection Log

```
//ADAPLPS    JOB
//*
//*   ADAPLP: PROTECTION LOG PRINT (FROM SEQUENTIAL PLOG)
//*
//PLP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDPLOG    DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOG,  <=== PLOG DATASET
//                          UNIT=TAPE,VOL=SER=PLOG5
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X

//DDCARD    DD   *
ADARUN PROG=ADAPLP,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAPLP IPLOGPRI
/*
```

Print Adabas Work

```
//ADAPLPW    JOB
//*
//*   ADAPLP: PRINT ADABAS WORK
//*
//PLP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
```

```
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAPLP,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAPLP WORKPRI PRINT
/*
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	Required for WORKPRI.
Work	DDWORKR1 DDWORKR4	disk	Required for WORKPRI.
Sequential protection log	DDPLOG	tape/ disk	Required for SPLOGPRI or IPLOGPRI.
Multiple protection log	DDPLOGR1	disk	Required for PLOGPRI if NUMBER=1 (default).
Multiple protection log	DDPLOGRn	disk	Required for PLOGPRI if NUMBER=n.
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAPLP parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAPLP messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAPLP Examples (z/VM)

Print Sequential Protection Log

```
DATADEF DDPLOG,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDPRINT,DSN=ADAPLP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAPLP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNPLP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=PLPSPLOG.CONTROL,MODE=A
ADARUN
```

Contents of RUNPLP CONTROL A1:

```
ADARUN PROG=ADAPLP,DEVICE=dddd,DB=yyyyy
```

Contents of PLPSPLOG CONTROL A1:

```
ADAPLP SPLOGPRI PRINT
```

Print Sequential Intermediate Protection Log

```
DATADEF DDPLOG,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDPRINT,DSN=ADAPLP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAPLP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNPLP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=PLPSPLOG.CONTROL,MODE=A
ADARUN
```

Contents of RUNPLP CONTROL A1:

```
ADARUN PROG=ADAPLP,DEVICE=dddd,DB=yyyyy
```

Contents of PLPSPLOG CONTROL A1:

```
ADAPLP IPLOGPRI PRINT
```

Print One of Multiple Protection Log Data Sets

```
DATADEF DDPLOG,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDPRINT,DSN=ADAPLP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDPLOGR1,DSN=ADAPLP.DDPLOG,MODE=A
DATADEF DDDRUCK,DSN=ADAPLP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNPLP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=PLPDPLOG.CONTROL,MODE=A
ADARUN
```

Contents of RUNPLP CONTROL A1:

```
ADARUN PROG=ADAPLP,DEVICE=dddd,DB=yyyyy
```

Contents of PLPDPLOG CONTROL A1:

```
ADAPLP PLOGPRI PRINT
```

Print Work

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADAPLP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAPLP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNPLP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=PLPWORK.CONTROL,MODE=A
ADARUN
```

Contents of RUNPLP CONTROL A1:

```
ADARUN PROG=ADAPLP,DEVICE=dddd,DB=yyyyy
```

Contents of PLPWORK CONTROL A1:

```
ADAPLP WORKPRI PRINT,TYPE=ASSO
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	Required for WORKPRI.
Work	WORKR1	disk	*	Required for WORKPRI.
Sequential protection log	PLOG	tape disk	SYS014 *	Required for SPLOGPRI or IPLOGPRI.
Multiple protection log	PLOGR1	disk	*	Required for PLOGPRI if NUMBER=1 (default).
Multiple protection log	PLOGRn	disk	*	Required for PLOGPRI if NUMBER=n.

File	Symbolic Name	Storage	Logical Unit	More Information
ADAPLP report		printer	SYS009	
ADARUN messages		printer	SYSLST	
ADARUN parameters	SYSRDR CARD	reader/tape/ disk		*Operations*
ADAPLP parameters	SYSIPT	reader		

* *Any programmer logical unit may be used.*

ADAPLP JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for a description of the VSE procedures (PROCs).

Refer to the following members for these examples:

Example	Member
Print sequential protection log	ADAPLPS.X
Print multiple protection log	ADAPLP.X
Print Adabas Work	ADAPLPW.X

Print Sequential Protection Log

```
* $$ JOB JNM=ADAPLPS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAPLPS
*       PROTECTION LOG PRINT (FROM SEQUENTIAL PLOG)
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS014,cuu
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL PLOG,'EXAMPLE.DByyyyy.PLOG'
// MTC REW,SYS014
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAPLP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAPLP SPLOGPRI
/*
/&
* $$ EOJ
```

Print Sequential Intermediate Protection Log

```
* $$ JOB JNM=ADAPLPS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAPLPS
*       PROTECTION LOG PRINT (FROM SEQUENTIAL PLOG)
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS014,cuu
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL PLOG,'EXAMPLE.DByyyyy.PLOG'
// MTC REW,SYS014
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAPLP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAPLP IPLOGPRI
/*
/&
* $$ EOJ
```

Print One of Multiple Protection Log Data Sets

```
* $$ JOB JNM=ADAPLP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAPLP
*       PROTECTION LOG PRINT (FROM MULTIPLE PLOG)
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAPLP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAPLP PLOGPRI
/*
/&
* $$ EOJ
```

Print Adabas Work

```
* $$ JOB JNM=ADAPLPW,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAPLPW
*       PRINT ADABAS WORK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAPLP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAPLP WORKPRI PRINT
```

```
/*
/&
* $$ EOJ
```

XV ADAPRI Utility: Print Selected Adabas Blocks

This chapter covers the following topics:

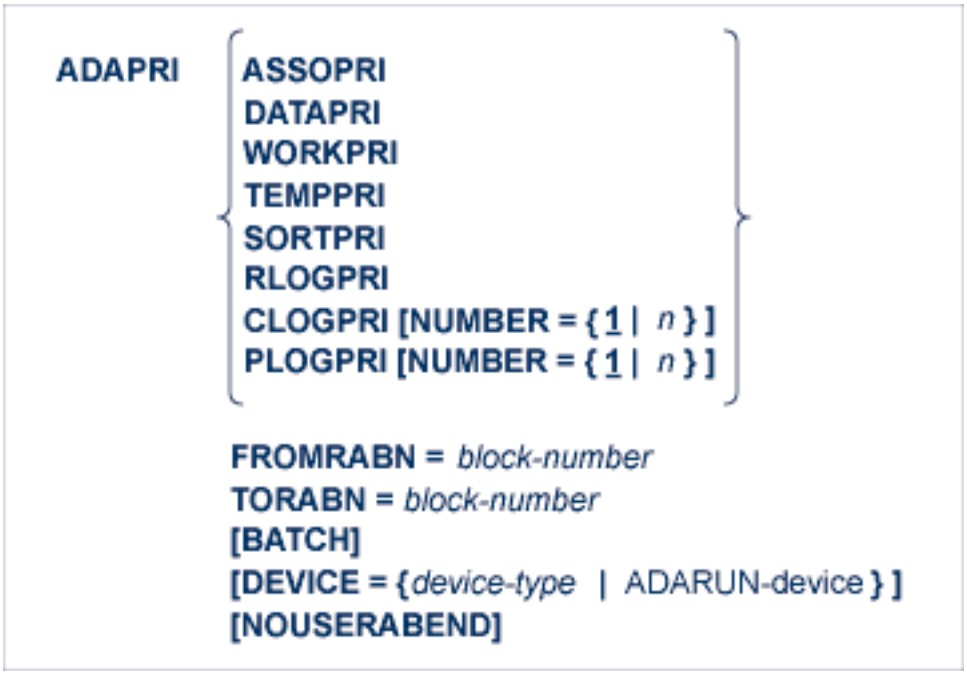
Functional Overview
ADAPRI Syntax and Examples
JCL/JCS Requirements and Examples

121 Functional Overview

The ADAPRI utility prints the contents of a block (or range of blocks) contained in the Associator (ASSO..), Data Storage (DATA..), Work (WORK..), temp (TEMP..), sort (SORT..), multiple data set command log (CLOG), multiple data set protection log (PLOG), or the recovery log (RLOG)) data set. More than one data set may be printed during a single ADAPRI execution.

122 ADAPRI Syntax and Examples

- Essential Parameters 716
- Optional Parameters 716
- Examples 872



This chapter describes the syntax and parameter of the ADAPRI utility.

Essential Parameters

FROMRABN/ TORABN: Range of Blocks to Be Printed

The beginning and ending numbers of the RABNs to be printed. Both values must be specified; there are no defaults.

Printing begins with the block number specified with the FROMRABN parameter and ends with the block number specified with the TORABN parameter. Each block in the range is printed in hexadecimal format.

Optional Parameters

BATCH: Output Format

Controls the line length of the printed output. If BATCH is not specified, the default line size is 80 characters. If BATCH is specified, the output line size is 120 characters.

DEVICE: Device Type

The device type that contains the data set to be printed. This parameter is required if the device type is different from the standard device type assigned by the ADARUN DEVICE parameter.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMBER: Command/ Protection Log Data Set Number

The number of the multiple (two through eight) command log (CLOG) or protection log (PLOG) data set from which the blocks are to be printed. NUMBER can only be specified when CLOGPRI or PLOGPRI is specified. When NUMBER=2 is specified, DD/CLOGR2 blocks are printed; if the CLOGPRI or PLOGPRI function is specified without NUMBER, the blocks are taken from DD/CLOGR1 (the default).

Examples

Example 1:

```
ADAPRI ASSOPRI FROMRABN=1,TORABN=2
```

Blocks 1 and 2 of the Associator (which contain the general control blocks) is printed.

Example 2:

```
ADAPRI DATAPRI FROMRABN=8000,TORABN=8120
```

Blocks 8000 to 8120, inclusively, of Data Storage are printed.

Example 3:

```
ADAPRI WORKPRI FROMRABN=1,TORABN=100,BATCH
```

Blocks 1 to 100 of the Adabas Work are to be printed. The output line size to be used is 120.

Example 4:

```
ADAPRI CLOGPRI FROMRABN=1,TORABN=80,BATCH,NUMBER=2
```

Blocks 1 to 100 of the command log data set DD/CLOGR2 are printed in 120-character-wide format.

Example 5:

```
ADAPRI DSIMPRI FROMRABN=1,TORABN=1
```

Block 1 only of the DSIM data set is printed. The DSIM data set is only used if Adabas Delta Save Facility is installed.

Example 6:

```
ADAPRI ASSOPRI FROMRABN=X'19D619',TORABN=X'19D619'
```

Block 1693209 of the Associator is printed. Note the use of hexadecimal values in the FROMRABN and TORABN parameters.

123 JCL/JCS Requirements and Examples

- BS2000 720
- z/OS 721
- z/VM 722
- VSE 723

This section describes the job control information required to run ADAPRI with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Data set containing blocks to print	DDASSORn DDDATARn DDWORKR1 DDWORKR4 DDTEMPR1 DDSORTRn DDCLOGRn DDPLOGRn DDDSIMR1	disk disk disk disk disk disk disk disk disk	Associator Data Storage* Work* Temp Sort Multiple command log Multiple protection log DSIM data set
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
ADARUN parameters	SYSDTA/ DDCARD	reader	*Operations*
ADAPRI parameters	SYSIPT/ DDKARTE	reader	
ADARUN messages	SYSOUT	printer	*Messages and Codes*
ADAPRI messages	SYSLST	printer	*Messages and Codes*

** When printing blocks from Data Storage or Work, the link name for the Associator must also be present.*

ADAPRI JCL Example (BS2000)

In SDF Format:

```
/.ADAPRI LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A P R I ALL FUNCTIONS
 /REMARK *
 /ASS-SYSLST L.PRI
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAPRI,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPRI DATAPRI FROMRABN=27,TORABN=34
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAPRI LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A P R I ALL FUNCTIONS
 /REMARK *
 /SYSFILE SYSLST=L.PRI
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAPRI,DB=yyyyy,IDTNAME=ADABAS5B
 ADAPRI DATAPRI FROMRABN=27,TORABN=34
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Data set containing the blocks print	DDASSORn DDDATARn DDWORKR1 DDWORKR4 DDTEMPR1 DDSORTR1 DDCLOGRn DDPLOGRn DDDSIMR1	disk disk disk disk disk disk disk disk disk	Associator Data Storage* Work* Temp Sort Multiple command log Multiple protection log DSIM data set
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
ADARUN parameters	DDCARD	reader	*Operations*
ADAPRI parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAPRI messages	DDDRUCK	printer	*Messages and Codes*

** When printing blocks from Data Storage or Work, the DD statement for the Associator must also be present.*

ADAPRI JCL Example (z/OS)

```
//ADAPRI    JOB
//*
//*   ADAPRI:
//*      MAINTENANCE PRINT
//*
//PRI      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDTEMPR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.TEMPR1 <=== TEMP
//DDSORTR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.SORTR1 <=== SORT
//DDPLOGR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR1 <=== PLOG 1
//DDPLOGR2  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR2 <=== PLOG 2
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAPRI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAPRI ASSOPRI DEVICE=eeee,FROMRABN=1,TORABN=1
/*
```

Refer to ADAPRI in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Data set containing the blocks to print	DDASSORn DDDATARn DDWORKR1 DDWORKR4 DDTEMPR1 DDSORTR1 DDCLOGRn DDPLOGRn DDDSIMR1	disk disk disk disk disk disk disk disk disk	Associator Data Storage* Work* Temp Sort Multiple command log Multiple protection log DSIM data set
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAPRI parameters	DDKARTE	disk/ terminal/ reader	

Data Set	DD Name	Storage	More Information
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAPRI messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

** When printing blocks from Data Storage or Work, the DD statement for the Associator must also be present.*

ADAPRI Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAPRI.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAPRI.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNPRI.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAPRI.CONTROL,MODE=A
ADARUN
```

Contents of RUNPRI CONTROL A1:

```
ADARUN PROG=ADAPRI,DEVICE=dddd,DB=yyyyy
```

Contents of ADAPRI CONTROL A1:

```
ADAPRI DATAPRI FROMRABN=27,TORABN=34
```

VSE

File	Symbolic Name	Storage	More Information
Files containing the blocks to be printed	ASSORn DATARn WORKR1 TEMPR1 SORTR1 CLOGRn PLOGRn SIMR1	disk disk disk disk disk disk disk disk	Associator Data Storage* Work* Temp Sort Multiple command log Multiple protection log DSIM data set
Recovery log (RLOG)	RLOGR1	disk	Required when using ADARAI
ADARUN messages	SYSLST	printer	*Messages and Codes*
ADAPRI messages	SYS009	printer	*Messages and Codes*

File	Symbolic Name	Storage	More Information
ADARUN parameters	SYSRDR CARD	reader tape/ disk	*Operations*
ADAPRI parameters	SYSIPT	reader	

** When printing blocks from Data Storage or Work, the JCS statement for the Associator must also be present.*

ADAPRI JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

```
* $$ JOB JNM=ADAPRI,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAPRI
*       MAINTENANCE PRINT
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAPRI,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAPRI ASSOPRI DEVICE=eeee,FROMRABN=1,TORABN=1
/*
/&
* $$ EOJ
```

Refer to member ADAPRI.X for this example.

XVI ADARAI Utility: Adabas Recovery Aid

The following functions are available for use with ADARAI:

Functional Overview
CHKDB: Check the Database Recovery Status
DISABLE: Disable Recovery Logging
LIST: Display Current RLOG Generations
PREPARE: Initialize and Start the RLOG
RECOVER: Build a Recovery Job Stream
REMOVE: Remove the Recovery Aid
JCL/JCS Requirements and Examples

124 Function Overview

- Concepts and Components 728

The ADARAI utility prepares the recovery log (RLOG), lists the information contained in the RLOG, creates the job control statements to recover the database, and disables ADARAI logging.

Transaction recovery is provided whenever an Adabas session is abnormally terminated. The Adabas autobackout routine, which is automatically invoked at the beginning of every Adabas session, removes the effects of all interrupted transactions from the database. See the restart/recovery information in the *Adabas Operations* documentation.

However, when a database data set (ASSO, DATA, or WORK) is destroyed, it is necessary to restore and regenerate the database to recover the lost data.

The Adabas Recovery Aid utility ADARAI can be used to automate and optimize *database* recovery. It records and reports all information needed to recover the database and builds the recovery job stream (JCL/JCS), which is the basis for reexecuting the jobs performed from the time of the last SAVE to the point of failure and error.

Note: The job stream generation function is not yet available under VSE or z/VM.

Concepts and Components

The Adabas Recovery Aid comprises two components:

- an interface (ADARAC) to collect information as relevant events occur against the database; and
- a utility (ADARAI) to list the information collected, generate jobs to recover the database or files on the database, or deactivate recovery logging.

This section covers the following topics:

- The Collection Interface
- Recovery Log (RLOG)
- Generation: The Unit of Recovery
- Retaining Noncurrent Generations

The Collection Interface

The collection interface is called by the nucleus and by all utilities to record information about each event that occurs; for example, a nucleus stop/start, a utility execution, or an event generated by the Adabas Online System.

Recovery Log (RLOG)

The interface records all event information into a Recovery log file (RLOG) for use by the utility component. The RLOG stores the information about data sets, utility parameters, and protection logs needed to build the recovery job control. The RLOG data set is DD/RLOGR1.

In a nucleus cluster environment, all nuclei use the same RLOG. Concurrent updates to the RLOG are controlled by a lock.

Notes:

1. Sequential data sets used by the utilities whose runs are logged on the RLOG must be kept and available for any recovery operation; for example, the DD/EBAND input to an ADALOD LOAD operation.
2. ADADBS file changes are now recorded on the RLOG data set.
3. Information recorded in the RLOG generally exceeds that required for recovery; it can also be used as a record of events that have occurred on a database over a period of time.

Generation: The Unit of Recovery

Information is stored on the RLOG by generation, the logical unit used for recovery.

A *generation* includes all activity between consecutive operations of:

- ADASAV SAVE/RESTORE (database),
- RESTORE GCB, or
- SAVE DELTA/RESTORE DELTA (database).

The first generation includes the first operation and extends to (but excludes) the second. A new generation is started when a database can be recovered in full after the previous operation.

Generations may be normal, restricted, or erroneous:

- A generation is labeled "normal" if a full save was available when it started and no unusual events occurred while activities were being logged in it.
- A generation is labeled "restricted" when certain events occur during the logging cycle that make it impossible for ADARAI to rebuild the database without user intervention. ADARAI generates a job, but the job will not run without help from the user. For example, if the Work data set is decreased in size, the user must create a Work data set with the original size so that the recovery job can run correctly up to the point where the Work data set size was decreased.
- A generation is labeled "erroneous" when errors occur during the logging cycle, for whatever reason. ADARAI generates a job, but the job will not run without changes.

Note: When a generation becomes restricted or erroneous, Software AG recommends that you start a new generation as soon as possible by performing an on- or off-line save of the database. If the Adabas Delta Save is installed, a SAVE DELTA will start a new generation.

Retaining Noncurrent Generations

Noncurrent generations provide a history of operations that have affected the database for use in problem resolution or for audit purposes.

Access to noncurrent generations is essential if an attempt to recover a database fails after the RESTORE step in the recovery job is executed. At this point, the generation being recovered becomes the current generation. If it then becomes necessary to rebuild the recovery job, the generation being recovered will be an older generation.

The RLOG retains the number of generations specified by the MINGENS parameter during the ADARAI PREPARE step. ADARAI recycles generations when the number stored on the RLOG reaches the number specified by the MINGENS parameter.

When a new generation plus those already stored exceed the available RLOG space, one of two events will occur:

- if the minimum number of generations as specified by MINGENS can be maintained, the oldest generation is overwritten; otherwise
- the RLOG is placed *out of service* by setting a flag in the RLOG control block. In this case, data is no longer logged.

125 CHKDB: Check the Database Recovery Status

```
ADARAI  CHKDB  [{ACTIVE | INACTIVE}]
```

The ADARAI CHKDB function checks that the recovery status of the nucleus is the status specified by the CHKDB function (active or inactive). To do this CHKDB issues a command to the nucleus and tests the nucleus response code.

If the command does not provide the expected response code, CHKDB reissues another command after ten seconds. Up to ten commands are issued. If the desired nucleus status (active/inactive) does not occur after ten tries, ADARAI terminates with error 158.

Example:

```
ADARAI CHKDB
```

Tests the recovery nucleus for active status.

126 DISABLE: Disable Recovery Logging

The ADARAI DISABLE function disables recovery logging by setting the RLOG table (control block) to inactive status.

Note: ADARAI DISABLE must be executed with the database inactive.

Following DISABLE, information is no longer recorded in the RLOG and the current generation is ended. The content of the RLOG before DISABLE is maintained and can still be listed or otherwise used for recovery purposes.

Recovery logging can be started again by starting a new generation. See **Generation: The Unit of Recovery** .

Example:

```
ADARAI DISABLE
```

Deactivates all Recovery Aid logging.

127 LIST: Display Current RLOG Generations

- Additional LIST Information on BS2000 736
- Syntax 737
- Optional Parameters 737
- Examples 739

Note: Adabas version 6 RLOGs cannot be listed; only version 7 and above RLOGs are supported.

The ADARAI LIST function is used to view the RLOG contents in table form:

- generations are listed in numerical order;
- RLOG block ranges are listed for each generation; and
- the stop/start dates and times covered by each generation are listed.

The following information is provided for each entry on the RLOG including utility executions and nucleus session start and session stop entries:

- name of the event for which the RLOG entry was written;
- date and time the information was written to the RLOG;
- PLOG number associated with the event (if any);
- PLOG block containing an associated checkpoint (if any);
- parameters specified for the logged event to the DD/CARD and DD/KARTE statements; and
- details of any files written or read during the logged event.

In a nucleus cluster environment, the PLOG data sets are also listed on nucleus session start entries. The cluster nucleus ID (NUCID) is also listed.

Example:

```
*** 2001-08-21  11:37:08  NUCLEUS PLOG NUMBER=4
*** START NUCLEUS SESSION NUCID 40002

SYNC PLOG BLOCK NUMBER = 1
ACTIVE PLOG DATA SET NAMES: EXAMPLE.DBddddd.PLOGR21
                            EXAMPLE.DBddddd.PLOGR22
```

Additional LIST Information on BS2000

On BS2000 systems, LIST also provides the following information:

- file or file generation group (FGG) characteristics and physical location (tape, disk, etc.);
- existence and condition of each data set (written and erased; overwritten; written as a temporary file) needed by the ADARAI RECOVER function; and
- error or warning message for incompatible coded file ID (CFID) comparison.

Since non-matching coded file IDs (CFIDs) are a reliable indication of overwritten data sets in BS2000, LIST compares CFIDs to determine whether any data sets have been overwritten. For lost or overwritten data sets, LIST provides an error or warning indication for the following conditions:

- An "ERROR" is indicated when the data set was written to:
- - disk, and was then erased;
  - disk, and then was overwritten (CFIDs do not match). Both the original and the catalog entry are reported;
  - a temporary disk data set.
- A "WARNING" is indicated when the data set was written to
- - tape, and the catalog entry has been erased;
  - tape, and then was overwritten (CFIDs do not match). Both the original and the catalog entry are reported;
  - a temporary tape data set.

The section **Output Examples** provides examples of the operating-system-dependent results provided by the LIST function.

Syntax

```
ADARAI   LIST   [GENS = { NO | YES } ]
                [RELGEN = { gen-number | gen-number - gen-number } ]
                [RLOGDEV = { device | ADARUN-device } ]
```

Optional Parameters

GENS: Generation Print Control

GENS determines whether generation information is listed. GENS=NO lists only the RLOG control information. GENS=YES (the default) lists generation information also.

RELGEN: Relative Recovery Generation Number

RELGEN specifies the *relative* generation number (or range of generation numbers) to be listed. The current generation is always coupled with relative generation "0" (zero). The last completed generation is coupled with relative generation "1"; *two generations ago*, the generation before the last completed generation, is specified as relative generation "2".

Example:

To list the generations ranging from three generations ago to the last complete generation (inclusive), specify RELGEN=3-1.

If the first generation number specified is lower than the second generation number, ADARAI reduces the second generation number to match the first.

Example:

If you specify RELGEN=2-3, ADARAI changes it to RELGEN=2-2.

If RELGEN is not specified, all generations are printed.

The specified generation must currently be in the RLOG. Note, however, that instead of a relative number, each listed generation has an ascending order number, beginning with 1 (the first generation following the start of RLOG operation).

Example:

RELGEN=0 is equivalent to generation number 690; RELGEN=3-2 is equivalent to the generation numbers 687 and 688.

```
I GEN-   I      BLOCK      I                 DATE /TIME                 I
I NUMBER I   FROM       TO I        FROM                       TO       I
I--------I-----------------I-------------------------------------------I
I    690 I    715      715 I 2001-08-20  02:07:13   2001-08-20 08:51:19 I
I    689 I    714      714 I 2001-08-17  18:24:49   2001-08-20 02:03:21 I
I    688 I    713      713 I 2001-08-16  18:24:26   2001-08-17 16:48:16 I
I    687 I    712      712 I 2001-08-15  18:29:09   2001-08-16 12:54:28 I
I    686 I    711      711 I 2001-08-14  18:24:30   2001-08-15 17:45:44 I
I    685 I    710      710 I 2001-08-13  18:32:07   2001-08-14 15:46:25 I
I    684 I    709      709 I 2001-08-13  02:07:15   2001-08-13 18:00:18 I
I    683 I    708      708 I 2001-08-10  18:25:59   2001-08-13 02:03:23 I
I    682 I    707      707 I 2001-08-09  18:36:39   2001-08-10 10:24:14 I
```

RLOGDEV: RLOG Alternate Device

RLOGDEV specifies the device type containing the RLOG file. If the RLOG file is located on the device type specified by the ADARUN DEVICE parameter (the default device type), you do not need to specify RLOGDEV.

Examples

Input Examples

```
ADARAI LIST
```

This example lists all generations in the RLOG.

```
ADARAI LIST RELGEN=15-1
```

LIST displays the last 15 generations (if they are available in the RLOG), not including the current generation (0).

Output Examples

BS2000

The following example shows LIST output for a single BS2000 disk data set:

```
LINK=DDSAVE1 PATHNAME=:A:$GEB.RAI.vv.SAVE.012
SIZE=6387 SEC-ALLO=96 LPP=6336
FCBTYPE=SAM RECFORM=V BLKSIZE=(STD,16) RECSIZE=32748 BLKCNTRL=PAMKEY
VSN/DEV PUBA00/D3480  /AC PUBA01/D3480  /AC
VSN/DEV PUBA02/D3480  /AC
```

The following example shows LIST output for a BS2000 file-generation group (FGG):

```
FGG INDEX BASE=10 CURRENT=10 FIRST=1 MAX=255 DISP=DEL
LINK=DDSAVE1 PATHNAME=:A:$GEB.RAI.vv.SAVE.TAPE.01(*0010)
TAPE  DEVICE=TAPE=C1 (B5) FSEQ=1 BLKCOUNT=4000
FCBTYPE=SAM RECFORM=V BLKSIZE=32760 RECSIZE=32756 BLKCNTRL=NO
VOLUMES GEBR11 GEBR12 GEBR13 GEBR17 GEBR19
```

The following examples are of ADARAI LIST error/warning output. The first is for a lost disk data set, and the second for an overwritten tape data set:

```
LINK=DDSAVE1 DISC DATASET NOT PRESENT - E R R O R
ORIGINAL CATALOG ENTRY:
LINK=DDSAVE1 PATHNAME=:A:$GEB.RAI.vv.SAVE.012
SIZE=6387 SEC-ALLO=96 LPP=6336
FCBTYPE=SAM RECFORM=V BLKSIZE=(STD,16) RECSIZE=32748 BLKCNTRL=PAMKEY
VSN/DEV PUBA00/D3480  /AC PUBA01/D3480  /AC
VSN/DEV PUBA02/D3480  /AC
```

```
LINK=DDSAVE1 CFID MISMATCH - W A R N I N G
ORIGINAL 379949EE  NOW: 379972F0
ORIGINAL CATALOG ENTRY:
LINK=DDSAVE1 PATHNAME=:A:$GEB.RAI.vv.SAVE.012
TAPE  DEVICE=TAPE=C1 (B5) FSEQ=1 BLKCOUNT=4000
FCBTYPE=SAM RECFORM=V BLKSIZE=32760 RECSIZE=32756 BLKCNTRL=NO
CREATION DATE yyyy-mm-dd 11:44:35
VOLUMES GEBR11 GEBR12 GEBR13 GEBR17 GEBR19
ACTUAL CATALOG ENTRY:
LINK=DDSAVE1 PATHNAME=:A:$GEB.RAI.vv.SAVE.012
TAPE  DEVICE=TAPE=C1 (B5) FSEQ=1 BLKCOUNT=3900
FCBTYPE=SAM RECFORM=V BLKSIZE=32760 RECSIZE=32756 BLKCNTRL=NO
CREATION DATE yyyy-mm-dd 12:34:56
VOLUMES GEBR23 GEBR65 GEBR66 GEBR67 GEBR68
```

z/OS

```
A D A R A I   Vv.v  SMv   DBID = 00203  STARTED          yyyy-mm-dd  hh:mm:ss

PARAMETERS:
-----------
   ADARAI LIST RELGEN=0
   RECOVERY LOG FILE FOR DATABASE 203

   START RABN FOR LOG DATA AREA IS     21
   HIGHEST LOG AREA RABN IS            633
   CURRENT VALUE FOR ROTATING RABN IS  23

I GEN-   I   I      BLOCK      I                 DATE /TIME                 I
I NUMBER I S I   FROM       TO I          FROM                     TO       I
I--------I---I-----------------I-----------------------------------------I
I      3 I N I     23       23 I yyyy-01-13  16:06:28   yyyy-01-13 16:11:35 I
I      2 I N I     22       22 I yyyy-01-09  16:07:10   yyyy-01-13 16:04:13 I
I      1 I N I     21       21 I yyyy-01-09  16:04:41   yyyy-01-09 16:06:16 I
I      0 I R I     20       20 I yyyy-01-09  16:04:07   yyyy-01-09 16:04:30 I
I--------I---I-----------------I-----------------------------------------I

*** yyyy-01-13  16:06:28
*** SAVE DATABASE OFFLINE

   DELTA SAVE ID IS AS FOLLOWS:
    FULL SAVE..............2
    LOW DELTA SAVE NUMBER...0
    HIGH DELTA SAVE NUMBER..0
    DATE WRITTEN............yyyy-01-13
    TIME WRITTEN............16:12:03

   FILES = 1,2,3,19


      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
```

```
      ADARUN NCLOG=2,CLOGDEV=3390,CLOGSIZE=150
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES
      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADASAV

     ADASAV SAVE

    //DDSAVE1  DD DSN=EXAMPLE.ADASAV.FULL.G0058V00,
    //            UNIT=3390,SPACE=(TRK,(5,5)),DISP=NEW,
    //            DCB=(RECFM=VB,BLKSIZE=27998,LRECL=27994),
    //            VOL=SER=(SMS018)
```

DDSAVE1 VOLSER=SMS018 FROM BLOCK=1 (ASSO) TO BLOCK =1598 VOLUME IS ASSOCIATED WITH PLOG NO. 6

```
   DDSAVE1    VOLSER=SMS018    FROM BLOCK=1 (DATA)
                                TO BLOCK  =750
              VOLUME IS ASSOCIATED WITH PLOG NO. 6

*** yyyy-01-13  16:07:09  NUCLEUS PLOG NUMBER=7
*** START NUCLEUS SESSION [NUCID=nnnnn]

   SYNC PLOG BLOCK NUMBER = 5
   [ACTIVE PLOG DATASET NAMES: EXAMPLE.DBddddd.PLOGR21
                               EXAMPLE.DBddddd.PLOGR22]

      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
      ADARUN NCLOG=2,CLOGSIZE=150,CLOGDEV=3390
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES
      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADANUC
      ADARUN MODE=MULTI
      ADARUN LOCAL=YES
      ADARUN SPT=NO
      ADARUN LWP=480000
      ADARUN LP=200
      ADARUN TT=1800
      ADARUN TNAE=1800
      ADARUN LBP=80000
      ADARUN NH=500
      ADARUN LFP=60000
      ADARUN LU=65525
      ADARUN NAB=45
      ADARUN LQ=12000
      ADARUN LI=20000
```

```
      ADARUN NT=10
      ADARUN NC=300
      ADARUN NU=300
      ADARUN LS=20000
      ADARUN TNAX=1800
      ADARUN CT=300
      ADARUN OPENRQ=NO
      ADARUN LOGGING=NO
      ADARUN LOGCB=NO
      ADARUN LOGSB=NO
      ADARUN LOGFB=NO

      ADARUN IGNDIB=NO
      ADARUN FORCE=NO

*** yyyy-01-13  16:07:18  NUCLEUS PLOG NUMBER=7
*** END NUCLEUS SESSION


   HIGHEST PLOG BLOCK WRITTEN = 7

*** yyyy-01-13  16:07:22
*** COPY MULTIPLE PROTECTION LOG DATASET FOR PLOG 7

      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
      ADARUN NCLOG=2,CLOGSIZE=150,CLOGDEV=3390
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES
      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADARES,MODE=MULTI

      ADARES PLCOPY OPENOUT
      ADARES DSIMSIZE=5

    //DDSIAUS1 DD DSN=EXAMPLE.PLOG.G0243V00,UNIT=3390,
    //            SPACE=(TRK,(10,1)),DISP=NEW,DCB=(RECFM=VB,
    //            BLKSIZE=27998,LRECL=27994),
    //            VOL=SER=(SMS018)

    DDSIAUS1   VOLSER=SMS018    FROM BLOCK=1
                                TO BLOCK  =7
                                FROM DATE =yyyy-01-13  17:07:09
                                TO   DATE =yyyy-01-13  17:07:18
               VOLUME IS ASSOCIATED WITH PLOG NO. 7

*** yyyy-01-13  16:07:39  NUCLEUS PLOG NUMBER=8
*** START NUCLEUS SESSION [NUCID=nnnnn]

   SYNC PLOG BLOCK NUMBER = 3
   [ACTIVE PLOG DATASET NAMES: EXAMPLE.DBddddd.PLOGR21
                               EXAMPLE.DBddddd.PLOGR22]
```

```
      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
      ADARUN NCLOG=2,CLOGSIZE=150,CLOGDEV=3390
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES

      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADANUC
      ADARUN MODE=MULTI
      ADARUN LOCAL=YES
      ADARUN SPT=NO
      ADARUN LWP=480000
      ADARUN LP=200
      ADARUN TT=1800
      ADARUN TNAE=1800
      ADARUN LBP=80000
      ADARUN NH=500
      ADARUN LFP=60000
      ADARUN LU=65525
      ADARUN NAB=45
      ADARUN LQ=12000
      ADARUN LI=20000
      ADARUN NT=10
      ADARUN NC=300
      ADARUN NU=300
      ADARUN LS=20000
      ADARUN TNAX=1800
      ADARUN CT=300
      ADARUN OPENRQ=NO
      ADARUN LOGGING=NO
      ADARUN LOGCB=NO
      ADARUN LOGSB=NO
      ADARUN LOGFB=NO
      ADARUN IGNDIB=NO
      ADARUN FORCE=NO

*** yyyy-01-13  16:09:16  NUCLEUS CHECKPOINT
ENCOUNTERED

    CHECKPOINT IS ON PLOG NUMBER 8 BLOCK NUMBER 4
    SYNS-CHECKPOINT IS 'DELETE FILE'
    FILES = 1

*** yyyy-01-13  16:09:16  NUCLEUS CHECKPOINT ENCOUNTERED

    CHECKPOINT IS ON PLOG NUMBER 8 BLOCK NUMBER 5
    SYNS-CHECKPOINT IS 'DELETE FILE'
    FILES = 2

*** yyyy-01-13  16:10:27  NUCLEUS PLOG NUMBER=8
```

```
*** ADABAS UTILITY RUN

    SYNP-CHECKPOINT ID IS 'ADALOD - LOAD'
    SYNP-CHECKPOINT IS FOUND ON PLOG 8 IN BLOCK NO. 6
    FILES = 1

      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
      ADARUN NCLOG=2,CLOGSIZE=150,CLOGDEV=3390
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES
      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADALOD,MODE=MULTI

      ADALOD LOAD FILE=1
      ADALOD NAME='EMPLOYEES'
      ADALOD MAXISN=1500,DSSIZE=1
      ADALOD TEMPSIZE=15,SORTSIZE=15

    //DDEBAND  DD
DSN=ADABAS.ADAvrs.EMPL,UNIT=3390,DISP=OLD,
    //            VOL=SER=(ADA001)

*** yyyy-01-13  16:11:21  NUCLEUS PLOG NUMBER=8
*** ADABAS UTILITY RUN

   SYNP-CHECKPOINT ID IS 'ADALOD - LOAD'
    SYNP-CHECKPOINT IS FOUND ON PLOG 8 IN BLOCK NO. 7
    FILES = 2

     ADARUN
PROG=ADALOD,MODE=SINGLE,SVC=249,DEVICE=3390,DBID=203

      ADALOD LOAD FILE=2
      ADALOD NAME='VEHICLES'
      ADALOD MAXISN=1000,DSSIZE=1
      ADALOD TEMPSIZE=15,SORTSIZE=15


    //DDEBAND  DD
DSN=ADABAS.ADAvrs.VEHI,UNIT=3390,DISP=OLD,
    //            VOL=SER=(ADA001)

*** yyyy-01-13  16:11:31  NUCLEUS PLOG NUMBER=8
*** END NUCLEUS SESSION

    HIGHEST PLOG BLOCK WRITTEN = 9

*** yyyy-01-13  16:11:35
*** COPY MULTIPLE PROTECTION LOG DATASET FOR PLOG 8
```

```
      ADARUN DBID=203,SVC=249,DEVICE=3390,PLOGRQ=YES
      ADARUN NCLOG=2,CLOGSIZE=150,CLOGDEV=3390
      ADARUN NPLOG=2,PLOGSIZE=1350
      ADARUN PLOGDEV=3390
      ADARUN DSF=YES
      ADARUN UEX2=USEREX2M
      ADARUN PROG=ADARES,MODE=MULTI

      ADARES PLCOPY OPENOUT
      ADARES DSIMSIZE=5

    //DDSIAUS1 DD DSN=EXAMPLE.PLOG.G0244V00,UNIT=3390,
    //            SPACE=(TRK,(10,1)),DISP=NEW,DCB=(RECFM=VB,
    //            BLKSIZE=27998,LRECL=27994),
    //            VOL=SER=(SMS018)

    DDSIAUS1   VOLSER=SMS018    FROM BLOCK=1
                                TO BLOCK  =9
                                FROM DATE =yyyy-01-13  17:07:39
                                TO   DATE =yyyy-01-13  17:11:30
               VOLUME IS ASSOCIATED WITH PLOG NO. 8

A D A R A I  TERMINATED NORMALLY                          yyyy-01-13  16:12:03
```

VSE

```
A D A R A I   Vv.v  SMv   DBID = 00059  STARTED    yyyy-mm-dd  hh:mm:ss

PARAMETERS:

ADARAI LIST


18:45:04 ADAI64  FILE RLOGR1  HAS BEEN OPENED IN ECKD MODE
18:45:04 ADAI64  FILE RLOGM1  HAS BEEN OPENED IN ECKD MODE

RECOVERY LOG FILE FOR DATABASE 59

START RABN FOR LOG DATA AREA IS     26
HIGHEST LOG AREA RABN IS            633
CURRENT VALUE FOR ROTATING RABN IS  26


I GEN-   I   I      BLOCK      I                 DATE /TIME                 I
I NUMBER I S I   FROM       TO I          FROM                     TO       I
I--------I---I-----------------I-------------------------------------------I
I      1 I N I     26       26 I yyyy-08-30  17:06:51   yyyy-08-30 18:44:35 I
I      0 I R I     25       25 I yyyy-08-30  17:01:02   yyyy-08-30 17:05:05 I
I--------I---I-----------------I-------------------------------------------I
```

```
*** yyyy-08-30  17:06:51
*** SAVE DATABASE OFFLINE NON INCREMENTAL

SAVE DATASET PLOG NUMBER         = 1966
ADASAV SAVE
// TLBL SAVE1,'PMIG.ADAvrs.SAVE1',0,ADES01

DDSAVE1    VOLSER=XXXXXX    FROM BLOCK=1
                          TO BLOCK  =6192
        VOLUME IS ASSOCIATED TO PLOG NO. 1966
FILE=001,002,003,004,005,006,008,009,010,011,012,013,014,015
FILE=016,017,019,021,022,023,025,027

DDSAVE1    VOLSER=XXXXXX    FROM BLOCK=1
                          TO BLOCK  =31961
        VOLUME IS ASSOCIATED TO PLOG NO. 1966
FILE=001,002,003,004,005,006,008,009,010,011,012,013,014,015
FILE=016,017,019,021,022,023,025,027

*** yyyy-08-30  17:08:12  NUCLEUS PLOG NUMBER=1967
*** START NUCLEUS SESSION

*** yyyy-08-30  17:10:15  NUCLEUS PLOG NUMBER=1967
*** ADABAS UTILITY RUNSYNP-CHECKPOINT ID IS 35 (UNLOAD FILE)
SYNP-CHECKPOINT IS FOUND ON PLOG 1967 IN BLOCK NO. 5
FILE=001
ADAULD FILE=1 NUMRECS=100
// DLBL OUT1,'VSESP.SAPLB.ULD2',7,SD
// EXTENT SYS034,SYSWK1,1,0,16365,30

*** yyyy-08-30  17:14:28  NUCLEUS PLOG NUMBER=1967
*** ADABAS UTILITY RUN

SYNP-CHECKPOINT ID IS 35 (UNLOAD FILE)
SYNP-CHECKPOINT IS FOUND ON PLOG 1967 IN BLOCK NO. 8
FILE=001
ADAULD FILE=1 NUMRECS=100
// DLBL OUT1,'VSESP.SAPLB.ULD2',7,SD
// EXTENT SYS034,SYSWK1,1,0,16365,30

*** yyyy-08-30  18:44:35  NUCLEUS PLOG NUMBER=1967
*** ADABAS UTILITY RUN

SYNP-CHECKPOINT ID IS 30 (LOAD FILE)
SYNP-CHECKPOINT IS FOUND ON PLOG 1967 IN BLOCK NO. 12
FILE=004
ADALOD LOAD FILE=4,ISNREUSE=YES,SORTSIZE=5,TEMPSIZE=5,DSSIZE=50B
ADALOD
MAXISN=10,NAME='TESTFILE',DSREUSE=YES,LWP=1000000,LIP=500
ADALOD NUMREC=5,NISIZE=5B,UISIZE=5B
// DLBL EBAND,'VSESP.SAPLB.ULD2',7,SD
// EXTENT SYS011,SYSWK1,1,0,16365,30
```

```
ADAIO3 RLOGR1           3 READS         0 WRITES
ADAIO3 RLOGM1           1 READS         0 WRITES

A D A R A I  TERMINATED NORMALLY                        yyyy-08-30  18:45:03
```

128 PREPARE: Initialize and Start the RLOG

- Syntax 750
- Essential Parameter 751
- Optional Parameters 751
- Examples 751

The recovery log (RLOG) must be prepared before it can be used. The following steps are required to start the RLOG file:

Step 1. Format the RLOG file using the ADAFRM RLOGFRM function.

Before running ADARAI PREPARE, the RLOG data set must be formatted using the RLOGFRM function of the ADAFRM utility. If it is not, an error 159 is returned.

Step 2. Run the ADARAI PREPARE function to prepare the RLOG.

ADARAI PREPARE must be executed with the database inactive.

The ADARAI PREPARE function is used to define

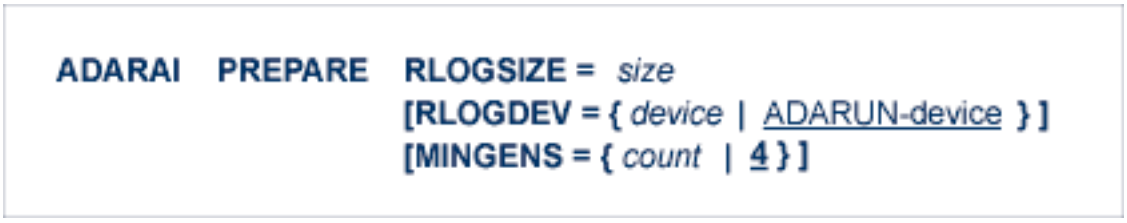
- the size of the RLOG (the size must be the same as the value of the SIZE parameter of the ADAFRM RLOGFRM function);
- the minimum number of generations to retain (4 is the default); and
- the device type (the default is the device type specified by the ADARUN DEVICE parameter).

Step 3. Run the ADASAV SAVE (database) function to begin the first log generation.

After the PREPARE function executes, logging begins for the initial generation; however, this generation has a *restricted* status because it has not been started by a full database save or restore.

See the *Adabas Operations* documentation for more information about generation statuses.

Syntax



Essential Parameter

RLOGSIZE: RLOG Area Size

RLOGSIZE defines the size of the RLOG file in cylinders or blocks. This value must be the same as that defined by the SIZE parameter of the ADAFRM RLOGFRM function. RLOGSIZE *must* be specified; there is no default.

Note: The RLOG data set is limited to 16,777,215 (x'FFFFFF') blocks/RABNs.

Optional Parameters

MINGENS: RLOG Generation Count

MINGENS specifies the number of logging generations to hold in the RLOG. The RLOG numbers the generations in ascending order starting with "0". The minimum is 4 generations (the default); the maximum is 32.

RLOGDEV: RLOG Device Type

RLOGDEV specifies the device type containing the RLOG file. If the RLOG file is located on the device type specified by the ADARUN DEVICE parameter (the default device type), you do not need to specify RLOGDEV.

Important: If you choose a device type for the RLOG data set that is different from the default, you must specify the RLOGDEV parameter for all ADARES PLCOPY and COPY executions as well.

Examples

Example 1:

```
ADARAI PREPARE MINGENS=4,RLOGSIZE=5
```

This ADARAI PREPARE function defines and initializes the RLOG to hold the minimum of four generations in a log size of five cylinders. The RLOG device defaults to that specified by the ADARUN DEVICE parameter.

Example 2:

```
ADARAI PREPARE
RLOGSIZE=20,MINGENS=20,RLOGDEV=3390
```

This example defines a larger RLOG size (20 cylinders) to hold as many as 20 generations on a 3390 device type.

129 RECOVER: Build a Recovery Job Stream

- Recovery Processing 754
- Optimized Recovery Processing 756
- Requirements 756
- Restrictions 757
- Input Needed for Recovery 926
- Output from the Recovery Operation 758
- Executing the RECOVER Function 759
- File-Level Recovery 760
- Syntax 761
- Optional Parameters and Subparameters 761
- Examples 764
- Skeleton Job Control 764
- User Exit to Change JCL 949
- Prerecovery Checking 767
- Restarting the RECOVER Function or Recovery Job Stream 768

Note: The RECOVER function is currently available for BS2000 and z/OS systems only. Support for z/VM and VSE systems is planned.

The ADARAI RECOVER function builds the job control information (recovery job stream) for recovering the Adabas database or selected database files. The RECOVER function

- reads the PLOG information to determine if a PLCOPY is needed; and
- reads the RLOG to build the recovery job stream from the skeleton job control.

ADARAI RECOVER builds the job stream necessary to restore the database or files to the condition before the RECOVER function was run. The completed job stream is sent to the DD/JCLOUT data set.

Where appropriate, ADARAI includes error or information messages in the generated job stream. You must then manually correct the errors before submitting the job. The existence of messages in the job stream is indicated by a nonzero return code from ADARAI RECOVER.

For BS2000 systems, RECOVER additionally

- performs, when generating the job control, the same checks performed by the LIST function for BS2000; and
- includes BS2000 /REMARK statements in the created job control for checks that produce errors.

Note: When such errors occur, the job control must be corrected manually.

Recovery Processing

The ADARAI RECOVER function builds a job based on the exact sequence it finds in the generation to be recovered:

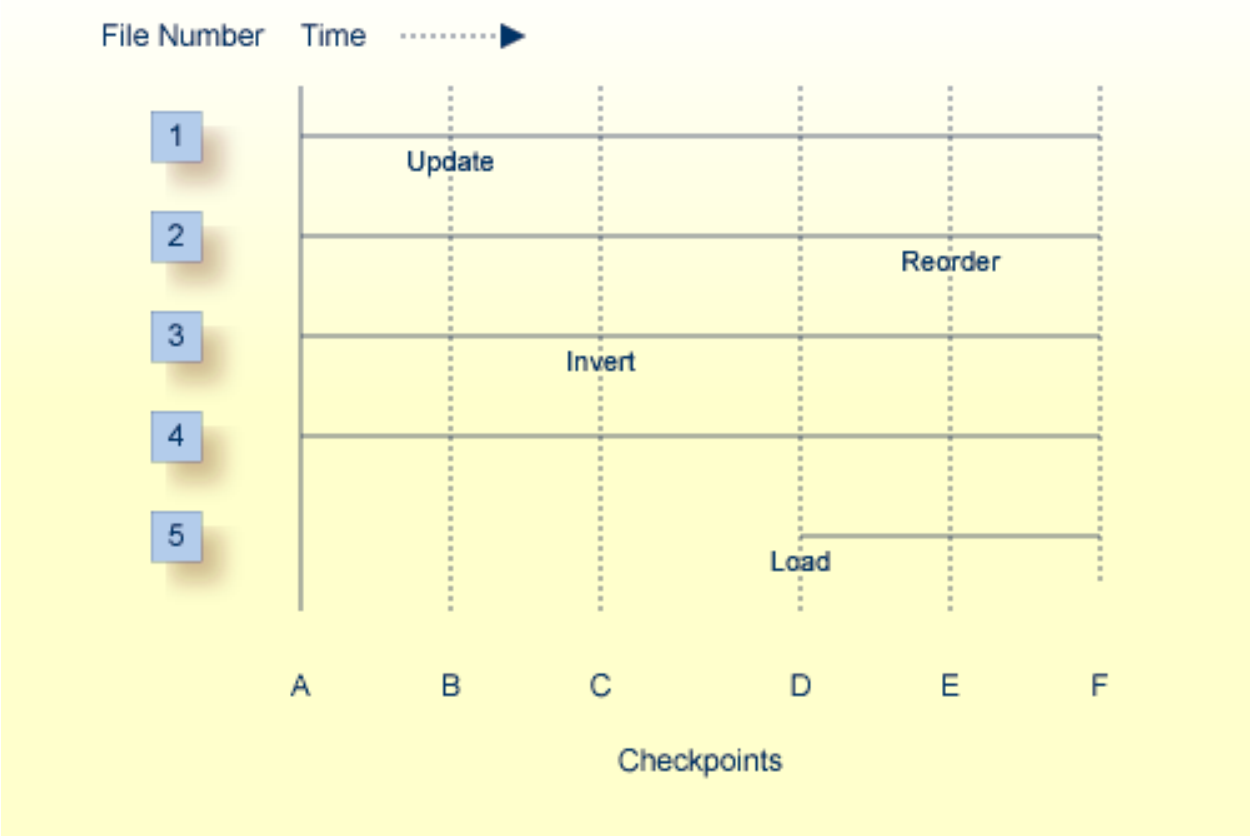
- it restores the database from the data sets created by the operation that started the generation;
- it regenerates PLOGs up to the next utility checkpoint found;
- it generates a job step to reexecute the utility and start the regeneration after that checkpoint.

This sequence continues until all utilities have been replayed and the last PLOG block in the generation has been regenerated.

The following diagram illustrates the functioning of ADARAI where

- a database is saved to start a new generation at A.
- the database runs and at various times during the generation
- an update is run against file 1;

- a reorder is run against file 2;
- an invert is run against file 3; and
- a load is run against file 5.



Given the above, the order of the recovery is as follows:

1. A full save or full save plus delta saves are restored to return the database to the status at A.
2. A database regenerate runs from the first checkpoint at A up to the update checkpoint at B. The regenerate job then terminates.
3. The update utility runs for file 1 and a database regenerate runs between the checkpoint at B and the invert checkpoint at C.
4. The invert utility runs for file 3 and a database regenerate runs between checkpoint C and the load checkpoint at D.
5. The load utility runs for file 5 and a database regenerate runs between checkpoint D and the reorder checkpoint at E.
6. The reorder runs for file 3 and a database regenerate runs between checkpoint E and the most up-to-date level of the database at F.

Optimized Recovery Processing

The OPT parameter for the RECOVER function of ADARAI is used to identify operations or sequences that would minimize the time required to recover a large database.

For example, it a file with 10,000 updates is deleted or reloaded, it should be possible to avoid restoring the file from the start, replaying the 10,000 updates, and then throwing it all away when the delete or load operation occurs.

When optimization is selected, ADARAI does not restore the example file in the main restore for the job. Regeneration for the file occurs only after the file has been deleted or created by the load:

- a deleted file has no more updates;
- for a file created by a load, only updates subsequent to the load are important.

When an optimized job stream is used, the recovered database is rebuilt in a way that is different from the original build. Because optimized recovery jobs do not replay in exactly the same way as the original jobs, problems may occur in the recovery; for example, insufficient space may be available on the database. In most cases, however, the risk is minimal compared with the potential benefits of optimizing the database recovery. Each situation must be examined for potential problems.

Requirements

To generate a recovery job that will run successfully, ADARAI imposes the following conditions:

- the database must be run with dual or multiple protection logging active.
- sequential data sets input to utility functions that update files in the database must be retained.
- sequential output data sets created by SAVE or MERGE functions must be retained. This applies to SAVE FILE functions only if RESTFILE=YES is used for ADARAI RECOVER.
- retained data sets must keep their original names; ADARAI cannot track copies with different names.

Software AG recommends that retained data sets be cataloged.

Restrictions

Shadow Databases

If *shadow* databases, or copies of normal production databases, are built by restoring the delta save output and DSIM data set for a save of the original database, ADARAI has no knowledge of the PLOG activity that occurred during the delta save on the original database and therefore cannot rebuild the DSIM data set if a restore operation becomes necessary on the shadow database.

If, however, the DSIM data set and the delta save data set are merged to create a new *offline* delta save data set and the new merged data set is restored to the shadow database, ADARAI has all the information needed to recover the shadow database since the PLOG is not necessary in this case.

Restoring Delta Saves with a DSIM Data Set

In general, ADARAI handles RESTORE DELTA processing without problems. However, if the RESTORE DELTA uses a DSIM data set (which is essentially a *working* data set), the DSIM data set may not be intact if an ADARAI RECOVER becomes necessary. ADARAI therefore records the COPY or PLCOPY requests used to create a DSIM data set and emits a job step to rebuild the data set before attempting to replay such a RESTORE DELTA during RECOVER processing.

ADARAI searches the entire RLOG for appropriate entries. If the entries cannot be found, ADARAI cannot rebuild the DSIM data set prior to the RESTORE step and therefore cannot replay the RESTORE DELTA.

DD/FILEA File

In a generated recovery job, ADARAI writes the DD/FILEA file of the ADAORD utility. This cannot be avoided because the REORDER functions must be replayed and they require that the DD/FILEA file be written.

In this case, the following restrictions apply:

- ADAORD STORE processing simply reads the same DD/FILEA read when the utility was originally run as part of the generation being recovered.
- A temporary file (DISP=(NEW,DELETE), which is deleted after the step is executed) can be used for DD/FILEA because the recovery job creates and deletes the file again when it is executed.
- An existing file (DISP=OLD) can be used for DD/FILEA. If it still exists when the recovery job is run, ADARAI simply allocates the file with the disposition it had when the original job was run.
- If a new file (DISP=NEW,CATLG) is allocated for DD/FILEA and retained in the original ADAORD REORDER step, and if it still exists when the recovery job comes to the REORDER

step (which is normal), ADARAI attempts to create the same file again, which causes the job to fail.

- If a GDG is used, the ADARAI recovery job sees only the name of the actual data set created by the generation. If the data set already exists (which is normal), ADARAI attempts to create the same file again, which causes the job to fail.

Input Needed for Recovery

The following data sets are input to the ADARAI RECOVER function:

- DD/RLOGR1, the recovery log.
- DD/PLOGR1 and DD/PLOGRn, the multiple protection logs, which are required when the ADARAI RECOVER parameter FEOFPL=YES (the default) is used.
- DD/JCLIN, which provides site-dependent skeleton job control statements. The RECOVER operation merges these statements with the RLOG information to create a complete database recovery job stream.

On BS2000 systems, DDJCLIN is a SAM data set with variable record format. EDT can be used to create and edit this data set. See the section **Skeleton Job Control** for more information.

On z/OS systems, the DDJCLIN data set must be defined with RECFM=FB, LRECL=80, and a BLKSIZE that is a multiple of 80 bytes.

Output from the Recovery Operation

The ADARAI RECOVER output is an execution-ready job stream for recovering the database. This recovery job stream is written to the DD/JCLOUT file. If a possible error condition is detected during the RECOVER operation, ADARAI issues a warning message and ends with a condition code of 4. See the section ***Prerecovery Checking***.

On BS2000 systems, DDJCLOUT and DDJCLCON are SAM data sets with variable record format. They conform to the BS2000 job control conventions.

On z/OS systems, the DDJCLOUT DD statement must point to a data set defined with RECFM=FB, LRECL=80, and a BLKSIZE that is a multiple of 80 bytes.

The recovery job stream includes job steps to start the nucleus

- before the first regenerate job step; and
- after any utility operation that causes the nucleus to terminate automatically.

ADARAI RECOVER jobs replay all utilities with the database active, whether the utility was originally run in single-user mode or not. Utilities originally run in single-user mode are replayed in multiuser mode. These job steps are described in the sections **Building the Recovery Job Stream** and **Skeleton Job Control**.

Executing the RECOVER Function

The RECOVER function is run a generation at a time under control of the RELGEN parameter. If RELGEN is not specified, the default is the current generation.

RECOVER can be executed with the nucleus active or inactive. It can be executed more than once for the same generation because it does not change the RLOG information for that generation.

However, if RECOVER is rerun after a failure while running a DD/JCLOUT recovery job stream, the new recovery job stream produced may be different from the original recovery job stream. The reason is that the original recovery job stream may execute utilities against the database that updates the RLOG. The new RECOVER operation then builds a recovery job stream for the utilities that ran as part of the failed recovery job stream.

Also, if the recovery job stream failed after executing an ADASAV RESTORE, a new generation is created. In this case, execute RECOVER using the RELGEN=1 parameter setting to obtain the original generation.

Processing the PLOG

At the start of execution, if ADARAI RECOVER FEOFPL=YES, RECOVER reads the PLOGs, looking for information that must be copied. If necessary, it calls the nucleus to force a PLOG switch. If the nucleus is inactive, it invokes user exit 2.

Reading the Recovery Log

Next, RECOVER reads the skeleton job control into storage and reads the RLOG in chronological order, starting at the beginning of the generation specified by the RELGEN parameter. See **Generation: The Unit of Recovery** for a definition of generation.

If the entire database is being recovered, RECOVER uses the ADASAV SAVE or RESTORE information to create a new RESTORE/RESTONL database operation. For file-level recovery, it uses the SAVE/RESTORE database information to create a RESTORE FILE=... function.

Building the Recovery Job Stream

After creating a job stream for restoring the database or file, RECOVER creates a job step for starting the nucleus, using the %%JCL-STARTNUC statement.

RECOVER then creates the first regenerate job step. This job step does not contain a FROM checkpoint (FROMCP) unless an online SAVE (or DELTA SAVE) was the basis for starting the generation. In that case, the regenerate starts at the end checkpoint (SYN2) of the online save.

All PLOGs up to and including the next utility checkpoint (at which the REGENERATE must stop) are included and appropriate parameters are provided to the ADARES REGENERATE function. If more than 99 PLOGs are to be regenerated, ADARAI generates multiple REGENERATE job steps, each one processing up to 99 input PLOG data sets.

Once the PLOGs are regenerated up to the next utility execution, the utility job step is generated into the output recovery job. ADARAI then inserts another REGENERATE job step that includes all PLOGs up to and including the next utility checkpoint.

The recovery job continues inserting REGENERATE steps and utility steps until it detects the end of the generation specified by the RELGEN parameter. At this point, the completed job stream is sent to the DD/JCLOUT file.

File-Level Recovery

Recovery can be made on a file level by specifying the RECOVER function's FILE parameter. The file-level recovery process is essentially the same as the database-level recovery process, but is restricted to the files specified using the FILE parameter.

ADARAI produces a file-specific result in DD/JCLOUT by adding parameters to utility execution statements. For example, assume that the following statement was in the original ADASAV RESTORE statement:

```
ADASAV RESTORE FMOVE=2,3,NIRABN=100,1000,DSSIZE=550B,20
```

In this case, RECOVER FILE=3 produces the following DD/JCLOUT statement:

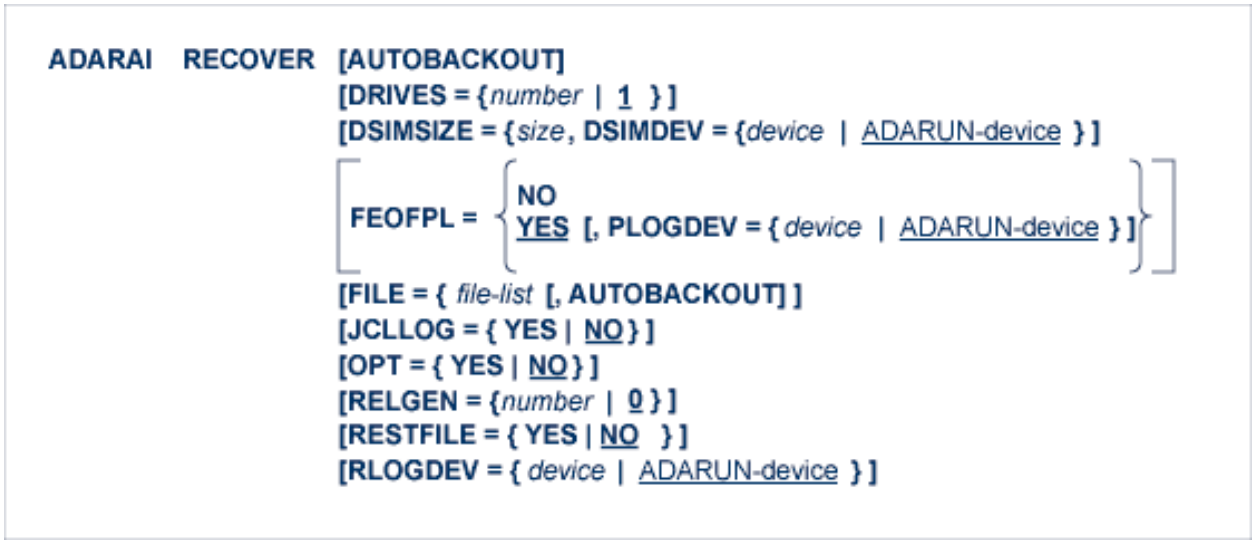
```
ADASAV RESTORE FMOVE=2,3,NIRABN=100,1000,DSSIZE=550B,20
ADASAV EXCLUDE=2
```

Note: If a file to be recovered is part of an expanded file chain or is coupled, all files in the chain or the coupled list must be recovered together. If all coupled files or expanded file chains are not recovered together, ADARAI detects this and the ADARAI RECOVER function fails.

The Adabas nucleus *must* be active before executing a file-level recovery job. This is different from the database-level recovery job, which starts the database itself.

A file-level RECOVER operation does not create job control for utilities that were executed on the whole database (for example, ADADEF NEWWORK). The exceptions to this are utilities that can be reexecuted for individual files as well as the complete database. An example is ADASAV RESTORE (database), which provides a DD/SAVE input data set that can be used to create ADASAV RESTORE FILE=... job control.

Syntax



Optional Parameters and Subparameters

AUTOBACKOUT: Back Out Uncompleted Transactions

AUTOBACKOUT may only be specified for file-level recovery.

If AUTOBACKOUT is specified, transactions that were not complete at the end of the last REGENERATE function in the recovery job are backed out. Only completed transactions are left on the database.

If AUTOBACKOUT is *not* specified, incomplete transactions are left on the database.

For database-level recovery, incomplete transactions at the end of the last REGENERATE function are always backed out.

DRIVES: ADASAV Restore Input Drive Volumes

DRIVES is the number of input data sets to be used as input to the RESTORE step of the recovery job being generated.

The specified DRIVES parameter must be equal to or less than the DRIVES parameter on the job that started the generation. For example, if the generation was started with a database save with DRIVE=4, the RECOVER DRIVES parameter may only be specified as 1, 2, 3, or 4.

When you specify a lower number of DRIVES for the RESTORE step, ADARAI RECOVER allocates only the DD/RESTn DD/DLBLs required and allocates an equal number of input data sets for each DD/RESTn DD/DLBL.

DSIMDEV: DSIM Data Set Device Type

DSIMDEV specifies the DSIM data set device type if different from that specified by the ADARUN DEVICE parameter, which is the default.

DSIMSIZE: Size of the DSIM Data Set

The size is specified in cylinders.

When the Adabas Delta Save Facility is active on the database being recovered, this parameter *must* be specified so that ADARAI can specify the DSIMSIZE parameter for any ADARES COPY operations it may have to generate.

FEOFPL: Synchronize Multiple PLOGs

If FEOFPL=YES (the default), ADARAI ensures that protection log (PLOG) data from all of the multiple PLOG data sets has been copied:

- If the nucleus is active, ADARAI forces a protection log switch. The nucleus then calls user exit 12, which copies the log data; ADARAI waits until the copying is completed. Note that the ADARUN parameter UEX12 must therefore be specified whenever FEOFPL=YES is specified.
- If the nucleus is not active, ADARAI itself calls user exit 12, which in turn copies the log data.

In a nucleus cluster environment, FEOFPL=YES functions differently:

- If at least one Adabas nucleus is available, ADARAI calls the nucleus to switch the PLOGs.
- If no Adabas nucleus is available, ADARAI generates a job that must be executed manually.

In either case, ADARAI must be restarted with FEOFPL=NO.

FILE: File Number

FILE specifies one or more database files to be included when the recovery job stream is built. Specifying FILE causes file- rather than database-level recovery; only those files specified are involved in the RECOVER operation. If FILE is not specified, all database files are included (the default).

JCLLOG: User-Supplied Job Control

JCLLOG controls listing of the user-supplied input job control (the JCL in DDJCLIN or the JCS in JCLIN). If JCLLOG=YES is specified, the user-supplied input job control elements are printed in the utility log. The default is no listing of input job control statements (NO).

OPT: Optimize Recovery Job for a Generation

When OPT=YES is specified, ADARAI attempts to optimize the recovery job it produces for a given generation; that is, it attempts to leave out steps that are not required to bring the database or file back to its original logical state.

When OPT=NO is specified, the recovery job is not optimized.

Note: When space on the database is limited, an optimized recovery job may fail due to the fact that the database is not built in exactly the same way as it originally was. If

this occurs, a recovery job generated without optimization should be used or the size of the database increased before recovery is attempted.

PLOGDEV: Multiple PLOG Device Type

The PLOGDEV value is only used when FEOFPL=YES is specified.

PLOGDEV specifies a PLOG device type different from that specified by the ADARUN DEVICE parameter, which is the default.

RELGEN: Relative Recovery Generation Number

RELGEN specifies the *relative* generation number to be used for recovery. The current generation is always coupled with relative generation "0" (zero), which is also the default. *Two generations ago*, or the generation before the last completed generation, is specified as relative generation "2".

The generation specified must currently be in the RLOG. Use the ADARAI LIST function to see the current RLOG generations available. Note, however, that the listed generations are numbered in ascending order, beginning with generation "1", the first generation following the start of RLOG operation.

RESTFILE: Create Restore File Jobstep

When RESTFILE=NO (the default), the DDJCLOUT recovery job stream does not include ADASAV RESTORE FILE=... job steps for logged ADASAV SAVE FILE= runs. Such job steps are not included because ADARES REGENERATE does not stop at ADASAV SAVE FILE=... checkpoints.

When RESTFILE=YES, ADARAI RECOVER creates an ADASAV RESTORE FILE=... job step in the recovery job stream for every ADASAV SAVE FILE=... utility execution logged.

Note: When using RESTFILE=YES, you must retain the file save data sets that are created in the generation.

When both RESTFILE=YES and OPT=YES are specified, the created RESTORE FILE= steps can speed the recovery process because restored files up to the RESTORE step are ignored.

When RESTFILE=YES and OPT=NO are specified, an unnecessary RESTORE step is included in the recovery job. You may wish to generate the recovery job in this way and then manually remove all steps prior to the RESTORE steps for the file(s) that are of interest.

RLOGDEV: RLOG Alternate Device

RLOGDEV specifies the device type containing the RLOG file. If the RLOG file is located on the device type specified by the ADARUN DEVICE parameter (the default device type), you do not need to specify RLOGDEV.

Examples

Example 1:

```
ADARAI RECOVER,DRIVES=3
```

The RECOVER function builds a recovery job stream based on the current generation (0, the default). The SAVE RESTORE portion of the job stream includes statements for three input data sets: DDREST1, DDREST2, and DDREST3.

Example 2:

```
ADARAI RECOVER  FILE=3,4,7,8,11
ADARAI          RELGEN=2, JCLLOG=YES
```

The recovery job stream is based on the third oldest generation; it includes activity for database files 3, 4, 7, 8, and 11 only; and creates a file-level job control. RECOVER also adds the user-supplied job control from data set DDJCLIN to the utility log.

Example3:

```
ADARAI RECOVER,RELGEN=1,OPT=Y
```

The RECOVER function builds a recovery job stream based on the last generation (i.e. the one preceding the current generation). ADARAI removes any unnecessary processing in order to speed up the recovery job.

Skeleton Job Control

Skeleton job control is contained in the DD/JCLIN file and is read as input to the RECOVER function. RECOVER merges it with the RLOG information to create the recovery job stream. Skeleton job control usually remains stable and is specific to your operating environment.

Each function in the skeleton job control is identified by a statement with the following format:

```
%%name
```

The name is specific to the function, such as %%JCL-ADASAV or %%JCL-STARTNUC. The job control statements follow the %% *name* statement; they are ended by the next %%JCL statement. Each skeleton section can contain any valid job control statement, including comments or program execution. This ability provides flexibility for the recovery process.

ADARAI does not check the validity of the statements in the skeleton job control. Invalid statements are first apparent when a job control error occurs during execution of the recovery job stream.

Job Header: %%JCL-JOB-HEADER

Job header statements are placed at the beginning of the recovery job stream before any other job control statements.

This job control relates to the complete recovery job and includes statements such as JOB and JOBLIB statements for OS or POWER JCL and JOB statements for VSE.

Job Trailer: %%JCL-JOB-TRAILER

Job trailer statements are placed at the end of the recovery job stream.

If the nucleus was started with the ADARUN UTIONLY=YES parameter as recommended in the %%JCL-STARTNUC section, you may want to provide a statement to execute an ADADBS OPERCOM UTIONLY=NO function in this section to make the database available after the recovery operation (see the skeleton job control examples later in this document).

Step Trailer: %%JCL-STEP-TRAILER

Step trailer statements are placed after each step in the recovery job stream.

DD/KARTE Job Control: %%JCL-DDKARTE

The operating-system-dependent DD/KARTE statements are included in each job step before DD/KARTE parameters generated by ADARAI from the RLOG.

For z/OS and VSE, these statements should indicate that the DD/KARTE parameters are contained in the job stream.

DD/FILEA Job Control: %%JCL-DDFILEA

This (optional) JCL card is provided to avoid problems with ADAORD REORDER processing. As a placeholder, it may be specified to provide a different DD/DLBL statement to the original DD/FILEA statement in the job. If specified, it will be inserted instead of the original DD/FILEA statement when an ADAORD REORDER is subsequently encountered.

Utility Job Control: %%JCL-*utility*

These skeleton sections are used to create utility job steps in the recovery job stream. The following utility jobs should be available in DD/JCLIN:

%%JCL-ADADEF	%%JCL-ADAORD
%%JCL-ADAINV	%%JCL-ADARES
%%JCL-ADALOD	%%JCL-ADASAV

Each of the sections should contain the following:

- The database files; for example, DD/ASSOR1, DD/DATAR1, DD/WORKR1, DD/SORTR1, DD/TEMPR1, and so on, as needed for the utility execution;
- DD/FILEA for ADALOD if used as a DD/TEMPR1 overflow file;
- A DD/PRINT and DD/DRUCK statement or assignment;
- A DD/CARD statement or assignment and all required ADARUN parameters; for example, DBID, DEVICE, PROG, SVC, and so on;

- Information needed about the Adabas library or other library.

It is possible to use a procedure or partitioned data set (PDS) member for the DD/CARD parameters, database files, or libraries.

Job Control to Start the Nucleus: %%JCL-STARTNUC

This job control comprises all the statements needed to start the Adabas nucleus. The RECOVER function uses this job control to create a job step for starting the nucleus before the first regenerate job step and, if the nucleus is not already active, before each call to a utility that requires an active nucleus.

The entire nucleus job must be included in this job control, including

- job statements;
- program execution statements;
- library definitions;
- database file definitions; and
- DD/CARD information, including the ADARUN parameters.

This section also requires a method for submitting the nucleus job control to the appropriate job entry system, such as EDT in procedure mode for BS2000 and IEBGENER for z/OS. For examples of this job control, see the %%JCL-STARTNUC sections in the examples of skeleton job control later in this document.

It is also important that this job control contain a way to stop execution of the recovery job stream until the nucleus is actually active. For example, a program can be created to issue a CL (close) command to the database; if a response code 148 indicates that the database is not active, the program can wait a specified time and reissue the CL command. The program continues until response code 0 occurs, and then ends to allow the next recover step to be performed. You can use the ADARAI CHKDB ACTIVE function for this purpose.

Job Control to Stop the Nucleus: %%JCL-ENDNUC

Whenever it detects a utility that requires an inactive nucleus, RECOVER inserts the %%JCL-ENDNUC job control in the job stream to ADAEND the Adabas nucleus. The ADADBS OPERCOM ADAEND function can also be used to stop the nucleus. If ADADBS OPERCOM is used, these job control statements must contain all necessary statements for running the ADADBS OPERCOM function. Like the Start Nucleus skeleton job control, a method to stop execution of the recovery job stream until the nucleus becomes inactive is also needed; the ADARAI CHKDB INACTIVE function can be used for this purpose.

Special Job Control Statements

The following special keywords/statements are used in the DD/JCLIN skeleton job control to control the generation of the DD/JCLOUT recovery job stream:

%STEP	When the (optional) %STEP keyword is included on the program execution statement, it generates a step number in the job stream for each job step that also includes the %STEP keyword. The step numbers run in ascending sequence, beginning with 1.
%SEQUENTIAL	*Must* be included in each %% skeleton section that generates a sequential file job control statement. ADARAI creates the necessary sequential job control statement in place of the %SEQUENTIAL statement. If this statement is not included, an error occurs during processing.
%KARTE	*Must* be included in each %% skeleton section where Adabas DD/KARTE parameters are generated. ADARAI creates the necessary DD/KARTE parameters in place of the %KARTE statement. If this statement is not included, an error occurs during processing.
%DBID	When the (optional) %DBID keyword is included on the program execution statement, it generates the five-digit database ID number. If the database number has less than five digits, the number is padded with leading zeros.

User Exit to Change JCL

ADARAI provides the user exit UEXRAI so that users may change an automatically generated recovery job before submitting it. Changes required might include the device type or the volume name.

UEXRAI obtains control of a JCL record immediately before it is written to DDJCLOUT.

The user exit is called with the following registers set:

R1	JCL record line that is about to be written to DDJCLOUT.
R13	standard 72-byte register save area
R14	return address
R15	entry point

Prerecovery Checking

Check the status of the recovery database and the recovery job stream before starting the recovery job stream.

For database-level recovery, check that

- the existing nucleus session has ended.
- the session entry has been deleted from the ID table.

Note: Any remaining DIB entry or pending nucleus session autorestart can be ignored; it is handled automatically by the initial RESTORE step.

- all required database components (ASSO, DATA, etc.) have been formatted at least once.
- Allocate and format any components changed during the generation to be recovered to the sizes and device types valid at the *beginning* of the generation.
  - Allocate and format any components that have changed size to the largest size used during the generation to be recovered.

For file-level recovery, check that

- the nucleus is active. The recovery job created by ADARAI does not start the nucleus automatically.

Restarting the RECOVER Function or Recovery Job Stream

If the ADARAI RECOVER function is interrupted, it can be restarted from the beginning, since the RECOVER function only reads the RLOG and does not change it.

The DD/JCLOUT recovery job stream created by the RECOVER function can be restarted as in a normal restore/regenerate process. However, the job stream may need to be edited to remove steps for the utility operations that were successfully completed. Following this, the recovery process can continue (providing the cause of the interruption has been removed), beginning with the failed utility operation.

It is always possible to restart an interrupted recovery job from the beginning. It may also be possible to restart the recovery job at the job step that failed or a few steps earlier, depending on the cause of the error and the job step that contained the error.

130 REMOVE: Remove the Recovery Aid

- Example 770

ADARAI REMOVE is functionally the same as the old ADARAI NORAI function; either REMOVE or NORAI can be specified.

Note: ADARAI REMOVE/NORAI must be executed with the database inactive.

The ADARAI REMOVE function disables recovery logging by updating the Associator GCBs to indicate that recovery logging (that is, the Recovery Aid) is no longer active in the database, and that information will no longer be recorded in the recover log (RLOG).

Existing RLOG information is maintained and available for listing or recovery operation following REMOVE, up until the next PREPARE operation is performed. Once the ADARAI PREPARE function is executed, all existing RLOG data is lost.

To restart recovery logging after using the REMOVE function, execute the ADARAI PREPARE function followed by an ADASAV SAVE/RESTORE database, RESTORE GCB, and/or SAVE DELTA/RESTORE DELTA (database) function to start a new generation. See the discussion of **ADARAI PREPARE** for information about preparing the RLOG.

Example

```
ADARAI REMOVE
```

Stops all Recovery Aid logging.

131 JCL/JCS Requirements and Examples

- BS2000 772
- z/OS 788
- z/VM 793
- VSE 795

This section describes the job control information required to run ADARAI with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

This section describes additional considerations and requirements for using ADARAI on a BS2000 system.

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	Required for RECOVER
Work	DDWORKR1 DDWORKR4	disk	Required for RECOVER
Data protection log	DDPLOGRn	tape/ disk	Required for RECOVER
Recovery log (RLOG)	DDRLOGR1	disk	
Job stream input	DDJCLIN	disk	Required for RECOVER
Recovery job output	DDJCLOUT	disk	Required for RECOVER
Recovery job output (JCL for console subtask)	DDJCLCON	disk	Optional; used only for RECOVER
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADARAI parameters	SYSDTA/ DDKARTE		*Operations*
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADARAI messages	SYSLST DDDRUCK		*Messages and Codes*

BS2000 Data Sets

Handling Sequential Disk Data Sets

There are no restrictions for sequential data sets on public disks.

Sequential data sets on private disks must not be exported; this means that the catalog entry must not be erased and that the recovery job control will not contain /IMPORT-FILE commands for data sets on private disks.

Handling Sequential Tape Data Sets

Software AG recommends not removing catalog entries for sequential data sets on tapes from the system's catalog. If RECOVER does not find a catalog entry for a sequential data set on tape or cartridge, it includes the following statement in the job stream:

```
/IMPORT-FILE FILE-NAME=tempfile ,...
```

where *tempfile* has the following structure:

```
#ADARAI.RECOVER.TAPE.nnnnn
```

Using File Generation Groups (FGGs)

There are no restrictions when using FGGs for sequential data sets, whether on disk or on tape.

Input Data Sets

ADARAI tries to access sequential BS2000 data sets using the original catalog entries. If a sequential data set is in a file generation group (FGG), ADARAI assigns the absolute member; ADARAI does not use or change the base pointer.

If sequential data sets are read-protected by passwords, the %%JCL-JOB-HEADER section in the skeleton job control must contain these passwords.

ADARAI neither exports nor erases data sets. If catalog entries for tape data sets no longer exist, ADARAI creates temporary data sets with the names #ADARAI.RECOVER.TAPE.nnnnn, where *nnnnn* is "00001", "00002", and so on.

ADARAI JCL Examples (BS2000)

Begin Recovery Logging (ADARAI PREPARE)

In SDF Format:

```
/.ADARAI LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A R A I   START RECOVERY LOGGING
/REMARK *
/ASS-SYSLST L.RAI.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,DB.yyyyy.ASSO
/SET-FILE-LINK DDRLOGR1,DB.yyyyy.RLOGR1,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARAI PREPARE RLOGSIZE=5,RLOGDEV=dddd,MINGENS=5
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARAI LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A R A I   START RECOVERY LOGGING
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.RAI.DATA
/FILE ADAvrs.MOD,LINK=DDLIB
/FILE DB.yyyyy.ASSO ,LINK=DDASSOR1
/FILE DB.yyyyy.RLOGR1,LINK=DDRLOGR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/EXEC (ADARUN,ADAvrs.MOD)
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARAI PREPARE RLOGSIZE=5,RLOGDEV=dddd,MINGENS=5
/LOGOFF NOSPOOL
```

List the RLOG (ADARAI LIST)

In SDF Format:

```
/.ADARAI LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A R A I   LIST RECOVERY LOGS
/REMARK *
/ASS-SYSLST L.RAI.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,DB.yyyyy.ASSO
/SET-FILE-LINK DDRLOGR1,DB.yyyyy.RLOGR1
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARAI LIST GENS=NO,RLOGDEV=dddd,RELGEN=1
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARAI LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A R A I   LIST RECOVERY LOGS
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.RAI.DATA
/FILE ADAvrs.MOD,LINK=DDLIB
/FILE DB.yyyyy.ASSO ,LINK=DDASSOR1
```

```
/FILE DB.yyyyy.RLOGR1,LINK=DDRLOGR1
/EXEC (ADARUN,ADAvrs.MOD)
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARAI LIST GENS=NO,RLOGDEV=dddd,RELGEN=1
/LOGOFF NOSPOOL
```

Create Recovery JCL (ADARAI RECOVER)

In SDF Format:

```
/.ADARAI LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A R A I   BUILD RECOVERY JCL STREAM
/REMARK *
/DELETE-FILE DB.yyyyy.JCLOUT
/SET-JOB-STEP
/CREATE-FILE DB.yyyyy.JCLOUT,PUB(SPACE=(48,48))
/SET-JOB-STEP
/DELETE-FILE DB.yyyyy.JCLCON
/SET-JOB-STEP
/CREATE-FILE DB.yyyyy.JCLCON,PUB(SPACE=(48,48))
/SET-JOB-STEP
/ASS-SYSLST L.RAI.DATA
/ASS-SYSDTA *SYSCMD

/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,DB.yyyyy.ASSO
/SET-FILE-LINK DDRLOGR1,DB.yyyyy.RLOGR1
/SET-FILE-LINK DDPLOGR1,DB.yyyyy.PLOGR1
/SET-FILE-LINK DDPLOGR2,DB.yyyyy.PLOGR2
/SET-FILE-LINK DDJCLIN,DB.yyyyy.JCLIN
/SET-FILE-LINK DDJCLOUT,DB.yyyyy.JCLOUT
/SET-FILE-LINK DDJCLCON,DB.yyyyy.JCLCON
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARUN UEX2=EXITR2
ADARAI RECOVER PLOGDEV=2201,FEOFPL=YES,RELGEN=1
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARAI LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A R A I   BUILD RECOVERY JCL STREAM
/REMARK *
/REMARK *

/SYSFILE SYSLST=L.RAI.DATA
/FILE ADAvrs.MOD,LINK=DDLIB
/FILE DB.yyyyy.ASSO,LINK=DDASSOR1
/FILE DB.yyyyy.RLOGR1,LINK=DDRLOGR1
/FILE DB.yyyyy.PLOGR1,LINK=DDPLOGR1
/FILE DB.yyyyy.PLOGR2,LINK=DDPLOGR2
/FILE DB.yyyyy.JCLIN,LINK=DDJCLIN
/FILE DB.yyyyy.JCLOUT,LINK=DDJCLOUT
/FILE DB.yyyyy.JCLCON,LINK=DDJCLCON
/EXEC (ADARUN,ADAvrs.MOD)
ADARUN PROG=ADARAI,DBID=yyyyy,MODE=MULTI
ADARUN UEX2=EXITR2
ADARAI RECOVER PLOGDEV=2201,FEOFPL=YES,RELGEN=1
/LOGOFF NOSPOOL
```

Skeleton Job Control

For a BS2000 system, the skeleton job control should have the following characteristics:

- The first two positions must be blank;
- Statements can be up to 256 characters long, *including* the blanks in the first two positions;
- Statements can be continued; the continuation mark (-) can be in any position (it does not have to be in position 74);
- In version 10, ADARAI automatically adds the command:

  ```
  /MODIFY-SDF-OPTIONS CONTINUATION=NEW-MODE
  ```

- Job control statements for the BS2000 executive are automatically broken up into 72-byte segments, if necessary. Normal program control statements are *not* segmented.
- Except for string substitution (described below), ADARAI does not check or change the job control statement syntax.

In addition to the general JCL characteristics described above, BS2000 systems can include the following in the skeleton JCL:

- Substitution strings for repeating often-used job control input;

- DDFILEA data set overwriting, tape volume assignment, and disk space credit;
- Subtask processing of RECOVER-generated console messages.

These options must be specified at the beginning of the skeleton JCL, immediately before the %%JCL-JOB-HEADER statement.

Specifying Substitution Strings

Frequently occurring strings such as data set names can be defined in the substitution section. The strings are then inserted into the BS2000 JCL as well as in the user program control statements.

To use string substitution, include the following JCL statement before %%JCL-JOB-HEADER:

```
%%JCL-BS2-SUBSTITUTION
```

This statement is followed by the substitution definitions, which have the following format:

```
%%argname=substring
```

where *argname* is the 1- to 8-character JCL or user program control argument to be replaced, and *substring* is the replacement string of up to 128 characters. One substitution statement per line is allowed; the substitution statements are ended by the next %%JCL statement.

DDFILEA JCL Options

The JCL statements described in this section must appear before %%JCL-JOB-HEADER in the skeleton JCL.

ADAORD is the only utility that opens DDFILEA output data sets during the BS2000 recovery job. ADARAI assists in allocating those data sets on disk or tape regardless of whether the data sets existed at RECOVER time.

Existing tape data sets are never overwritten. To overwrite existing DDFILEA disk data sets, specify the following statement in the skeleton JCL:

```
%%JCL-BS2-WORK-DATASET-OVERWRITE=YES
```

OVERWRITE defaults to NO. If no DDFILEA assignments are found and overwriting is prohibited (the default), ADARAI tries to write the DDFILEA data sets on TAPE-C1 cartridges.

To allocate DDFILEA on disk instead of tape, include the following JCL statement:

```
%%JCL-BS2-WORK-DISK-SPACE
```

Then specify one or more disk space assignment statements, as follows:

```
:catid :=pam-pages
```

ADARAI checks the availability of the specified pubsets and for permission of the ADARAI task's logon user ID to allocate the specified number of PAM pages on those pubsets. If the checks fail, the user's logon ID must be added to the joinfile of the related pubset, and the ADARAI RECOVER job step must be repeated before starting the generated recovery job. Otherwise, the ADAORD job steps may abend.

To assign DDFILEA output tape devices and volumes, include the following statement:

```
%%JCL-BS2-WORK-TAPE-VOLUMES=device-type
```

If output tape data sets must be created, ADARAI uses temporary data sets named #ADARAI.RECOVER.TAPE.*nnnnn*, where *nnnnn* equals "00001", "00002", and so on. If the console subtask option described in the next section is enabled, each line can contain one or more volser numbers separated by blanks or commas. The first two positions on each line must be blanks; the maximum line length is 256 characters.

The following is an example:

```
%%JCL-BS2-WORK-TAPE-VOLUMES=TAPE-C1
  A00001,A00002,A00003,A00004
  A00005,A00006,A00007,A00008
```

Up to 512 volumes can be specified.

BS2000 Console Subtask

When a recovery job created by ADARAI RECOVER is submitted, console messages may occur that require operator intervention, such as the following:

- The catalog entry for a tape data set is not available, and ADARAI inserts a /IMPORT-FILE statement for a temporary tape data set, causing a DMS0DA5 console message.
- Tape output for the DDFILEA data set is required, and ADARAI includes JCL that causes a message requesting that a scratch tape be mounted.

These messages can be answered automatically by a UCON program, which runs as a subtask in parallel with the recovery job. The UCON program (ADAR2C) receives all relevant console messages and sends answers whenever possible.

To run a console subtask, include the following section before %%JCL-JOB-HEADER:

```
%%JCL-BS2-CONSOLE-SUBTASK-SPECIFICATION
CONSOLE-NAME=name,C'password'      (this statement is required)
DCAM-APPL=dcamappl                              (this statement is optional)
```

where

name	is a /LOGON userid;
password	is the user ID's logon password; and
dcamappl	is the name of a DCAM application (the default is "RAIRUCON").

For more information, see the *Authorized User Tasks* section in the BS2000 Systems Administration documentation.

The following optional statement can be included in the skeleton JCL, followed by volser definitions. If the console subtask is not called, the volser definitions have no effect.

```
%%JCL-BS2-WORK-TAPE-VOLUMES=device-type
```

Include the following statement in the JCL for the ADARAI RECOVER function (*not* in the skeleton JCL):

```
/SET-FILE-LINK
FILE-NAME=console-job,LINK-NAME=DDJCLCON
```

The following example JCL for the RECOVER function includes the console subtask:

```
/  LOGON
/  SET-FILE-LINK  FILE-NAME=ADA.JCLIN,LINK-NAME=DDJCLIN
/  SET-FILE-LINK  FILE-NAME=ADA.JCLOUT,LINK-NAME=DDJCLOUT
/  SET-FILE-LINK  FILE-NAME=ADA.JCLCON,LINK-NAME=DDJCLCON
.
.

/  ASSIGN-SYSDTA  TO-FILE=*SYSCMD
/  SET-FILE-LINK  ADAvrs.MOD,LINK-NAME=DDLIB
/  START-PROG *M(E=ADARUN,L=ADAvrs.MOD)
ADARUN PROG=ADARAI,DB=47
ADARAI RECOVER
/  LOGOFF NOSPOOL
```

The console subtask evaluates the following BS2000 console messages, where *tsn* is the task serial number, and *mn* is the technical device name:

NKVT010 VOLUME volser IS MOUNTED ON DEVICE mn

If the volser and device are specified as described in the section **DDFILEA JCL Options**, they are registered in the subtask's online volser table. If the message NKVT013 is also outstanding at this time, the subtask returns the response "tsn.mn".

NKVT011/97 VOLUME volser IS DISMOUNTED FROM DEVICE mn...

The volser is removed from the online volser table.

NKVT013 MOUNT TAPE '*SCRAT' ON DEVICE ...

If this message is related to the RECOVER task and the volser is available in the online volser table, the subtask sends the response "tsn.mn".

DMS0DFB ACKNOWLEDGE VSN volser ON DEVICE mn...

If this message is related to the RECOVER task, the subtask sends the response "tsn".

DMS0DA5 INVALID FILE SPECIFICATION: VSN volser FOR FILE file...

If the related "tsn" is the RECOVER task's "tsn" and *file* is a temporary data set with the name #ADARAI.RECOVER.TAPE.nnnnn, the message is answered with the response "tsn.I".

For the subtask to respond, the logon ID must be able to ignore tapes with incorrect file IDs. To enable this, issue the following statement under the system administrator's logon ID (TSOS):

```
/MOD-USER userid,PROTECTION-ATTRIBUTE=(TAPE-ACCESS=READ)
```

Skeleton Job Control Example (BS2000)

```
%%JCL-BS2-WORK-DATASET-OVERWRITE = NO
%%JCL-BS2-WORK-DISK-SPACE
      :A: = 500000
      :B: = 1000000
%%JCL-BS2-CONSOLE-SUBTASK-SPECIFICATION
  CONSOLE-NAME=CON1,C'PASSWORD'
%%JCL-BS2-WORK-TAPE-VOLUMES=TAPE-C1
   A00001,A00002,A00003,A00004,A00005
   A00006,A00007,A00008,A00009,A00010
%%JCL-BS2-SUBSTITUTION
  %%USERID = ADAvrs
  %%AC  %%ASSOR1 = $ADAvrs.DByyyyy.ASSOR1
  %%ASSOR2 = $ADAvrs.DByyyyy.ASSOR2
```

```
  %%DATAR1 = $ADAvrs.DByyyyy.DATAR1
  %%DATAR2 = $ADAvrs.DByyyyy.DATAR2
  %%WORK   = $ADAvrs.DByyyyy.WORK
  %%SORT   = $ADAvrs.DByyyyy.SORT%
  %%TEMP   = $ADAvrs.DByyyyy.TEMP
  %%RLOGR1 = $ADAvrs.DByyyyy.RLOGR1
  %%PLOGR1 = $ADAvrs.DByyyyy.PLOGR1
  %%PLOGR2 = $ADAvrs.DByyyyy.PLOGR2
  %%DBID   = yyyyy
  %%IDTNAME = ADABASvB
  %%USEREX  = ADAvrs.USEREXITS.MOD
  %%DDLIB   = ADAvrs.MOD
  %%DEVICE  = dddd
%%JCL-JOB-HEADER
  /.RECOVER LOGON %%USERID,%%ACCOUNT
  /       SET-FILE-LINK FILE-NAME=%%USEREX,LINK-NAME=BLSLIB01
  /       SET-FILE-LINK FILE-NAME=%%DDLIB,LINK-NAME=DDLIB
  /       ASSIGN-SYSOUT TO-FILE=$ADAvrs.RAI.OUT
  /       ASSIGN-SYSLST TO-FILE=$ADAvrs.RAI.LST%%JCL-JOB-TRAILER
  /REMARK ------------------------------------------------------------*
  /REMARK *          RECOVERY JOB SUCCESSFULLY TERMINATED            *
  /REMARK ------------------------------------------------------------*
  /LOGOFF NOSPOOL
  /.JOBERROR REMARK
  /REMARK ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
  /REMARK *          RECOVERY JOB TERMINATED WITH ERROR              *
  /REMARK ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
  /LOGOFF NOSPOOL
%%JCL-ADARES
  /REMARK
  /REMARK REGENERATE/ BACKOUT
  /REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADARES,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADARES,LINK-NAME=DDPRINT
  %SEQUENTIAL
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADARES
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
```

```
  %KARTE
%%JCL-STEP-TRAILER
  /SET-JOB-STEP%%JCL-ADASAV
  /REMARK
  /REMARK RESTORE FILE(S)/DATABASE
  /REMARK
  /.%STEP REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK
FILE-NAME=RAI.DRU.JCLO.RESTORE,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK
FILE-NAME=RAI.PRI.JCLO.RESTORE,LINK-NAME=DDPRINT
  %SEQUENTIAL
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADASAV
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  %KARTE
%%JCL-ENDNUC
  /REMARK
  /REMARK ADADBS END NUCLEUS

/REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%SORT,LINK-NAME=DDSORTR1
  /       SET-FILE-LINK FILE-NAME=%%TEMP,LINK-NAME=DDTEMPR1
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADADBS,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADADBS,LINK-NAME=DDPRINT
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
```

```
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADADBS
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  ADADBS OPERCOM ADAEND
  /REMARK
  /REMARK CHECK INACTIVE DATABASE
  /REMARK
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADARAI,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADARAI,LINK-NAME=DDPRINT
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN
MODE=MULTI,PROG=ADARAI,DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  ADARAI CHKDB INACTIVE
%%JCL-ADADEF
  /REMARK
  /REMARK DEFINE NEWWORK
  /REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%SORT,LINK-NAME=DDSORTR1
  /       SET-FILE-LINK FILE-NAME=%%TEMP,LINK-NAME=DDTEMPR1
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADADEF,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADADEF,LINK-NAME=DDPRINT
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADADEF
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  %KARTE
%%JCL-ADALOD
  /REMARK
  /REMARK LOAD A FILE/ MASS UPDATE
  /REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
```

```
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%SORT,LINK-NAME=DDSORTR1
  /       SET-FILE-LINK FILE-NAME=%%TEMP,LINK-NAME=DDTEMPR1
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADALOD,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADALOD,LINK-NAME=DDPRINT
  %SEQUENTIAL
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADALOD
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  %KARTE
%%JCL-ADAORD
  /REMARK
  /REMARK REORDER
  /REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%SORT,LINK-NAME=DDSORTR1
  /       SET-FILE-LINK FILE-NAME=%%TEMP,LINK-NAME=DDTEMPR1
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADAORD,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADAORD,LINK-NAME=DDPRINT
  %SEQUENTIAL
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADAORD
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  %KARTE
%%JCL-ADAINV
  /REMARK
  /REMARK INVERT/ COUPLE
  /REMARK
  /       SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
```

```
  /       SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,       -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,         -
  /                     SUP=DISK(SHARE-UPD=YES)
  /       SET-FILE-LINK FILE-NAME=%%SORT,LINK-NAME=DDSORTR1
  /       SET-FILE-LINK FILE-NAME=%%TEMP,LINK-NAME=DDTEMPR1
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADADBS,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADADBS,LINK-NAME=DDPRINT
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADAINV
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  %KARTE
%%JCL-STARTNUC
  /REMARK
  /REMARK START NUCLEUS
  /REMARK
  /       MODIFY-JOB-SWITCHES ON=(4,5)
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM FROM-FILE=EDT
  :/.ADANUC LOGON %%USERID,%%ACCOUNT
  :/    SET-FILE-LINK FILE-NAME=%%RLOGR1,LINK-NAME=DDRLOGR1,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%PLOGR1,LINK-NAME=DDPLOGR1,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%PLOGR2,LINK-NAME=DDPLOGR2,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%ASSOR1,LINK-NAME=DDASSOR1,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%ASSOR2,LINK-NAME=DDASSOR2,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%DATAR1,LINK-NAME=DDDATAR1,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%DATAR2,LINK-NAME=DDDATAR2,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=%%WORK,LINK-NAME=DDWORKR1,-
  :/                  SUP=DISK(SHARE-UPD=YES)
  :/    SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADANUC,LINK-NAME=DDDRUCK
  :/    SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADANUC,LINK-NAME=DDPRINT
  :/    ASSIGN-SYSDTA TO-FILE=*SYSCMD
  :/    SET-FILE-LINK FILE-NAME=%USEREX,LINK-NAME=BLSLIB01
  :/    SET-FILE-LINK FILE-NAME=%%DDLIB,LINK-NAME=DDLIB
  :/    START-PROGRAM FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  :ADARUN PROG=ADANUC
  :ADARUN DATABASE=%%DBID           DATA BASE ID
  :ADARUN IDTNAME=%%IDTNAME                NAME OF IDT
  :ADARUN DEVICE=%%DEVICE
```

```
  :ADARUN IDTNAME=%%IDTNAME
  :ADARUN LFIOP=1000000,LWP=800000,LBP=4000000,LU=64000
  :ADARUN LP=200,LS=20000,TT=900,TNAE=900,OPENRQ=NO,PLOGRQ=YES
  :ADARUN NAB=20,NH=400,NU=200,NISNHQ=50,NC=20,NPLOG=2,PLOGDEV=2000
  :ADARUN PLOGSIZE=1000,NCLOG=2,CLOGDEV=2000,CLOGSIZE=500
  :ADARUN UEX2=USEREX2
  :/LOGOFF SYSOUT=DEL
  @D &:1-1:
  @W '#NUCENT' 0
  @H
  /STEP
  /ENTER-JOB #NUCENT,PROC-ADMIS=PAR(USER-ID=%%USERID,                -
  /                                 ACCOUNT=%%ACCOUNT)
  /STEP
  /       SET-FILE-LINK FILE-NAME= %%DDLIB,LINK-NAME=DDLIB
  /       SET-FILE-LINK FILE-NAME=RAI.DRU.JCLO.ADARES,LINK-NAME=DDDRUCK
  /       SET-FILE-LINK FILE-NAME=RAI.PRI.JCLO.ADARES,LINK-NAME=DDPRINT
  /       ASSIGN-SYSDTA TO-FILE=*SYSCMD
  /       START-PROGRAM
FROM-FILE=*MODULE(ELEMENT=ADARUN,LIBRARY=%%DDLIB)
  ADARUN MODE=MULTI,PROG=ADARAI
  ADARUN DBID=%%DBID,DE=%%DEVICE,IDTNAME=%%IDTNAME
  ADARAI CHKDB ACTIVE
```

ADAR2E Utility

If a pubset member is lost because of, for example, a head crash or other unrecoverable hardware error, initialize a new member, include it in the pubset, and restore all data sets in the pubset. In most cases, you must also reallocate the ASSO, DATA, WORK, and PLOG data sets on exactly the same disk locations they were on before the hardware failure.

The ADAR2E utility reads the TSOSCAT entries of assigned data sets and creates a procedure that reallocates the Adabas files at exactly the same disk locations where they were before. The ADAR2E utility operates totally independently of ADARAI and other Adabas utilities; ADAR2E should be run whenever disk allocation changes are made to the major Adabas components (ASSO, DATA, WORK, PLOG, CLOG, TEMP, or SORT).

The following example JCL is for running the ADAR2E utility:

```
/  LOGON
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.ASSO-01,LINK-NAME=DDASSOR1
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.DATA-01,LINK-NAME=DDDATAR1
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.WORK-01,LINK-NAME=DDWORKR1
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.TEMP-01,LINK-NAME=DDTEMPR1
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.PLOGR1,LINK-NAME=DDPLOGR1
/  SET-FILE-LINK FILE-NAME=DB.yyyyy.PLOGR2,LINK-NAME=DDPLOGR2
/  SET-FILE-LINK FILE-NAME=ALLOCATE.JOB,LINK-NAME=DDJCLOUT
/  START-PROGRAM *MOD(LIB=ADAvrs,ELEM=ADAR2E)
/  LOGOFF SYS-OUTPUT=DEL
```

The following is a list of available link names:

```
DDASSOR1,DDASSOR2,DDASSOR3,DDASSOR4,DDASSOR5
DDDATAR1,DDDATAR2,DDDATAR3,DDDATAR4,DDDATAR5
DDWORKR1,DDWORKR2
DDSORTR1,DDSORTR2
DDTEMPR1,DDTEMPR2
DDPLOGR1,DDPLOGR2,DDPLOGR3,DDPLOGR4,DDPLOGR5,DDPLOGR6,DDPLOGR7,DDPLOGR8
```

The following is an example of the job control created by the ADAR2E utility:

```
/  BEGIN-PROCEDURE
/  MODIFY-SDF-OPTIONS CONTINUATION=NEW-MODE
/  CREATE-FILE FILE-NAME=:A:$ADABAS.DB.yyyyy.ASSO-01,-
/       SUPPORT=PUBLIC-DISK(VOLUME=PUBA00,-
/       DEVICE-TYPE=Ddddd,SPACE=ABSOLUTE-
/       (FIRST-PAGE=32833,SIZE=5001))
/  MODI-FILE-ATTR FILE-NAME=:A:$ADABAS.DB.yyyyy.ASSO-01,-
/       SUPPORT=PUBLIC-DISK(VOLUME=PUBA00,-
/       DEVICE-TYPE=Ddddd,SPACE=ABSOLUTE-
/       (FIRST-PAGE=29761,SIZE=5001))
        .
        .
/  CREATE-FILE FILE-NAME=:A:$ADABAS.DB.yyyyy.DATA-01,-
/       SUPPORT=PUBLIC-DISK(VOLUME=PUBA02,-
/       DEVICE-TYPE=Ddddd,SPACE=ABSOLUTE-
/       (FIRST-PAGE=119809,SIZE=8001))
/  MODI-FILE-ATTR FILE-NAME=:A:$ADABAS.DB.yyyyy.DATA-01,-
/       SUPPORT=PUBLIC-DISK(VOLUME=PUBA03,-
/       DEVICE-TYPE=Ddddd,SPACE=ABSOLUTE-
/       (FIRST-PAGE=75841,SIZE=8001))
```

Execute the ADAR2E-generated reallocation procedure under the system administrator's logon ID (TSOS) before starting the ADARAI RECOVER function.

Under BS2000 version 11 and above, it is possible to allow all users with access rights to a certain pubset to allocate disk space absolutely by issuing the following command under TSOS:

```
/MODIFY-MASTER-CATALOG CATID= catid,PHYSICAL-ALLO=USER-ALLOWED
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data protection log	DDPLOGRn	tape/ disk	Required for RECOVER only if FEOFPL=YES is specified.
Recovery log (RLOG)	DDRLOGR1	disk	
Job stream input	DDJCLIN	disk	Required only for RECOVER
Recovery job output	DDJCLOUT	disk	Required only for RECOVER
ADARAI messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations*
ADARAI parameters	DDKARTE	reader	

JCL Examples (z/OS)

Prepare for Recovery Logging (ADARAI PREPARE):

```
//RAIPREP   EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOADLIB
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.ADAyyyyy.ASSOR1
//DDRLOGR1  DD   DISP=SHR,DSN=EXAMPLE.ADAyyyyy.RLOGR1
//DDDRUCK   DD   SYSOUT=A
//DDPRINT   DD   SYSOUT=A
//DDCARD    DD   *
ADARUN PROGRAM=ADARAI,SVC=xxx,DEVICE=dddd,DBID=yyyyy,MODE=MULTI
//DDKARTE   DD   *
ADARAI PREPARE RLOGSIZE=5,MINGENS=5
```

List the RLOG (ADARAI LIST)

```
//RAILIST  EXEC   PGM=ADARUN
//STEPLIB  DD DSN=ADABAS.ADAvrs.LOADLIB,DISP=SHR
//DDASSOR1 DD DSN=EXAMPLE.ADAyyyyy.ASSOR1,DISP=SHR
//DDRLOGR1 DD DSN=EXAMPLE.ADAyyyyy.RLOGR1,DISP=SHR
//DDDRUCK  DD      SYSOUT=A
//DDPRINT  DD      SYSOUT=A
//DDCARD   DD      *
ADARUN PROGRAM=ADARAI,SVC=xxx,DEVICE=dddd,DBID=yyyyy,MODE=MULTI
//DDKARTE  DD      *
ADARAI LIST
/*
```

Create Recovery JCL (ADARAI RECOVER)

```
//ADARAI   EXEC   PGM=ADARUN
//STEPLIB  DD DSN=ADABAS.ADAvrs.LOADLIB,DISP=SHR
//*
//DDASSOR1 DD DSN=EXAMPLE.ADAyyyyy.ASSOR1,DISP=SHR
//*
//DDRLOGR1 DD DSN=EXAMPLE.ADAyyyyy.RLOGR1,DISP=SHR
//DDJCLIN  DD DSN=EXAMPLE.ADAyyyyy.RAIJCL(JCLIN),DISP=SHR
//DDJCLOUT DD      SYSOUT=A
//*
//DDPLOGR1 DD DSN=EXAMPLE.ADAyyyyy.PLOGR1,DISP=SHR
//DDPLOGR2 DD DSN=EXAMPLE.ADAyyyyy.PLOGR2,DISP=SHR
//*
//DDDRUCK  DD      SYSOUT=A
//DDPRINT  DD      SYSOUT=A
//DDCARD   DD      *
ADARUN PROGRAM=ADARAI,SVC=xxx,DEVICE=dddd,DBID=yyyyy,MODE=MULTI
ADARUN UEX2=UEX2
//DDKARTE  DD      *
ADARAI RECOVER JCLLOG=YES,RELGEN=0,DRIVES=2
```

Skeleton Job Control Example (z/OS)

This example can be found in member ADARAIIN of the JOBS data set.

```
%%JCL-JOB-HEADER
  //ADARECOV JOB 5,'ADA-USER',MSGCLASS=X,CLASS=A,REGION=4096K
  //*
  //JOBLIB DD DSN=ADABAS.ADAvrs.LOAD,DISP=SHR
  //*
%%JCL-JOB-TRAILER
  //*
  //* END OF RECOVERY
  //*
  //
%%JCL-STEP-TRAILER
  //*
  //* END OF STEP
  //*
%%JCL-DDKARTE
  //DDKARTE  DD  *
%%JCL-ADADEF
  //*
  //* DEFINE NEWWORK
  //*
  //%STEP   EXEC PGM=ADARUN          ADADEF JOB STEP
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
```

```
  //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
  //DDTEMPR1 DD  DSN=ADABAS.ADAvrs.TEMPR1,DISP=OLD
  //DDSORTR1 DD  DSN=ADABAS.ADAvrs.SORTR1,DISP=OLD
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADADEF,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-ADAINV
  //*
  //* INVERT / COUPLE
  //*
  //%STEP   EXEC PGM=ADARUN          ADAINV JOB STEP
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //DDSORTR1 DD  DSN=ADABAS.ADAvrs.SORTR1,DISP=OLD
  //DDTEMPR1 DD  DSN=ADABAS.ADAvrs.TEMPR1,DISP=OLD
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADAINV,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-ADALOD
  //*
  //* LOAD A FILE / MASS UPDATE
  //*
  //%STEP   EXEC PGM=ADARUN          ADALOD JOB STEP
  %SEQUENTIAL
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
  //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
  //DDTEMPR1 DD  DSN=ADABAS.ADAvrs.TEMPR1,DISP=OLD
  //DDSORTR1 DD  DSN=ADABAS.ADAvrs.SORTR1,DISP=OLD
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADALOD,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-ADAORD
  //*
  //* REORDER
  //*
  //%STEP   EXEC PGM=ADARUN          ADAORD JOB STEP
  %SEQUENTIAL
```

```
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
  //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADAORD,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-ADARES
  //*
  //* PLCOPY / REGENERATE / BACKOUT
  //*
  //%STEP   EXEC PGM=ADARUN
  %SEQUENTIAL
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //*  Omit DDPLOGRx for Parallel and Cluster Services
  //DDPLOGR1 DD  DSN=ADABAS.ADAvrs.PLOGR1,DISP=SHR
  //DDPLOGR2 DD  DSN=ADABAS.ADAvrs.PLOGR2,DISP=SHR
  //*  Include MERGIN1 and MERGIN2 for Parallel and Cluster Services
  //MERGIN1 DD  DSN=ADABAS.ADAvrs.INTERI,DISP=SHR
  //MERGIN2 DD  DSN=ADABAS.ADAvrs.INTERO,DISP=SHR
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADARES,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-DDSIAUS1
  //DDSIAUS1 DD  DSN=ADABAS.ADAvrs.PLCOPY(+1),
  //             DISP=(NEW,CATLG),UNIT=TAPE
%%JCL-DDSIAUS2
  //DDSIAUS2 DD  DSN=ADABAS.ADAvrs.PLCOPY2(+1),     Optional second copy
  //             DISP=(NEW,CATLG),UNIT=TAPE
%%JCL-ADASAV
  //*
  //* RESTORE FILE(S)/DATABASE
  //*
  //%STEP   EXEC PGM=ADARUN          ADASAV JOB STEP
  %SEQUENTIAL
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
  //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
  //DDTEMPR1 DD  DSN=ADABAS.ADAvrs.TEMPR1,DISP=OLD
  //SYSUDUMP DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
```

```
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROG=ADASAV,DBID=xxxxx,DE=yyyy,SVC=zzz
  //DDKARTE  DD  *
  %KARTE
%%JCL-STARTNUC
  //*
  //* START NUCLEUS
  //*
  //%STEP   EXEC PGM=IEBGENER
  //SYSPRINT DD  SYSOUT=*
  //SYSIN    DD  DUMMY
  //SYSUT2   DD  SYSOUT=(*,INTRDR)
  //SYSUT1   DD  DATA,DLM='$$'
  //ADANUC   JOB 5,'ADANUC',CLASS=A,MSGCLASS=X,REGION=6M,TIME=1440
  //NUCxxxx EXEC PGM=ADARUN
  //STEPLIB  DD  DSN=ADABAS.ADAvrs.LOAD,DISP=SHR
  //*
  //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
  //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
  //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
  //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
  //DDPLOGR1 DD  DSN=ADABAS.ADAvrs.PLOGR1,DISP=SHR
  //DDPLOGR2 DD  DSN=ADABAS.ADAvrs.PLOGR2,DISP=SHR
  //INTRDR2  DD  SYSOUT=(*,INTRDR)
  //SYSUDUMP DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN PROG=ADANUC
  ADARUN UEX2=UEX2
  ADARUN DUALPLS=nnn            dual PLOG size
  ADARUN DUALPLD=mmmm           dual PLOG device type
  ADARUN MODE=MULTI
  ADARUN DATABASE=xxxxx         DATA BASE ID
  ADARUN DEVICE=yyyy            ASSOCIATOR DEVICE TYPE
  ADARUN SVC=zzz                SVC NUMBER
  $$
  //*
  //* End of input for internal reader. Check whether nucleus is active.
  //*
  //ADARAI  EXEC PGM=ADARUN
  //STEPLIB  DD  DSN=ADABAS.ADAvrs.LOAD,DISP=SHR
  //SYSOUT   DD  SYSOUT=*
  //DDPRINT  DD  SYSOUT=*
  //DDDRUCK  DD  SYSOUT=*
  //DDCARD   DD  *
  ADARUN MODE=MULTI,PROGRAM=ADARAI,DBID=xxxxx,SVC=zzz,DE=yyyy
  //DDKARTE  DD  *
  ADARAI CHKDB ACTIVE
  //*
%%JCL-ENDNUC
  //*
  //* ADADBS END NUCLEUS
```

```
 //*
 //%STEP   EXEC PGM=ADARUN
 //DDASSOR1 DD  DSN=ADABAS.ADAvrs.ASSOR1,DISP=SHR
 //DDDATAR1 DD  DSN=ADABAS.ADAvrs.DATAR1,DISP=SHR
 //DDWORKR1 DD  DSN=ADABAS.ADAvrs.WORKR1,DISP=SHR
 //DDTEMPR1 DD  DSN=ADABAS.ADAvrs.TEMPR1,DISP=OLD
 //DDSORTR1 DD  DSN=ADABAS.ADAvrs.SORTR1,DISP=OLD
 //DDRLOGR1 DD  DSN=ADABAS.ADAvrs.RLOGR1,DISP=SHR
 //DDDRUCK  DD  SYSOUT=*
 //DDPRINT  DD  SYSOUT=*
 //DDCARD   DD  *
 ADARUN MODE=MULTI,PROG=ADADBS,DBID=xxxxx,DE=yyyy,SVC=zzz
 //DDKARTE  DD  *
 ADADBS OPERCOM ADAEND
//*
//* Check whether nucleus is inactive.
//*
//ADARAI  EXEC PGM=ADARUN
//STEPLIB  DD  DSN=ADABAS.ADAvrs.LOAD,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//DDPRINT  DD  SYSOUT=*
//DDDRUCK  DD  SYSOUT=*
//DDCARD   DD  *
ADARUN MODE=MULTI,PROGRAM=ADARAI,DBID=xxxxx,SVC=zzz,DE=yyyy
//DDKARTE  DD  *
ADARAI CHKDB INACTIVE
//* ↩
```

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	Required for RECOVER
Work	DDWORKR1 DDWORKR4	disk	Required for RECOVER
Data protection log	DDPLOGRn	tape/ disk	Required for RECOVER
Recovery log (RLOG)	DDRLOGR1	disk	
Job stream input	DDJCLIN	disk	Required for RECOVER
Recovery job output	DDJCLOUT	disk	Required for RECOVER
ADARAI messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADARAI parameters	DDKARTE	disk/ terminal/ reader	

JCL Examples (z/VM)

Start Recovery Logging (ADARAI PREPARE)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDRLOGR1,DSN=ADABASVv.RLOG1,VOL=RLOGV1
DATADEF DDDRUCK,DSN=ADARAI.DDDRUCK,MODE=A
DATADEF DDPRINT,DSN=ADAFRM.DDPRINT,MODE=A
DATADEF DDCARD,DSN=RUNRAI.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADARAI.CONTROL,MODE=A
ADARUN
```

Contents of RUNRAI CONTROL A1:

```
ADARUN PROG=ADARAI,DEVICE=dddd,DB=99,MODE=MULTI
```

Contents of ADARAI CONTROL A1:

```
ADARUN PREPARE RLOGSIZE=5,RLOGDEV=eeee,MINGENS=5
```

List The RLOG (ADARAI LIST)

```
DATADEF ASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF RLOGR1,DSN=ADABASVv.RLOG1,VOL=RLOGV1
DATADEF DDDRUCK,DSN=ADARAI.DDDRUCK,MODE=A
DATADEF DDPRINT,DSN=ADARAI.DDPRINT,MODE=A
DATADEF DDCARD,DSN=RUNRAI.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADARAI.CONTROL,MODE=A
```

Contents of RUNRAI CONTROL A1:

```
ADARUN
PROG=ADARAI,DEVICE=dddd,DB=yyyyy,MODE=MULTI
```

Contents of ADARAI CONTROL A1:

```
ADARUN LIST
```

VSE

The following functions are available for use with ADARAI on a VSE system:

Function	Action
CHKDB	check the database status
DISABLE	disable the recovery log (RLOG)
LIST	list the RLOG contents
PREPARE	start the RLOGs
REMOVE	remove the RLOG

The ADARAI RECOVER function used to rebuild the job stream is not currently supported on a VSE system.

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	See note 1	
Data Storage	DATARn	disk	See note 1	Required for RECOVER
Work	WORKR1	disk		Required for RECOVER
Recovery log (RLOG)	RLOGR1	disk		
Data protection log	PLOGRn	tape disk		Required for RECOVER
Job stream input	JCLIN	SYSIPT		Required for RECOVER
Recovery job output	JCLOUT	SYSPCH		Required for RECOVER
ADARAI messages		printer	SYS009	*Messages and Codes*
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 See note 1	
ADARAI parameters		reader	SYSIPT	

Note: Any programmer logical unit may be used.

Example JCS (VSE)

Start Recovery Logging (ADARAI PREPARE)

```
// ASSGN SYS009,00F
// EXEC PROC=ADAVvFIL
// EXEC PROC=ADAVvLIB
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARAI,SVC=xxx,DEVICE=dddd,DB=yyyyy,MODE=MULTI
/*
ADARAI PREPARE RLOGSIZE=5,MINGENS=5
/*
```

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

List the RLOG (ADARAI LIST)

```
// ASSGN SYS009,00F
// EXEC PROC=ADAVvFIL
// EXEC PROC=ADAVvLIB
// ASSGN SYS000,SYSIPT
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARAI,SVC=xxx,DEVICE=dddd,DB=yyyyy,MODE=MULTI
/*
ADARAI LIST
/*
```

XVII ADAREP Utility: Database Status Report

This chapter covers the following topics:

Functional Overview
Report Syntax
Processing Save Tape Input
Report Description
JCL/JCS Requirements and Examples

132 Functional Overview

The ADAREP utility produces the database status report, which provides information concerning the current physical layout and logical contents of a database or a save tape (if the SAVETAPE parameter is specified).

The information provided in the database status report includes

- database name, number, creation date/time, file status, and current log number.
- in cylinders and blocks, the amount and locations of Associator, Data, and Work space currently used, and allocated but unused.
- Associator and Data Storage RABN information including device type, VOLSER number, file number (if appropriate), and usage (AC, NI/UI, Data Storage, DSST, alternate (only from a save tape), or unused).
- alternate RABN block amounts and locations (only from a save tape).
- by file, a summary of ISN, extent, padding factor, used/unused Associator and Data Storage space, and file options.
- detailed information (optionally by file) that includes all summary information plus MIN-ISN/MAXISN settings, MUPEX settings, detailed space information, creation and last use date/time, field definition table (FDT) contents, and general or extended checkpoint file information.

133 Report Syntax

- Optional Parameters 802
- Examples 805

```
ADAREP  [REPORT]
        [ {CPEXLIST [, OFFSET = {column-number |  63 } ]  |  CPLIST }
              [FROMDATE = yyyymmdd ]  [TODATE =  yyyymmdd ]
              [FROMSESSION = session ]  [TOSESSION = session ] ]
        [ {FILE = { file-list  |  all-files } [, LAYOUT = 1 ]  |  NOFILE } ]
        [LIMCOUNT | NOCOUNT]
        [NOFDT]
        [ { NOLGLIST | NOPHLIST | NOSTD } ]
        [NOUSERABEND] }
        [SAVETAPE
              [PLOGNUM = protection-log-number  [ { SYN1 | SYN4 } = PLOG-block-number ] ] ]
```

This chapter describes the syntax and parameters of the ADAREP utility.

Optional Parameters

ADAREP can be specified alone to retrieve a database status report. You can optionally customize the report by added parameter values.

CPEXLIST: Print Checkpoint List in Extended Format
CPLIST : Print Checkpoint List in Normal Format

These parameters are used to print the checkpoint list in normal (CPLIST) or extended (CPEXLIST) format. Either CPEXLIST or CPLIST must be specified to display checkpoint information. CPEXLIST adds the following information to the normal CPLIST information, depending on the checkpoint origin:

- (Utility or Adabas Online System/Basic Services) function name;
- Checkpoint-specific data.

If the CPEXLIST report is to be displayed, OFFSET can also be specified for a more readable display.

The FROMDATE, TODATE, FROMSESSION, and TOSESSION parameters may be used to indicate the range of checkpoints to be printed.

FILE or NOFILE: File Information to be Displayed or Suppressed

FILE defines the list of files for which status information is to be printed or displayed. If this parameter is omitted, status information for all files will be included.

If NOFILE is specified, the printing of all file and field description information is suppressed.

FROMDATE/ TODATE: Start/ End Checkpoint Dates for Report

When CPLIST or CPEXLIST is specified, specific start and/or end dates for checkpoint information can be specified. Examples of valid *yyyymmdd* date specification are:

```
ADAREP FROMDATE=19960101,TODATE=19960228    January 1-February 28, 1996
ADAREP FROMDATE=19951111            November 11, 1995 to checkpoint file end
ADAREP TODATE=19951223            From checkpoint file begin to (and including)
                                        December 23, 1995
```

If FROMDATE is not specified, the report begins with the earliest checkpoint information in the system (or with the first on the FROMSESSION session, if later); if TODATE is not specified, the report continues up to the most recent checkpoint (or ends with the last on the TOSESSION session, if earlier).

FROMSESSION/ TOSESSION: Start/ End Session for Report

Specify a start and/or end session number. Sessions before FROMSESSION and/or after TOSESSION session numbers are not included in the report information. If FROMSESSION is not specified, the report begins with the earliest checkpoint information in the system (or with the first on the FROMDATE date, if later); if TOSESSION is not specified, the report continues up to the last checkpoint (or ends with the last on the TODATE date, if earlier).

LAYOUT: Format Output for Printing

LAYOUT=1 specifies that the "Contents of Database" table should be printed in a single 120-character column format. Normally, the "Contents of Database" and "File Space Allocations" information are presented in two separate sections in the report. However, when LAYOUT=1 is specified, they are merged together into the "Contents of Database" section. In addition, the padding factor is added as well.

The following is an example of how the "Contents of Database" section of the report would appear if LAYOUT=1 is specified. You compare this sample to the "Contents of Database" and "File Space Allocations" sections described in ***Contents of the Database: General File Status*** and in ***File Space Allocations***, elsewhere in this section.

```
*********************************************                              ↩
* Contents of Database    68 (EXAMP68         ) *         yyyy-mm-dd  hh:mm:ss ↩
*********************************************                              ↩
                                                                           ↩
 File     Name           Loaded      TOP-ISN    MAX-ISN   PADD              ↩
Blocks Allocated      (Blocks Unused)
                                                        A% D%          NI    ↩
   UI         AC       DATA/CYL                                            ↩
                                                                           ↩
    1 EMPLOYEES        2001-12-28       1107       1695  10 10        200    ↩
   15          2        150/1
    1   Extents: NI    1  UI    1  AC    1  DS    1                  138    ↩
    2                   104/0
    2 MISCELLANEOUS    2001-12-28       1779       2543  10 10        200    ↩
   15          3        150/1
    2   Extents: NI    1  UI    1  AC    1  DS    1                  174    ↩
   10                    97/0
    3 VEHICLES         2001-12-28        773       1695  10 10        200    ↩
   15          2        150/1
    3   Extents: NI    1  UI    1  AC    1  DS    1                  183    ↩
    8                   138/0
```

LIMCOUNT or NOCOUNT: Counting of Number of Records Loaded

ADAREP reads the address converter to determine the value for RECORDS LOADED for a file. For very large files, this can result in a large amount of I/O activity. If LIMCOUNT is specified, ADAREP checks the value for TOPISN for the file. If TOPISN is greater than 1000, "NOT COUNTED" appears under RECORDS LOADED.

If NOCOUNT is specified, no value is printed for RECORDS LOADED for any file. If neither LIMCOUNT nor NOCOUNT are specified, ADAREP compiles the exact value for RECORDS LOADED for each file.

NOFDT: Suppress Printing of Field Definitions

The printing of the field definition table (FDT) information for each file is to be omitted. The FDT of the Adabas checkpoint and security files are not printed by ADAREP.

NOLGLIST, NOPHLIST, or NOSTD: Suppress Database Layout Printing

If NOLGLIST is specified, the logical database layout information is to be omitted. If NOPHLIST is specified, physical database layout information is omitted. Specifying NOSTD suppresses all database layout information, and is equivalent to specifying NOLGLIST and NOPHLIST.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OFFSET: Defines Extended Report Line Alignment

The OFFSET parameter aligns the beginning of the extended information with the end of the normal checkpoint information for printing on a single line. The default is 63. To display extended information, OFFSET must be reduced or the extended portion may not be displayable within 80 columns.

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the nucleus protection log used during the ADASAV save operation. The number of the nucleus protection log is supplied on the tape, so specifying a value for the PLOGNUM parameter *overrides* the information on the tape.

SAVETAPE: Print Save Tape Report

If SAVETAPE is specified, the report is printed from a save tape. The purpose of the save tape report is to determine what the save tape contains. For more information, see the section **Processing Save Tape Input**.

CPLIST/ CPEXLIST cannot be specified with SAVETAPE. NOCOUNT must be specified with SAVETAPE because ADAREP does not count the number of records in a file on a save tape. If CPLIST/ CPEXLIST is specified or NOCOUNT is not, ADAREP prints a warning message, changes these options internally, and continues processing.

SYN1/SYN4: Beginning Block Number

The SYN1 and SYN4 parameters are mutually exclusive and specify the block number containing the SYN1 or SYN4 checkpoint at which the corresponding save operation began. These block numbers are supplied on the tape, as appropriate, so specifying a value for the SYN1 or SYN4 parameters *overrides* the information on the tape.

Examples

Example 1:

```
ADAREP REPORT
```

All database, file, and field information is to be printed. Checkpoint data is not to be printed.

Example 2:

```
ADAREP REPORT NOFDT
```

Database and file information is to be printed. Checkpoint data and field definitions for each file are not to be printed.

Example 3:

```
ADAREP CPEXLIST,FROMDATE=19980701,TODATE=19980715
ADAREP NOSTD,NOFILE
```

A checkpoint list (extended format) is to be printed for all checkpoints taken between July 1, 1998 and July 15, 1998. No other information is to be printed.

Example 4:

```
ADAREP REPORT SAVETAPE,NOCOUNT
```

All database, file, and field information is to be printed from a save tape. NOCOUNT is required because ADAREP does not count the number of records in a file on a save tape.

134 Processing Save Tape Input

- Supplying Protection Log Input 808
- Checking Input Tapes 809
- Concurrent Parameters 809
- Reports for Delta Save Tapes 809
- Report Layout 1020

If the SAVETAPE keyword is specified, the report is printed from a save tape.

The save tape

- must have been created using ADASAV version 7.1 or above;
- may have been created online or offline;
- may be a database save, file save, or delta save tape; and
- must be supplied as a DD/SAVE sequential input file.

ADAREP does not scan the complete save tape: normally, it is sufficient to supply only the first cassette/tape reel.

ADAREP reads through the save tape to pick up the general control blocks (GCBs), the block of unreadable blocks (BUB), the mirror table, the mirror BUB, the free space table (FST), and all the file control blocks (FCBs). Once these are in main memory, ADAREP continues as for a normal database report. The file definition tables (FDTs) are read from the save tape as they are needed: they are not buffered in main memory.

Note: Adabas versions 7.2 and above do not support and therefore do not save BUB or mirror information. However, BUB and mirror information on save tapes from Adabas 7.1 is still reported.

Supplying Protection Log Input

If an online save tape is used, the corresponding protection log may optionally be specified as a DD/PLOG sequential input file:

- If DD/PLOG is supplied, ADAREP scans the protection log for FCB and FST blocks to ensure that it has the most recent versions.
- If DD/PLOG is not supplied, ADAREP prints a warning message and continues. It displays the database status as of the beginning of the online save operation (time of SYN1/SYN4 checkpoint). Any secondary extents allocated during the online save operation are not reflected in the report because they are only visible on the protection log. In addition, the physical layout section may report phantom errors due to inconsistency in the FCB and FST blocks on the save tape. This happens only if a secondary extent was allocated during the short phase when ADASAV was saving the FCB and FST blocks.

Parameters PLOGNUM and SYN1 or SYN4 identify the protection log number and block number of the SYN1 or SYN4 checkpoint. The information provided by these parameters is supplied on the tape, so specifying PLOGNUM or the SYN1 or SYN4 parameters *overrides* the information on the tape.

When DD/PLOG is supplied, two tape units are needed in parallel: it is not possible to concatenate the save tape and the protection log as for ADASAV RESTONL.

Checking Input Tapes

After opening the DD/SAVE and DD/PLOG input data sets, ADAREP cross-checks to ensure that the input tapes are correct:

- If an invalid save tape is supplied, ADAREP terminates and displays error-128 (invalid save tape supplied).
- If an invalid protection log is supplied, ADAREP displays an appropriate warning message, sets the condition code to 4, and continues.

Concurrent Parameters

CPLIST/ CPEXLIST information and the number of records loaded for a file cannot be printed from the save tape. If the CPLIST/ CPEXLIST parameter is specified or the NOCOUNT parameter is not specified with SAVETAPE, ADAREP prints a warning message, changes these options internally, and continues processing.

If the save tape was created using ADASAV version 5.3.2 or above, the VOLSER number is printed on the report. For save tapes created using earlier versions of ADASAV, asterisks are printed as VOLSER numbers.

Reports for Delta Save Tapes

For delta save tapes, much of the information is either inaccessible or must be reconstructed:

- The delta save status is always *enabled*; the DLOG area usage is only one block (the header) which is displayed as "n%".
- The last full save number, last delta save number, and the date/time of the last delta save are taken from the DSID.
- The estimated number of changed blocks is MAXFILES times 5 plus 30 rounded to the next multiple of 100.
- The DLOG area location is derived from the GCBs.
- The date/time of last full save cannot be reconstructed and is always displayed as "unknown".

Report Layout

The purpose of the save tape report is to determine what the save tape contains.

The save tape report is preceded by a short header indicating the kind of save tape supplied, whether it was created online or offline, when it was created, the version of ADASAV used to create it, the database ID on the save tape, and possibly the delta save identifier of the save tape. For online save tapes, the session number of the corresponding protection log and the block number of the SYN1/SYN4 checkpoint (either supplied or derived from the tape) is displayed.

```
A D A R E P    Vv.r   SMs     DBID = nnnnn  STARTED           yyyy-mm-dd   hh:mm:SS ↩


PARAMETERS:
-----------

ADAREP REPORT SAVETAPE

************************************************************
*                                                          *
*  REPORT FROM            ONLINE DATABASE SAVE             *
*  CREATED AT             yyyy-mm-dd  hh:mm:ss             *
*  BY ADASAV VERSION      V vr                             *
*  DBIB                   nnnnn                            *
*  DSID                   1 / 0 /   yyyy-mm-dd  hh:mm:ss   *
*  PLOG SESSION NR        17                               *
*  SYN1 BLOCK NR          137                              *
*                                                          *
************************************************************

**********************
*                     *
*  DATA BASE REPORT   *                                    yyyy-mm-dd   hh:mm:ss ↩

*                     *
**********************
```

The database ID printed in the first line of the report is taken from the ADARUN DBID parameter. This DBID is *not* cross-checked with the database ID on the save tape. Instead, the save tape DBID is used throughout the report once the save tape is opened and the GCBs read.

The physical layout for file save reports is a table of RABN ranges indicating how each RABN in the database is used. Because a file save tape contains only the FCBs of the saved files, gaps exist in the physical layout table and are reported as "unknown" ranges rather than errors.

135 Report Description

- General Database Information 812
- File Information 823
- Checkpoint Information 835

The ADAREP database status report contains general database information followed by information about the status, allocation, and definition of each file in the database. Although the report is designed for printing from the SYSLST (BS2000), DDDRUCK (z/OS or z/VM), or SYS009 (VSE) data set, the following figures show examples of the report output displayed at a terminal. The examples display sections in the order they appear in the report; a description of each part is provided with them.

Note: Individual Adabas add-on products may supplement the information displayed on the ADAREP report. For example, if the database supports replication (via the Event Replicator for Adabas), additional statistics appear in various areas of this report. For complete information on the impact of the add-on products to this report, refer to the documentation for the Adabas add-on product.

General Database Information

The first section contains general information about the database and its physical layout:

```
********************
* *
* Data Base Report * yyyy-mm-dd hh:mm:ss
* *
********************
Data Base Name = EXAMPLE-DB
Data Base Number = 238
Date Loaded = yyyy-mm-dd
Time Loaded = hh:mm:ss
Checkpoint File = 5
Security File = 4
Maximum number of files = 255
Number of files loaded = 130
Current Log Tape Number = 184
RABNSIZE = 3
Recovery Aid = No
Universal Encoding Sup. = No
Replication = Yes
```

Additionally, if universal encoding support (UES) is enabled (UES=YES), the following information is displayed:

```
Universal Encoding Sup. = Yes
 ALPHA FILE ENCODING     =       37
 WIDE FILE ENCODING      =     4095
 ALPHA ASCII ENCODING    =      437
 WIDE USER ENCODING      =     4095
 Replication             = No
```

If UES=NO, this information is suppressed.

Field	Explanation
ALPHA ASCII ENCODING	Current user encoding set for alphanumeric (A) format fields in the database. Must be ASCII-compatible.
ALPHA FILE ENCODING	Current file encoding set for alphanumeric (A) format fields in the database. Must be EBCDIC-compatible.
CURRENT LOG TAPE NUMBER	Number of the most recent data protection log for the database.
DATABASE NAME	Name assigned to the database. See the ADADEF utility, DBNAME parameter.
DATABASE NUMBER	Number (ID) assigned to the database. See the ADADEF utility, DBIDENT parameter.
DATE LOADED	Date the database was initially defined.
MAXIMUM NUMBER OF FILES	Maximum number of files permitted for the database. See the ADADEF utility, MAXFILES parameter.
NUMBER OF FILES LOADED	Number of files currently in the database.
RABNSIZE	Length of the blocks in the database. RABNSIZE=3 indicates 24-bit blocks; RABNSIZE=4 indicates 31-bit blocks.
RECOVERY AID	Whether the Adabas Recovery Aid (ADARAI) is active for the database.
Replication	Whether or not replication (using Event Replicator for Adabas) is active for the database.
Replicator File	Identifies the file number of the Replicator system file if one is loaded on an Event Replicator Server.
Reptor SLOG File	Identifies the file number of the SLOG system file if one is loaded on an Event Replicator Server.
SYSTEM FILES	File numbers of Adabas system files.
TIME LOADED	Time of day when the database was initially defined.
TRIGGER FILE	If the database contains a trigger file, this entry displays the file number. If no trigger file exists in the database, this line does not print.
UNIVERSAL ENCODING SUPPORT	Whether universal encoding support (UES) is active for the database.
WIDE FILE ENCODING	Current file encoding set for wide-character (W) format fields in the database.
WIDE USER ENCODING	Current user encoding set for wide-character (W) format fields in the database.

Space Allocated to Database Components

The "physical layout" table lists the space allocations for the major components of the database (Associator, Data Storage, and Work).

The "unused storage" table lists the unused space in the Associator and Data Storage areas. This space is not assigned to any file in the database.

```
P H Y S I C A L  L A Y O U T

       DD- I DEV   I NMBR OF I NMBR OF EXTENTS IN BLK. I BLOCK I NMBR OF I
     NAMES I TYPE  I CYLS    I BLOCKS  FROM TO         I LNGTH I M-BYTE  I
 ----------I-------I---------I--------------------------I-------I---------I
           I       I         I                          I       I         I
    ASSOR1 I 3380  I 100     I 28481    1      28481   I 2004  I 54      I
           I       I         I                          I       I         I
    DATAR1 I 3380  I 200     I 26991    1      26991   I 4820  I 124     I
           I       I         I                          I       I         I
    WORKR1 I 3380  I  40     I  5391    1      5391    I 4820  I 24      I
           I       I         I                          I       I         I
 ------------------------------------------------------------------------

  U N U S E D S T O R A G E

       DD- I DEV   I NMBR OF I NMBR OF EXTENTS IN BLK. I BLOCK  I NMBR OF I
     NAMES I TYPE  I CYLS    I BLOCKS  FROM TO         I LNGTH  I M-BYTE  I
 ----------I-------I---------I--------------------------I--------I---------I
           I       I         I                          I        I         I
    ASSOR1 I 3380  I  98     I 28134   328       28461 I 2004   I 54      I
           I       I         I                          I        I         I
    DATAR1 I 3380  I 198     I 26811   131       26941 I 4820   I 124     I
```

The columns in these tables provide the following information:

Column	Explanation
DDNAMES	The job/task control name (without the DD prefix) that defines the Associator, Data Storage, or Work component of the database.
DEV TYPE	The physical device containing the Associator, Data Storage, or Work component.
NMBR OF CYLS	The DASD cylinders allocated to the Associator, Data Storage, and Work components. If less than one full cylinder has been allocated, "0" is shown in this column.
NMBR OF BLOCKS	The total number of blocks assigned to the Associator, Data Storage, or Work component. Please note that for Data Storage, Associator, and Work, the first track is not used. ADAREP only shows the number of blocks that are used by Adabas, and not the blocks that are allocated and formatted for use.

Column	Explanation
EXTENTS IN BLK	The extents, listed by block range.
BLOCK LNGTH	The block size. The block size depends on the component and the device type.
NUMBER OF M-BYTES	The component storage size, in megabytes.

Contents of the Database: General File Status

The next section contains information on the status of each file in the database. Here is an example:

```
************************************************
* Contents of Database    99 (EXAMPLE-DB)       *         yyyy-mm-dd  hh:mm:ss
************************************************

  File    Name            Loaded      TOP-ISN    MAX-ISN          EXTENTS
                                                                 N    U    A    D

     1 EMPLOYEES        2001-12-28       1107       1695     1    1    1    1
     2 MISCELLANEOUS    2001-12-28       1779       2543     1    1    1    1
     3 VEHICLES         2001-12-28        773       1695     1    1    1    1
```

Here is another example showing a database that uses a LOB file:

```
************************************************
* Contents of Database    99 (EXAMPLE-DB)       *         yyyy-mm-dd  hh:mm:ss
************************************************

File    Name            Loaded      TOP-ISN    MAX-ISN          EXTENTS
                                                               N    U    A    D

   1 SQLNC            2006-02-17          0       1377     1    1    1  301 *
   2 BASEFILE         2006-09-01         45       1377     1    1    1    1
   3 LOBFILE          2006-09-01          3       1377     1    1    1    1
   6 abcdefghijklmnop 2006-07-17          0       4133     1    1    1  300

Warning: * indicates less than 10 file extents remain for use
```

When LAYOUT=1 is specified for the ADAREP utility run, this section includes padding factor information merged with all the information in the File Space Allocation section of the report.

The columns in this table provide the following information:

Column	Explanation
FILE	Adabas file number.
NAME	File name (see the ADALOD utility, NAME parameter). If the file cannot build at least ten further extents, it is marked with an asterisk (*) to the right of the name.
LOADED	Date the file was loaded.
TOP-ISN	Highest ISN currently used in the file.
MAX-ISN	Highest ISN that can be assigned to a record in the file (see the ADALOD utility, MAXISN parameter).
EXTENTS	Number of logical extents currently assigned to the normal index (N), upper index (U), address converter (A), and Data Storage (D). The maximum number of logical file extents that you can now define is derived from the block size of the first Associator data set (DDASSOR1). The extent information is stored in a variable section of the FCB. New extents can be added now until the used FCB size reaches the block size of the Associator data set. If the extent limit has been reached, reorder the file (using ADAORD REORFILE or the ADAULD, ADADBS DELETE, ADALOD LOAD utility sequence) before the last extent fills, or Adabas will lock the file.
PADD	The block padding factor defined for the Associator (A%) and Data Storage (D%) (read about the ASSOPFAC and DATAPFAC parameters of the ADALOD LOAD utility for more information). **Note:** This column appears only when LAYOUT=1 is specified.

File Options

The next section lists the file options that are active for each file in the database. Here is an example:

```
****************
* File Options *
****************


                       ADAM File
                       .  Coupled File
                       .  .  ISNREUSE
                       .  .  .  DSREUSE
                       .  .  .  .  Ciphered File
                       .  .  .  .  .  Expanded File
                       .  .  .  .  .  .  USERISN
                       .  .  .  .  .  .  .  NOACEXTENSION
                       .  .  .  .  .  .  .  .  MIXDSDEV
                       .  .  .  .  .  .  .  .  .  PGMREFRESH
                       .  .  .  .  .  .  .  .  .  .  Multi-Client File
                       .  .  .  .  .  .  .  .  .  .  .  Index Compressed
                       .  .  .  .  .  .  .  .  .  .  .  .  2-Byte MU/PE Index
                       .  .  .  .  .  .  .  .  .  .  .  .  .  Spanned Record
                       .  .  .  .  .  .  .  .  .  .  .  .  .  .  Replicated
                       .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  Priv Use
 File  Name            .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
-------------------------------------------------------------------------------
    1  EMPLOYEES        .  .  I  D  .  .  .  .  .  .  .  .  .  .  .  .
    2  64BIT-1          .  .  .  D  .  .  .  .  .  .  .  .  .  .  .  .
    3  MISC             .  .  I  D  .  .  .  .  .  .  .  .  .  .  .  .
   19  CHECKPOINT-FILE  .  .  .  D  .  .  .  .  .  .  .  .  .  .  .  .
```

Here is another example showing a database that uses a LOB file:

```
****************
* File Options *
****************


                        ADAM File
                        . Coupled File
                        . . ISNREUSE
                        . . . DSREUSE
                        . . . . Ciphered File
                        . . . . . Expanded File
                        . . . . . . USERISN
                        . . . . . . . NOACEXTENSION
                        . . . . . . . . MIXDSDEV
                        . . . . . . . . . PGMREFRESH
                        . . . . . . . . . . Multi-Client File
                        . . . . . . . . . . . Index Compressed
                        . . . . . . . . . . . . 2-Byte MU/PE Index
                        . . . . . . . . . . . . . Spanned Record
                        . . . . . . . . . . . . . . Replicated
                        . . . . . . . . . . . . . . . Contains LOB Fields
                        . . . . . . . . . . . . . . . . Privileged Use
                        . . . . . . . . . . . . . . . . . LOB File
 File  Name             . . . . . . . . . . . . . . . . . .
------------------------------------------------------------------------------
    2  BASEFILE         . . I D . . . . . . C . . . L . .
    3  LOBFILE          . . I D . . . . . . C . . . . . L
   11  UES-EMPLOYEES    . . . D . . . . . . . . . . . . .
   13  UES-TEST         . . . D . . . . . . . . . . . . .
   14  DBCS3035         . . . D . . . . . . . . . . . . .
   15  COLLATION1       . . I D . . . . . P . . . . . . .
   18  SECURITY         . . . D . . . . . . . . . . . . .
   19  CHECKPOINT       . . . D . . . . . . . . . . . . .
   23  EMPL23-EXT       . . . D . . U . . P . . . . . . .
   24  SAGABS_MYFILEXX  . . I D . . . . M . . C . . . . . .
   50  EMPL5O-EXT       . . . D . . U . . . . . . . . . .
   88  BASEF-LBLA       . . . D . . . . . . . . . . L . .
   89  LOBF-LBLA        . . . D . . . . . . C . . . . . L
  101  file101          . . I D . . . . M . . C . . . . . .
  266  UES-EMPLOYEES    . . . D . . . . . . . . . . . . .
  267  UES-EMPLOYEES    . . . D . . . . . . . . . . . . .
```

Options that are active for a file are indicated by the following codes in the row containing the file name:

Code	Explanation
A	ADAM file. The file was loaded with the ADAM option.
C	Coupling, ciphering, or index compression. The file is coupled to one or more files, and/or the file data is ciphered, and/or the file index is compressed.
D	Space reuse. Space which has been released within a block as a result of a record deletion may be used for a new record.
I	ISN reuse. ISNs of deleted records may be reassigned to new records.
L	If this appears in the "Contains LOB Fields" column, the file contains LB fields (it is a base file). If this appears in the "LOB File" column, the file is a LOB file, not a base file. **Note:** These two columns are mutually exclusive; an "L" will only appear in one of them, if it appears at all.
M	MIXDSDEV active (multiple Data Storage device types) and/or a multiclient file.
N	File is defined with the NOACEXTENSION option.
P	PGMREFRESH is active. "P" is also used in the Priv Use column to indicate that the file is locked by the nucleus for privileged use. The privileged use information is also available in the *File Information* section of the ADAREP report.
R	Replication (Event Replicator for Adabas processing) is active for the file.
S	Spanned record support is activated for the file.
T	Two-byte MU/PE indexes (when MU/PE occurrences exceed 191) are active for the file.
U	File was loaded with the USERISN option.
X	File is a component of an expanded file.

File Space Allocations

The next section shows the space allocated for each file in the database. Here is an example:

```
      **************************
      * FILE SPACE ALLOCATIONS *
      **************************

      FILE    NAME     ALLOC.: NI    UI   AC   DATA/CYL
                       UNUSED:

      1    EMPLOYEES         100     30   03     80/0
      1                       24     17          31/0
      2    VEHICLES           10     20   03     30/0
      2                       03     02          12/0
     10    CHECKPOINT         10     01   01     20/0
     10                       05      0          11/0

↩
```

Here is another example showing a database that uses a LOB file:

```
*************************
* File Space Allocations *
*************************

  File   Name          Alloc.:   NI          UI          AC        Data/Cyl
                       Unused:

     2 BASEFILE                 10           5           1          10/0
     2                          10           5                       9/0
     3 LOBFILE                  10           5           1         100/1
     3                           9           3                      99/1
    11 UES-EMPLOYEES           47          20           1          75/1
    11                          0           4                      53/0
    13 UES-TEST                30          20           1          10/0
    13                         25          14                       9/0
    14 DBCS3035                60          22           1          30/0
    14                         22           9                       8/0
    15 COLLATION1               6           3           1           1/0
    15                          4           0                       0/0
    18 SECURITY                 1           1           1           1/0
    18                          1           0                       1/0
    19 CHECKPOINT              30           3           4          90/1
    19                         30           2                      53/0
    23 EMPL23-EXT              60          24           2          30/0
    23                         52          19                      27/0
    24 SAGABS_MYFILEXX         24           4           1           5/0
    24                         24           0                       5/0
    50 EMPL50-EXT              80          49           2         100/1
    50                         25          28                      75/1
    88 BASEF-LBLA              10           5           1          10/0
    88                         10           4                      10/0
    89 LOBF-LBLA               10           5           8         411/5
    89                         10           4                     411/5
   101 file101                  7          14           2          72/0
   101                          6          12                      71/0
   266 UES-EMPLOYEES           30          20           1          75/1
   266                         30          18                      75/1
   267 UES-EMPLOYEES           63          20           1          75/1
   267                         16           4                      53/0
```

When LAYOUT=1 is specified, this section is merged into the Contents of Database section of the report.

Each file listed has two rows in the file space allocations table. The first row shows the number of blocks and cylinders *allocated*. The second row shows the number of blocks and cylinders currently *unused*.

The first two columns give the number and logical name of the file. The remaining columns provide the following information:

Column	The number of . . .
NI	blocks for the normal index.
UI	blocks for the upper index.
AC	blocks for the address converter.
DATA/CYL	blocks and cylinders for Data Storage.

LOB File

If a database includes a LOB file, an additional section describing the LOB file is included in the report. Here is an example:

```
*************
*           *
* LOB Files *                                          yyyy-mm-dd  hh:mm:ss
*           *
*************



File with  I Associated I
LOB fields I LOB file   I
--------------------------
         2 I          3 I
-----------I------------I
        88 I         89 I
-----------I------------I
```

The columns provide the following information:

Column	Lists:
File with LOB fields	The file numbers of files containing LB fields.
Associated LOB file	The number of the LOB file in which the actual LB field values are stored.

Physical Layout of the Database

The next section lists all space allocations for the database in RABN sequence. RABNs allocated to the Associator are listed first, followed by RABNs allocated to Data Storage.

```
**********************************
*                                *
* Physical Layout of the Database *                        yyyy-mm-dd  hh:mm:ss
*                                *
**********************************



      From         To     Number  Dev   Table File VOLSER
       Blk        Blk    of Blks Type   Type       Number

       131    -   162         32 3390    PPT     0 ADA001
       163    -   163          1 3390   DSST     0 ADA001
       164    -   164          1 3390    AC     19 ADA001
       165    -   174         10 3390    UI     19 ADA001
       175    -   224         50 3390    NI     19 ADA001
       225    -   242         18 3390    AC      2 ADA001
       243    -   243          1 3390    AC2     2 ADA001
       244    -   254         11 3390  UNUSED    0 ADA001
       255    -   259          5 3390    AC      1 ADA001
```

Note: Normally, a gap in the physical layout table is accompanied by an error message pointing to the gap. However, this is not the case for the physical layout of a file save. Since the file save contains only the FCBs of the saved files, there will be gaps in the physical layout table and these are reported as 'unknown' ranges.

The columns in this table provide the following information:

Column	Explanation
FROM BLK	The RABN of the first block in the logical extent.
TO BLK	The RABN of the last block in the logical extent.
NUMBER OF BLKS	The number of blocks contained within the extent.
DEV TYPE	The physical device type.
TABLE TYPE	The element for which the allocation was made: AC — address converter NI — normal index UI — upper index DS — Data Storage DSF — Delta Save logging area DSST — Data Storage Space Table UNUSED — available space
FILE	The file for which the allocation was made. Zero indicates that the extent is not related to a particular file.

Column	Explanation
VOLSER NUMBER	The serial number of the volume on which the extent is contained. This is shown for Data Storage only if the Data Storage data sets are present in the JCL.

File Information

General Characteristics

Detailed information on each file in the database is provided after the database information. This information can be limited to certain files or omitted altogether. The first part of this section displays information about the file's characteristics. Here is an example of a file containing spanned records:

```
*********************************
*                               *
* File     2 (64BIT-1         ) *                          yyyy-mm-dd  hh:mm:ss
*                               *
*********************************



TOP-ISN            =            25     Highest Index Level =  3
MAX-ISN Expected   =        15,263     Padding Factor ASSO = 10%
Records Loaded     =            25     Padding Factor DATA =  0%
MIN-ISN            =             1     Length of Client NR =  0
TOP AC2 ISN        =           100
MAX AC2 ISN Exp.   =           848
MIN AC2 ISN        =             1
Number of Updates  =             0     ISNSIZE             =  3

MAX COMP REC LEN   =           N/A     Date Loaded         = 2006-03-25
BLK/ADD DS  EXT    =             0     Time Loaded         = 16:41:21
BLK/ADD UI  EXT    =             0
BLK/ADD NI  EXT    =             0
 ↩
```

Here is an example of the general characteristics of the *base file* of a base file-LOB file pair:

```
*********************************
*                               *
* File     2 (BASEFILE        ) *                          yyyy-mm-dd  hh:mm:ss
*                               *
*********************************



TOP-ISN            =            45     Highest Index Level =  0
MAX-ISN Expected   =         1,377     Padding Factor ASSO = 10%
Records Loaded     =            44     Padding Factor DATA = 10%
MIN-ISN            =             1     Length of Client NR =  0
Number of Updates  =        55,937     ISNSIZE             =  3

MAX COMP REC LEN   =        10,792     Date Loaded         = 2006-09-01
BLK/ADD DS  EXT    =             0     Time Loaded         = 14:16:57
BLK/ADD UI  EXT    =             0     Date of last update = 2007-01-23
BLK/ADD NI  EXT    =             0     Time of last update = 07:00:52

FILE ALPHA CODE    =            37
FILE WIDE CODE     =         4,095
USER WIDE CODE     =         4,095
```

Here is an example of the general characteristics of the *LOB file* of a base file-LOB file pair:

```
*********************************
*                               *
* File     3 (LOBFILE         ) *                          yyyy-mm-dd  hh:mm:ss
*                               *
*********************************



TOP-ISN            =             3     Highest Index Level =  3
MAX-ISN Expected   =         1,377     Padding Factor ASSO = 10%
Records Loaded     =             0     Padding Factor DATA = 10%
MIN-ISN            =             1     Length of Client NR =  0
Number of Updates  =             0     ISNSIZE             =  3

MAX COMP REC LEN   =        10,792     Date Loaded         = 2006-09-01
BLK/ADD DS  EXT    =             0     Time Loaded         = 14:16:58
BLK/ADD UI  EXT    =             0
BLK/ADD NI  EXT    =             0

FILE ALPHA CODE    =    DB DEFAULT
FILE WIDE CODE     =    DB DEFAULT
USER WIDE CODE     =    DB DEFAULT
```

The following information can be provided on this report (although all of these fields may not appear on the samples above):

Field	Explanation
BLK/ADD DS EXT	Maximum number of blocks which may be allocated for each Data Storage secondary extent. See the ADALOD utility, MAXDS parameter.
BLK/ADD NI EXT	Maximum number of blocks which may be allocated for each secondary normal index extent. See the ADALOD utility, MAXNI parameter.
BLK/ADD UI EXT	Maximum number of blocks which may be allocated for each secondary upper index extent. See the ADALOD utility, MAXUI parameter.
Collect before images of updates	Indicates whether or not before images of data storage are collected for replication (if replication is activated) during the update of a record on a file. This information is shown whether or not a file is replicated with a primary key defined.
DATE LOADED	Date the file was loaded.
DATE OF LAST UPDATE	Date the file was last changed.
FILE ALPHA CODE	Current file encoding set for alphanumeric fields in the file. This information is not displayed if UES=NO.
FILE WIDE CODE	Current file encoding set for wide-character fields in the file. This information is not displayed if UES=NO.
HIGHEST INDEX LEVEL	Highest index level currently active for the file.
ISNSIZE	Whether the file contains 3-byte or 4-byte ISNs.
LENGTH OF CLIENT NR	Length of the owner ID for a multiclient file.
MAX AC2 ISN EXP	The highest secondary ISN expected in the file. This statistic is given only if spanned records are activated for the file.
MAX COMP REC LEN	Maximum compressed record length permitted for the file. See the ADALOD utility, MAXRECL parameter. If spanned record support is enabled for the file, the value for this field is shown as "N/A".
MAX-ISN EXPECTED	Highest ISN planned for the file. See the ADALOD utility, MAXISN parameter.
MIN AC2 ISN	The lowest secondary ISN in the file. This statistic is given only if spanned records are activated for the file.
MIN-ISN	Lowest ISN that can be assigned to a record in the file. See the ADALOD utility, MINISN parameter.
Number of Updates	Number of updates that have been applied to the file after it was loaded.
PADDING FACTOR ASSO	Associator padding factor. For more information, read about the ADALOD LOAD **ASSOPFAC** parameter or the ADAORD **REORASSO** and **REORFASSO** functions, elsewhere in this guide.
PADDING FACTOR DATA	Data Storage padding factor. For more information, read about the ADALOD LOAD **DATAPFAC** parameter or the ADAORD **REORDATA** and **REORFDATA** functions, elsewhere in this guide. **Note:** If spanned records are used, the padding factor is ignored, in an attempt to fully use the block. So it is frequently listed as zero in this report. The padding factor is only used in the last, short, segment of a spanned record.

Field	Explanation
Primary key of replicated records	Identifies the primary key for replication, if replication is activated. This information is shown only if a file is replicated with a primary key defined.
Records Loaded	Number of records currently contained in the file.
Replicator target ID	Identifies the database ID of the Event Replicator Server used for replication, if replication is activated. This information is shown whether or not a file is replicated with a primary key defined.
TIME LOADED	Time the file was loaded.
TIME OF LAST UPDATE	Time the file was last changed.
TOP AC2 ISN	The highest secondary ISN in use in the file. This statistic is given only if spanned records are activated for the file.
TOP-ISN	Highest ISN currently used in the file.
USER WIDE CODE	Current user encoding set for wide-character fields in the file. This information is not displayed if UES=NO.

Options

File option settings for the file are displayed next. Here is an example showing that spanned records are used:

```
ADAM File          No
Ciphered File      No
ISN Reusage        No
Space Reusage      Yes
Coupled Files      None
Expanded File      No
USERISN            No
NOACEXTENSION      No
MIXDSDEV           No
PGMREFRESH         No
Multi Client File  No
Privileged usage   No
Online INVERT      None
Index Compressed   No
Spanned Rec Supp   Yes
Two Byte MU/PE     No


ADABAS version needed for this file: V71 or later
```

Here is an example of the *base file* of a base file-LOB file pair, showing that the file contains LB fields:

```
ADAM File          No
Ciphered File      No
ISN Reusage        Yes
Space Reusage      Yes
Coupled Files      None
Expanded File      No
USERISN            No
NOACEXTENSION      No
MIXDSDEV           No
PGMREFRESH         No
Multi Client File  No
Privileged usage   No
Online INVERT      None
Index Compressed   Yes
Spanned Rec Supp   No
Two Byte MU/PE     No
LOB file           No
Contain LOB fields Yes
```

Here is an example of the *LOB file* of a base file-LOB file pair, showing that the file is itself a LOB file:

```
ADAM File          No
Ciphered File      No
ISN Reusage        Yes
Space Reusage      Yes
Coupled Files      None
Expanded File      No
USERISN            No
NOACEXTENSION      No
MIXDSDEV           No
PGMREFRESH         No
Multi Client File  No
Privileged usage   No
Online INVERT      None
Index Compressed   Yes
Spanned Rec Supp   No
Two Byte MU/PE     No
LOB file           Yes
Contain LOB fields No
```

Field	Indicates . . .
ADAM File	whether the file was loaded with the ADAM option.
Ciphered File	whether the file was loaded with the cipher option.
ISN Reusage	whether the file ISNs can be reused.
Space Reusage	whether the file Data Storage space can be reused.
Coupled Files	the file(s) to which this file is physically coupled.
Expanded File	whether the file is part of an expanded file; if so, the number of the expanded file is displayed.
USERISN	whether the file was loaded with the USERISN option.
NOACEXTENSION	whether the file permits increasing the MAXISN setting.
MIXDSDEV	whether the file Data Storage extents can be on different device types.
PGMREFRESH	whether the file can be refreshed using the E1 command.
Multiclient File	whether the file can contain records belonging to multiple owners/owner IDs.
Privileged usage	whether the file was locked by the nucleus for privileged usage; if so, only Adabas utilities are allowed to access the file.
Online INVERT	the descriptor(s) being inverted online.
Index Compressed	whether the file index is compressed.
Spanned Rec Supp	whether spanned record support is activated for the file.
Two Byte MU/PE	whether two-byte MU/PE indexes (when MU/PE occurrences exceed 191) are active for the file.
LOB file	whether the file is a LOB file.
Contain LOB fields	whether the file contains one or more LB fields (it is a base file).

Delta Save Change Flags

If the Delta Save Facility is installed on the database and delta save logging is enabled, ADAREP shows the delta save change flags for each file:

```
     DELTA SAVE CHANGE FLAGS:
      SAVE ENTIRE INDEX     = [YES | NO]
      SAVE ENTIRE ADDR CONV = [YES | NO]
      SAVE ENTIRE DATA STOR = [YES | NO]
      TOTAL CHANGES BY UTILITIES = nnn BLOCKS

↩
```

Each flag indicates whether all of the index, address converter, or Data Storage, respectively, of the file have been changed by a utility and will be saved entirely in the next delta save operation.

The "TOTAL CHANGE BY UTILITIES" include the blocks within extents that will be saved entirely as well as the blocks changed by ADALOD UPDATE executions.

Space Allocation

The next section lists the space allocations for the file. Here is an example showing space allocations when spanned records are used:

```
List I Dev  Block I   Space Alloc.   I     From        To  I   Unused Space   I
Type I Type Lngth I  Blocks     Cyl  I     RABN       RABN I  Blocks     Cyl  I
     I            I                  I                     I                  I
-----I------------I------------------I---------------------I------------------I
     I            I                  I                     I                  I
 AC  I 3390  2544 I        18       0I       225        242I                  I
 AC2 I 3390  2544 I         1       0I       243        243I                  I
 NI  I 3390  2544 I       500       1I       524       1023I       499       1I
 UI  I 3390  2544 I       100       0I      1024       1123I        98       0I
DSST I 3390  2544 I         1       0I       163        163I                  I
     I            I                  I                     I                  I
 DS  I 3390  5064 I      1500      10I       473       1972I      1397       9I
     I            I                  I                     I                  I
-------------------------------------------------------------------------------- ↩
```

Here is an example of the space allocation of a *base file* in a base file-LOB file pair:

```
 List I Dev  Block I   Space Alloc.   I     From        To  I   Unused Space   I
 Type I Type Lngth I  Blocks     Cyl  I     RABN       RABN I  Blocks     Cyl  I
      I            I                  I                     I                  I
 -----I------------I------------------I---------------------I------------------I
      I            I                  I                     I                  I
  AC  I 8391  4136 I         1       0I      1717       1717I                  I
  NI  I 8391  4136 I        10       0I      1988       1997I        10       0I
  UI  I 8391  4136 I         5       0I      1718       1722I         5       0I
  FDT I 8391  4136 I         4       0I       335        338I                  I
 DSST I 8391  4136 I         1       0I      1563       1563I                  I
      I            I                  I                     I                  I
  DS  I 8391 10796 I        10       0I       176        185I         9       0I
      I            I                  I                     I                  I
 --------------------------------------------------------------------------------
```

Here is an example of the space allocation of a *LOB file* in a base file-LOB file pair:

```
List I Dev  Block I   Space Alloc.   I     From        To  I   Unused Space   I
Type I Type Lngth I  Blocks     Cyl  I     RABN       RABN I  Blocks     Cyl  I
     I            I                  I                    I                  I
-----I------------I------------------I----------------------I------------------I
     I            I                  I                    I                  I
 AC  I 8391  4136 I         1       0I      1998       1998I                  I
 NI  I 8391  4136 I        10       0I      1732       1741I         9       0I
 UI  I 8391  4136 I         5       0I      1999       2003I         3       0I
 FDT I 8391  4136 I         4       0I       339        342I                  I
DSST I 8391  4136 I         1       0I      1563       1563I                  I
     I            I                  I                    I                  I
 DS  I 8391 10796 I       100       1I       403        502I        99       1I
     I            I                  I                    I                  I
-------------------------------------------------------------------------------
```

The space allocations table provides the following information:

Column	Explanation	
LIST TYPE	The database component:	
	AC	address converter
	AC2	secondary address converter extents (for spanned records)
	NI	normal index
	UI	upper index
	DS	Data Storage
	DSF	File-specific delta save logging area
	DSST	Data Storage Space Table
	UNUSED	Available space
DEV TYPE	Physical device containing the component.	
BLOCK LNGTH	Block length depends on the component and device type.	
SPACE ALLOC.	Total number of blocks and cylinders allocated to the component; "0" indicates less than one full cylinder.	
FROM RABN	RABN of the first block in the logical extent.	
TO RABN	RABN of the last block in the logical extent.	
UNUSED SPACE	Number of allocated blocks and cylinders but currently unused; "0" indicates less than one full cylinder.	

Field Definition Table

The Field Definition Table (FDT) is displayed next. This information can be omitted. Here is a general example of the FDT section of the report:

```
       FIELD DESCRIPTION TABLE

             I      I        I        I               I
       LEVEL I NAME I LENGTH I FORMAT I    OPTIONS    I           PARENT OF
             I      I        I        I               I
      ------I------I--------I--------I---------------I-----------------------------I
             I      I        I        I               I                             I
         1   I AA   I    8   I   A    I     DE,UQ     I                             I
         1   I AB   I        I        I               I                             I
         2   I AC   I   20   I   A    I     NU        I                             I
         2   I AE   I   20   I   A    I     DE        I              SUPERDE,PHONDE I
         2   I AD   I   20   I   A    I     NU        I                            ↩
I
         1   I AF   I    1   I   A    I     FI        I                             I
         1   I AG   I    1   I   A    I     FI        I                            I
         1   I AH   I    6   I   U    I     DE        I                             I
         1   I A2   I        I        I               I                             I
         1   I AO   I    6   I   A    I     DE        I              SUBDE,SUPERDE  I
         1   I AQ   I        I        I     PE        I                             I
         2   I AR   I    3   I   A    I     NU        I              SUPERDE        I
         2   I AS   I    5   I   P    I     NU        I              SUPERDE        I
         1   I A3   I        I        I               I                             I
         2   I AU   I    2   I   U    I               I              SUPERDE        I
         2   I AV   I    2   I   U    I     NU        I              SUPERDE        I

↩
```

Here is an example of part of the FDT associated with the *base file* of a base file-LOB file pair, showing the LB fields in the base file.

```
Field Description Table

      I      I        I        I                       I                        I
Level I Name I Length I Format I  Options              I  Parent of            I
      I      I        I        I                       I                        I
------I------I--------I--------I-----------------------I------------------------I
      I      I        I        I                       I                        I
  1   I  AA  I    6   I    A   I NU                    I                        I
  1   I  AP  I    2   I    P   I NU                    I                        I
  1   I  A1  I    0   I    A   I NU                    I                        I
  1   I  A2  I    0   I    A   I NU,NV                 I                        I
  1   I  A3  I    0   I    A   I MU,NU                 I                        I
  1   I  A4  I    0   I    A   I MU,NU,NV              I                        I
  1   I  A5  I    0   I    A   I NC                    I                        I
  1   I  A6  I    0   I    A   I NC,NV                 I                        I
  1   I  B1  I    0   I    A   I LA,NU                 I                        I
  1   I  B2  I    0   I    A   I LA,NB,NU              I                        I
  1   I  B3  I    0   I    A   I LA,NU,NV              I                        I
  1   I  B4  I    0   I    A   I LA,NB,NU,NV           I                        I
  1   I  B5  I    0   I    A   I LA,MU,NU              I                        I
  1   I  B6  I    0   I    A   I LA,NB,MU,NU           I                        I
  1   I  B7  I    0   I    A   I LA,MU,NU,NV           I                        I
  1   I  B8  I    0   I    A   I LA,NB,MU,NU,NV        I                        I
  1   I  C1  I    0   I    A   I LB,NU                 I                        I
  1   I  C2  I    0   I    A   I LB,NB,NU              I                        I
  1   I  C3  I    0   I    A   I LB,NU,NV              I                        I
  1   I  C4  I    0   I    A   I LB,NB,NU,NV          I                        I
  1   I  C5  I    0   I    A   I LB,MU,NU              I                        I
  1   I  C6  I    0   I    A   I LB,NB,MU,NU          I                        I
  1   I  C7  I    0   I    A   I LB,MU,NU,NV          I                        I
  1   I  C8  I    0   I    A   I LB,NB,MU,NU,NV       I                        I
  1   I  D1  I    0   I    W   I NU                    I                        I
  1   I  D2  I    0   I    W   I NU,NV                 I                        I
  1   I  D3  I    0   I    W   I MU,NU                 I                        I
  1   I  D4  I    0   I    W   I MU,NU,NV              I                        I
  1   I  D5  I    0   I    W   I NC                    I                        I
  1   I  D6  I    0   I    W   I NC,NV                 I                        I
  1   I ↩
```

FDT sections are not printed for *LOB files*.

Field	Explanation
LEVEL	Field level.
NAME	Field name.
LENGTH	Field length, in bytes.
FORMAT	Field's data type: A – alphanumeric B – binary

Field	Explanation	
	F	fixed point
	P	packed decimal
	G	floating point
	U	unpacked decimal
	W	wide-character
OPTIONS	DE	Descriptor
	FI	Fixed storage
	LA	Long alphanumeric
	LB	Large object field
	MU	Multiple-value field
	NB	No blank compression
	NC	Null/not counted
	NN	Null not allowed
	NU	Null value suppression
	NV	Not converted (alpha and wide-character fields)
	PE	A periodic group. The fields composing the periodic group are those which follow and have a higher level number.
	UQ	Unique descriptor
	XI	Index (occurrence) number excluded from UQ in PE
PARENT OF	Shows whether this field is a parent field for a collation descriptor, sub/superfield, sub/superdescriptor, hyperdescriptor, or phonetic descriptor.	

Special Descriptors

The next section displays information about any special descriptors (collation descriptors, sub-descriptors, subfields, superdescriptors, superfields, phonetic descriptors, and hyperdescriptors) in the file:

```
SPECIAL DESCRIPTOR TABLE
        I      I        I        I                        I                I
   TYPE I NAME I LENGTH I FORMAT I                OPTIONS I      STRUCTURE I
        I      I        I        I                        I                I
 -------I------I--------I--------I------------------------I----------------I
        I      I        I        I                        I                I
  SUPER I   H1 I      4 I      B I                  DE,NU I    AU ( 1 - 2) I
        I      I        I        I                        I    AV ( 1 - 2) I
    SUB I   S1 I      4 I      A I                     DE I    AO ( 1 - 4) I
  SUPER I   S2 I     26 I      A I                     DE I    AO ( 1 - 6) I
        I      I        I        I                        I   AE ( 1 - 20) I
  SUPER I   S3 I     12 I      A I               DE,NU,PE I    AR ( 1 - 3) I
        I      I        I        I                        I    AS ( 1 - 9) I
        I      I        I        I                        I                I
   PHON I   PH I        I        I                        I  PH = PHON(AE) I
        I      I        I        I                        I                I
    COL I   Y1 I     20 I      W I                     DE I       CDX 8,PA I
    COL I   Y2 I     12 I      A I               DE,NU,PE I       CDX 1,AR I
        I      I        I        I                        I                I
        I      I        I        I                        I                I
 ------------------------------------------------------------------------------
```

Along with the name, length, and format of each special descriptor, this table provides the following information:

Column	Explanation	
TYPE	SUB	Subfield/subdescriptor
	SUPER	Superfield/superdescriptor
	PHON	Phonetic descriptor
	HYPER	Hyperdescriptor
	COL	Collation descriptor
OPTIONS	DE	Descriptor field
	FI	Fixed point
	LA	Long alphanumeric
	MU	Multiple-value field
	NC	Null not counted (SQL null representation)
	NN	Null not allowed
	NU	Null value suppression
	NV	Not converted (alpha and wide-character fields)
	PE	Periodic group
	UQ	Unique descriptor
	XI	Index (occurrence) number excluded from UQ in PE

Column	Explanation
STRUCTURE	The component fields and field bytes of the sub-, super-, or hyperdescriptor. Phonetic descriptors show the equivalent alphanumeric elementary fields. Collation descriptors show the associated collation descriptor user exit and the name of the parent field.

Checkpoint Information

Checkpoint information is also provided if the CPLIST or CPEXLIST parameters are specified:

```
     ********************
     * CHECK-POINT-LIST *                                  yyyy-mm-dd hh:mm:ss
     ********************

     CP         CP         DATE         TIME       PLOG     BLOCK    JOBNAME
     NAME       TYPE                               NR       NR
                     USER TYPE                     VOLSER NR....

     SYNP        30        1995-06-03   14:07:38   47       1         DUAL GA0TB1
                     LOAD                          VOLSER = WRK001
     SYNC        01   ET   1995-06-03   14:08:16   48       2         DUAL GANUC70A
                     SESSION OPEN IGNDIB=N FORCE=N
     SYNP        1C   UTI  1995-06-03   14:08:36   48       3         DUAL GA0TB1
                     RESTRUCT

↩
```

The columns in this table provide the following information:

Column	Explanation
CP-NAME	The checkpoint identifier. In the case of a user non-synchronized checkpoint, this is the checkpoint identifier supplied by the user program. Checkpoint names starting with "SYN"" are reserved for the Adabas nucleus and utilities: ■ SYNC -- A synchronized checkpoint made during nucleus initialization, including the status of the ADARUN IGNDIB and FORCE parameters. ■ SYNF -- A checkpoint taken by a user program or utility that requires exclusive (EXF) control of one or more files. ■ SYNP -- A checkpoint from a utility that requires privileged control. Such a utility can perform updating without using the Adabas nucleus. ■ SYNS -- A checkpoint from Adabas Online System (SYSAOS) or ADADBS with three exceptions from the nucleus. The function identified by this checkpoint is implemented without user intervention during regeneration.

<table>
<tr><th>Column</th><th>Explanation</th></tr>
<tr><td></td><td>Exceptions include a second SYNS 5B recorded at the end of a nucleus session, SYNS 60 recorded at an interval specified by the ADARUN INTNAS parameter,and SYNS 61 recorded when more space is allocated for a file.
■ SYNV -- Indicates that a volume ID changed during sequential write to a data set is being closed.
■ SYNX -- A checkpoint from a utility requiring exclusive control (EXU) of one or more files.
■ SYN1 -- A checkpoint made at the beginning of online ADASAV execution (SAVE database function).
■ SYN2 -- A checkpoint made at the end of online ADASAV execution (SAVE database function).
■ SYN4 -- A checkpoint made at the beginning of online ADASAV execution (SAVE files operation).
■ SYN5 -- A checkpoint made at the end of online ADASAV execution (SAVE files operation).</td></tr>
<tr><td>CP TYPE</td><td>The checkpoint number. See the following table of checkpoints for the possible checkpoint numbers.</td></tr>
<tr><td>USER TYPE</td><td>The Adabas user type that set the checkpoint. The user types are:<table><tr><td>ET</td><td>ET user</td></tr><tr><td>EXF</td><td>exclusive-file-control user or utility (privileged user)</td></tr><tr><td>EXU</td><td>exclusive-file-update user or utility</td></tr><tr><td>UTI</td><td>utility-update-control utility (privileged user)</td></tr><tr><td>UTS</td><td>Online ADASAV SAVE file (privileged user)</td></tr></table></td></tr>
<tr><td>DATE TIME</td><td>The date and time the checkpoint was taken.</td></tr>
<tr><td>PLOG NR.</td><td>The number of the data protection log in use when the checkpoint was written to the checkpoint file.</td></tr>
<tr><td>BLOCK NR.</td><td>The block number of the data protection log in which the checkpoint was written.</td></tr>
<tr><td>VOLSER-NUMBER</td><td>The volume serial number of the sequential protection (DD/SIBA) log. The volume serial number is "DUAL" if dual logging is used and "MULTI" if multiple logging is used.</td></tr>
<tr><td>JOBNAME</td><td>The name of the job that created the checkpoint.</td></tr>
</table>

The following table describes the checkpoints written by the Adabas nucleus or utilities:

Type	Name	Originator	Description
01	SYNC	ADANUC	Written by nucleus at start of nucleus session.
01	SYNF	User/Util.	User/utility session OPEN with files used in EXF (exclusive use) mode.
01	SYNX	EXU user	EXU user open.
02	SYNV	ADANUC	VOLSER entry. Written at volume switch on DD/SIBA and at the end of the session if sequential logging is used.
03	SYNF	User/Util.	Close checkpoint for an EXF user.
03	SYNX	EXU	Close checkpoint for an EXU user.
05	SYNP	ADASAV	SAVE file(s)-start of operation
06	SYNP	ADASAV	SAVE database-start of operation
07	SYNP	ADASAV	RESTORE file(s)-end of operation
08	SYNP	ADASAV	RESTPLOG-end of operation
09	SYNV	ADASAV	SAVE file(s), VOLSER entry. Written at volume change on DD/SAVE and at SAVE-operation end.
0A	SYNV	ADASAV	SAVE database, VOLSER entry. Written at volume switch on DD/SAVE and at SAVE-operation end.
0B	SYNP	ADASAV	SAVE DELTA-end of operation
0C	SYNP	ADASAV	RESTORE DELTA-end of operation
0D	SYNP	ADASAV	MERGE-end of operation
0E	SYNV	ADASAV	SAVE DELTA, VOLSER entry
0F	SYNV	ADASAV	MERGE, VOLSER entry
10	SYNP	ADAINV	COUPLE files
11	SYNP	ADAINV	INVERT field(s)
15	SYNP	ADAORD	REORDER Associator database
16	SYNP	ADAORD	REORDER Data Storage database
17	SYNP	ADAORD	REORDER database
18	SYNP	ADAORD	REORDER Associator file
19	SYNP	ADAORD	REORDER Data Storage file
1A	SYNP	ADAORD	REORDER file
1B	SYNP	ADAORD	STORE
1C	SYNP	ADAORD	RESTRUCTURE
1D	SYNP	ADADEF	DEFINE NEWWORK
1E	SYNP	ADADEF	MODIFY default character encodings
22	SYNX	ADARES	REGENERATE file
23	SYNX	ADARES	BACKOUT file
24	SYNX	ADARES	REGENERATE all; CPEXLIST lists excluded files
25	SYNX	ADARES	BACKOUT all; CPEXLIST lists excluded files
26	SYNP	ADARES	REPAIR Data Storage

Type	Name	Originator	Description
27	SYNV	ADARES	COPY sequential protection log
28	SYNP	ADARES	PLCOPY function successfully completed
28	SYNV	ADARES	PLCOPY dual or multiple protection log
29	SYNV	ADARES	CLCOPY dual or multiple command log
2A	SYNP	ADARES	PLCOPY MERGE function successfully completed
2A	SYNV	ADARES	PLCOPY MERGE dual or multiple protection log
2B	SYNP	ADARES	CLOG MERGE function successfully completed
2B	SYNV	ADARES	CLOG MERGE dual or multiple command log
30	SYNP	ADALOD	LOAD file
31	SYNP	ADALOD	Mass update
35	SYNX	ADAULD	Unload file
3F	SYNP	ADAZAP	Successful VERIFY - REPLACE
40	SYNS	SYSAOS	Add extent
41	SYNS	SYSAOS	CHANGE default field length
42	SYNS	SYSAOS	DECREASE database size
44	SYNS	SYSAOS	Delete file
45	SYNS	SYSAOS	INCREASE database size
47	SYNS	SYSAOS	RECOVER space
48	SYNS	SYSAOS	Refresh file
49	SYNS	SYSAOS	Remove component file from expanded-file chain
4A	SYNS	SYSAOS	Release descriptor
4B	SYNS	SYSAOS	RENAME file
4C	SYNS	SYSAOS	RENUMBER file
4D	SYNS	SYSAOS	RESET DIB
4E	SYNS	SYSAOS	Reuse ISN
4F	SYNS	SYSAOS	Reuse Data Storage
50	SYNS	SYSAOS	UNCOUPLE files
51	SYNS	SYSAOS	ALLOCATE file extent
52	SYNS	SYSAOS	DEALLOCATE file extent
53	SYNS	SYSAOS	Delete checkpoint
54	SYNS	SYSAOS	Set user priority
55	SYNS	SYSAOS	Modify FCB
57	SYNS	SYSAOS	DEFINE file
58	SYNS	SYSAOS	Write FDT
59	SYNS	SYSAOS	DEFINE new field
5B	SYNS	ADADBS	Write refreshed statistics (some or all per user request)

Type	Name	Originator	Description
5B	SYNS	ADANUC	Write (all) statistics at end of nucleus session
5B	SYNS	ADARES	Write refreshed statistics (command, file, and thread usage; DRES and DSTAT)
5C	SYNS	SYSAOS	CHANGE default field format
5D	SYNS	SYSAOS	Change file encoding
5E	SYNS	ADADBS	ADADBS REPTOR function (refer to your Event Replicator for Adabas documentation)
60	SYNS	ADANUC	Nucleus statistic checkpoint
61	SYNS	ADANUC	Allocate file space
64	SYNS	ADASCR	Protect files
65	SYNS	ADASCR	Protect fields
66	SYNS	SYSAOS	Link component file into expanded-file chain
68	SYNS	SYSAOS	Set USERISN on/off
69	SYNS	SYSAOS	Set MIXDSDEV on/off
6A	SYNS	SYSAOS	Install Delta Save DLOG area
6B	SYNS	SYSAOS	Change Delta Save DLOG area
6C	SYNS	SYSAOS	Remove Delta Save DLOG area
6E	SYNS	ADADBS	ADADBS REPLICATION function (refer to your Event Replicator for Adabas documentation)
6F	SYNS	SYSAOS	Online process initiated
70	SYNS	SYSAOS	Online invert process
71	SYNS	SYSAOS	Online reorder process
73	SYNC	ADANUC	Nucleus (nuclei) successfully quiesced.
74	SYNC	ADANUC	Nucleus (nuclei) have resumed normal processing.
75	SYNS	ADANUC	Delete heuri-user-entry after Response 72 was detected during nucleus startup.
76	SYNS	ADANUC	Delete heuri-user-entry after Response 72 was detected during nucleus session.
77	SYNS	ADADBS	Enable spanned record support.
78	SYNS	ADADBS	Enable or disable extended MU or PE fields

136 JCL/JCS Requirements and Examples

- BS2000 842
- z/OS 843
- z/VM 844
- VSE 845

This section describes the job control information required to run ADAREP with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Save tape	DDSAVE	tape/ disk	Only with SAVETAPE
Protection log	DDPLOG	tape/ disk	Option with online save tape
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAREP parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAREP messages	SYSLST DDDRUCK		*Messages and Codes*

ADAREP JCL Example (BS2000)

In SDF Format:

```
/.ADAREP LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E P ALL FUNCTIONS
 /REMARK *
 /ASS-SYSLST L.REP
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAREP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAREP CPLIST
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAREP LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E P ALL FUNCTIONS
 /REMARK *
 /SYSFILE SYSLST=L.REP
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAREP,DB=yyyyy,IDTNAME=ADABAS5B
 ADAREP CPLIST
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Save tape	DDSAVE	tape/ disk	Only with SAVETAPE
Protection log	DDPLOG	tape/ disk	Option with online save tape
ADARUN parameters	DDCARD	reader	*Operations*
ADAREP parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAREP messages	DDDRUCK	printer	*Messages and Codes*

ADAREP JCL Example (z/OS)

All Functions

```
//ADAREP    JOB
//*
//*   ADAREP: ALL FUNCTIONS
//*
//REP      EXEC  PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDDRUCK   DD   SYSOUT=X
```

```
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAREP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAREP    CPEXLIST
/*
```

Refer to ADAREP in the JOBS data set for this example.

Report from a Save Tape

```
//ADAREPS   JOB
//*
//*   ADAREP: REPORT FROM A SAVE TAPE
//*
//REP      EXEC  PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDSAVE    DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.SAVE   <=== SAVE DATASET
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAREP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAREP    REPORT SAVETAPE,NOCOUNT
/*
```

Refer to ADAREPS in the JOBS data set for this example.

z/VM

Data Set	DD Name	Medium	Additional Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Save tape	DDSAVE	tape/ disk	Only with SAVETAPE
Protection log	DDPLOG	tape/ disk	Option with online save tape
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAREP parameters	DDKARTE	disk/ terminal/ reader	

Data Set	DD Name	Medium	Additional Information
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAREP messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAREP Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAREP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADAREP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNREP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAREP.CONTROL,MODE=A
ADARUN
```

Contents of RUNREP CONTROL A1

```
ADARUN PROG=ADAREP,DEVICE=dddd,DB=yyyyy
```

Contents of ADAREP CONTROL A1

```
ADAREP CPLIST
```

VSE

Data Set	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk		
Data Storage	DATARn	disk		
Save tape	SAVE	tape disk	SYS010 see note	Only with SAVETAPE
Protection log	PLOG	tape disk	SYS011 see note	Option with online save tape
ADARUN parameters	SYSRDR CARD CARD	reader tape disk	SYSRDR SYS000 see note	*Operations*
ADAREP parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	*Messages and Codes*

Data Set	Symbolic Name	Storage	Logical Unit	More Information
ADAREP report		printer	SYS009	

Note: Any programmer logical unit may be used.

ADAREP JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

All Functions

```
* $$ JOB JNM=ADAREP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAREP
*       ALL FUNCTIONS
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAREP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAREP    CPEXLIST
/*
/&
* $$ EOJ
```

Refer to member ADAREP.X for this example.

Report from a Save Tape

```
* $$ JOB JNM=ADAREPS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAREPS
*       REPORT FROM A SAVE TAPE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYSTEN,TAPE
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cuu
// TLBL SAVE,'EXAMPLE.DByyyyy.SAVE'
// MTC REW,SYS010
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAREP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAREP    REPORT SAVETAPE,NOCOUNT
/*
/&
* $$ EOJ
```

Refer to member ADAREPS.X for this example.

XVIII ADARES Utility: Database Recovery

This chapter covers the following topics:

Functional Overview
BACKOUT Functions
CLCOPY: Copy Dual Command Log
COPY: Copy a Sequential Protection Log or Save Tape
MERGE CLOG: Merge Nucleus Cluster Command Logs
PLCOPY: Copy Protection Log to Sequential Data Set
REGENERATE: Regenerate Updates
REPAIR: Repair Data Storage Blocks
Multithreaded Processing Statistics
JCL/JCS Requirements and Examples

137 Functional Overview

- Information about using ADARES 852

The ADARES utility performs functions related to database recovery.

Information about using ADARES

1. The functions BACKOUT (except BACKOUT DPLOG or MPLOG), REGENERATE, and REPAIR require a sequential data set containing protection log data as input. A dual/multiple protection log cannot be used directly. To convert dual/multiple protection logs to sequential logs, use the ADARES PLCOPY function.
2. The REGENERATE, BACKOUT, and COPY functions can process several sessions within one job if the following is true:
   - The DDSIIN/SIIN input file contains the sessions in ascending order by session number; gaps in the session number sequence exist only for those sessions representing save data set generations;
   - Each session on the file begins with block number 1, and there are no gaps in block numbering.
3. To select a *single* session only, specify the session number with PLOGNUM, or with FROMPLOG without TOPLOG; to specify a *range* of sessions, specify FROMPLOG and TOPLOG.
4. The ADARES COPY function accepts ADASAV output save (DD/SAVEn) files. No parameters indicating a save file can be specified; ADARES recognizes a save file by its structure. Only one save file can be copied during an ADARES COPY run. When copying a save file, specify the session number with PLOGNUM.
5. Adabas expanded files: the BACKOUT and REGENERATE (file) functions process complete expanded files only. If the file specified is either the anchor or component file of an expanded file, all other component files of the expanded file must also be specified.
6. To perform the additional Delta Save Facility operations of ADARES, ADARUN parameter DSF=YES must be specified in the DD/CARD input.
7. Multithreaded BACKOUT, BACKOUT DPLOG or MPLOG, and REGENERATE require additional main memory, which can be estimated using the values for the corresponding nucleus ADARUN parameter NT and NU:

   ```
   (NT x 70,000) + (NU x 72)
   ```

   For example, if NT=28 and NU=1000, about 2MB of main memory is required.
8. For optimal processing when using the multithreaded backout/regenerate feature, Software AG recommends that you set the nucleus ADARUN parameter NAB to at least

   ```
   NAB=NT x (32K + 108) / 4096
   ```

Using ADARES in Adabas Nucleus Cluster Environments

In an Adabas nucleus cluster environment, the protection logs (and optionally, the command logs) of all individual nuclei in the cluster are merged into single log files in chronological order for the cluster database shared by all the nuclei as a whole. The chronological order is determined by timestamps on all individual nucleus log records, which are synchronized across the cluster by the operating system.

Merging Logs

For recovery processing, all protection log data sets (PLOGs) must be merged into a single log stream for each cluster database. PLOGs are merged automatically when an ADARES PLCOPY is executed. The PLCOPY process accesses the parallel participant table (PPT) to determine which PLOGs to copy and uses dynamic allocation to access the appropriate data sets.

An existing PLCOPY job must be modified to run in a cluster environment. The user exit 2 may also need to be modified. A sample PLCOPY job ADARESMP that illustrates the necessary addition of the intermediate data sets and a sample user exit 2 (USEREX2P) is provided. See **Automatically Copy/Merge Nucleus Cluster Protection Logs**. It is not necessary to remove the PLOG DD statements, however. If they remain, they are ignored.

By default, dual/multiple command log data sets (CLOGs) can be copied to a sequential data set for each nucleus using the ADARES CLCOPY function, but the resulting data sets are not then automatically merged across the cluster into a single CLOG data set for the cluster database. You can choose to merge the CLCOPY output from each nucleus manually by using the ADARES MERGE CLOG function. By default, the CLOG data sets must be specified in the user exit 2 JCL; they are not dynamically allocated.

However, for accounting or other tracking purposes, you may want to automate the CLOG merge process the same way the PLOG merge process is automated. When you specify ADARUN CLOGMRG=YES, the CLOG merge process is invoked automatically when the ADARES CLCOPY job is submitted from UEX2 and executed. ADARUN LOGGING=YES must also be specified. As with the PLCOPY process, the CLCOPY process then accesses the parallel participant table (PPT) to determine which CLOGs to copy and uses dynamic allocation to access the appropriate data sets.

Existing CLCOPY jobs must be modified to include the intermediate data sets. A sample CLCOPY job ADARESMC is provided that illustrates the necessary addition of the intermediate data sets. See **Automatically Copy/Merge Nucleus Cluster Command Logs**. The sample user exit 2 (USEREX2P) includes both CLCOPY and PLCOPY functionality for the merge.

The automated PLCOPY and CLCOPY jobs copy/merge as much data as possible; if a nucleus is still writing to a log data set, the job 'partially' merges the data set.

Intermediate Data Sets

The merge begins with the lowest timestamp from all PLOGs and CLOGs being merged and ends with the lowest of the ending timestamps from all data sets. Records beyond this point

are written to an 'intermediate' data set, which must be supplied as input to the subsequent merge. A cross-check ensures that the correct intermediate data set has been supplied.

ADARES expects that at least one of the PLOGs or CLOGs being merged is at 'completed' status. If this is not the case, ADARES reports that there is no data to be copied.

A sample user exit 2 (USEREX2P for both PLOGs and CLOGs) is provided that illustrates the necessary JCL for the intermediate data sets. When intermediate data sets are used for both CLCOPY and PLCOPY jobs, the data set names for each must be unique so that they are not overwritten.

- PLCOPY example:

```
//MERGIN1 DD DISP=SHR,DSN=EXAMPLE.PINTERI
//MERGIN2 DD DISP=SHR,DSN=EXAMPLE.PINTERO
```

- CLCOPY example:

```
//MERGIN1 DD DISP=SHR,DSN=EXAMPLE.CINTERI
//MERGIN2 DD DISP=SHR,DSN=EXAMPLE.CINTERO
```

Depending on whether it is a PLCOPY or a CLCOPY, the job submitted by user exit 2 must refer to the appropriate set of statements.

Once DD statements for the PLOG data sets have been supplied on the session startup JCL, you do not need to supply them again for ADARES as these are opened using dynamic allocation. If the DD statements are supplied, they are ignored.

It is not necessary to manually change the JCL after each execution. ADARES maintains control information in the parallel participant table (PPT) to determine which intermediate data set to expect as input. It checks the header information in both data sets to determine which to use for input and which for output.

The following checks are made to ensure that the intermediate data set has been supplied correctly:

1. The DBID is stored in the intermediate data set header and must match the DBID in the log.
2. The log number is stored in the intermediate data set header and must either match or be one less than the current number from the log data set.
3. The STCK in the intermediate data set header must match the STCK stored in the PPT.

If any of the checks fails, ADARES ERROR 157 is returned.

ADARES also ensures that the intermediate data set contains the number of records expected. If not, ADARES ERROR 164 is returned.

Uniquely Identifying Checkpoints

After the protection log (PLOG) merge process, the block number will not necessarily be the same. To uniquely identify the checkpoint in this situation, it is necessary to also specify the NUCID for all functions that can specify a TOBLK/ FROMBLK parameter; that is, BACKOUT and REGENERATE.

The merge process ensures that there is at most one checkpoint per block. It records the (old) block number prior to the merge and the NUCID that wrote the checkpoint. When you then specify the block number and NUCID as reported in ADAREP, ADARES is able to uniquely identify the block.

Note: In an Adabas nucleus cluster environment, ADAREP includes the NUCID when printing all checkpoint information.

The additional parameters that are required in an Adabas nucleus cluster environment are NUCID, TONUCID, and FROMNUCID. If the NUCID is the same for the starting and ending checkpoint, only the NUCID needs to be specified.

Note: ADASAV stores this information in the header so that it can uniquely identify the block for the RESTONL and RESTPLOG functions.

138 BACKOUT Functions

Data protection information in the form of *before* and *after* images of all updated records is written to the protection log during each Adabas session. This information is needed to remove or reapply updates.

The protection log may be assigned to a sequential data set or to a dual/multiple protection log data set (direct access) on disk. If the dual/multiple protection log is used, the ADARES PLCOPY function must be used to copy it to a sequential data set. This data set can be used as input to ADARES BACKOUT or REGENERATE.

Software AG does not recommend the use of 3480/3490 tape cartridge compression (IDRC) for protection log files. The ADARES utility BACKOUT function runs at least twice as long under z/OS when processing compressed data. Also, the BACKOUT function is not supported for compressed data on VSE and z/VM systems.

The ADARES BACKOUT {DPLOG | MPLOG} function is not valid for a cluster database. This is disallowed because a merged PLOG is required in order to perform the BACKOUT.

BACKOUT	Back Out Updates Using the Sequential Protection Log (SIBA)
BACKOUT DPLOG or MPLOG	Back Out Updates Using the Dual or Multiple Protection Log

139 BACKOUT: Back Out Updates Using the Sequential Protection Log (SIBA)

- Essential Parameters 861
- Optional Parameters and Subparameters 861
- Examples 865

The BACKOUT function removes all the updates applied between two specified checkpoints. Both checkpoints must be contained on the sequential protection log input data set.

The BACKOUT function requires that the read backward feature is supported by the tape drive to be used for sequential input.

Note: An interrupted BACKOUT run must be reexecuted from the beginning.

You can specify either the log number (PLOGNUM) or the session number (FROMPLOG) of the protection log as a starting point for BACKOUT processing. If you specify a session number, you can also specify a range of sessions to be processed using the TOPLOG parameter.

By default, ADARES processes the database specified by the ADARUN DBID parameter. If BACKOUT processing is required against a different database, use the PLOGDBID parameter to specify the database.

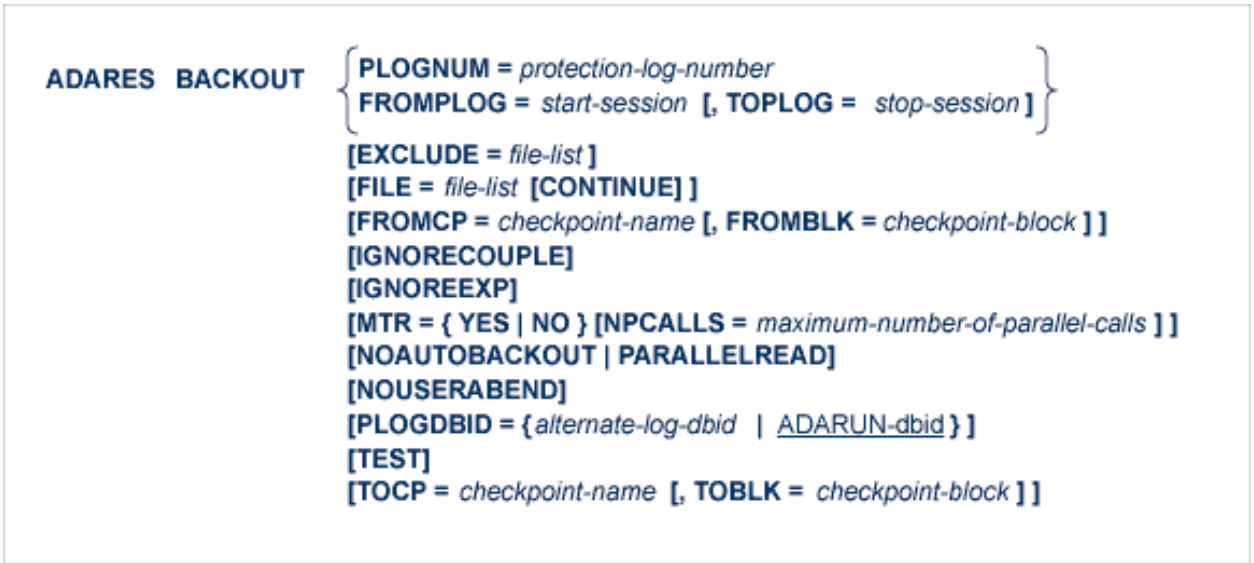
By default, BACKOUT processing continues until the end of the input data set is reached. You can limit the extent of BACKOUT processing using the TOCP parameter.

By default, all files in the specified input data set are included in the BACKOUT processing. You have to option to identify specific files to be included.

At the end of BACKOUT processing, ADARES automatically backs out all incomplete logical transactions when BACKOUT is specified for the entire database and continues until the end of the input data set is reached. This also occurs if

- the FILE parameter and the CONTINUE parameter are both specified; and
- the TOCP parameter is not specified.

You can override this process by specifying the NOAUTOBACKOUT parameter.



Essential Parameters

PLOGNUM: Protection Log Number

PLOGNUM specifies the log number of the sequential protection log to be used as input for BACKOUT processing. The log number may be obtained from the database status report.

FROMPLOG: Starting Session for BACKOUT

FROMPLOG specifies the session number at which BACKOUT processing is to start. ADARES searches the sequential PLOG input (DD/SIIN) file for the correct starting session. To define the starting point more precisely, specify the FROMCP and FROMBLK parameters.

Optional Parameters and Subparameters

CONTINUE: Continue File Recovery with Autobackout

When FILE is specified, CONTINUE locks the complete database for exclusive use by the BACKOUT function.

It allows autobackout of incomplete transaction changes, if any, during file backout. If specified, all changes made by incomplete transactions are backed out of the database data sets specified by the FILE parameter.

If the file list contains coupled or expanded component files and CONTINUE is specified, the usual default checking of the list for all coupled and/or remaining component files does not occur; in this case, IGNORECOUPLE or IGNOREEXP does not have to be specified to stop the checking.

EXCLUDE: Exclude Specified Files from Backout

EXCLUDE lists the numbers of the files to be excluded from BACKOUT processing; that is, the files that are not to be backed out. Any protection records that pertain to these files are ignored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

When the FILE parameter is specified, all files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter has no bearing on whether the BACKOUT is performed with or without transaction logic.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

Excluded files are listed in the extended CPLIST of the ADAREP report.

FILE: Files to Be Included

If all files are to be included in the BACKOUT processing, this parameter should not be specified.

If the specified file is a component file of an Adabas expanded file, then all other component files for the expanded file must also be specified. If a specified file is coupled to other files, these files must also be specified.

FROMBLK: Beginning Block for BACKOUT

FROMBLK specifies the block number containing the FROMCP checkpoint entry. This block number, which may be obtained from the database status report, refers to either PLOGNUM or FROMPLOG. FROMBLK can only be specified if FROMCP is specified.

FROMCP: Beginning Checkpoint for BACKOUT

FROMCP specifies the checkpoint before which the backout process is to begin. The checkpoint identification (name), which may be obtained from the database status report, refers to either PLOGNUM or FROMPLOG.

If backout processing is to begin at the end of the log, this parameter should be omitted.

IGNORECOUPLE: Ignore Unspecified Couple Files

IGNORECOUPLE (or CONTINUE) stops the BACKOUT function from checking the FILE list for complete coupled file pairs. If neither CONTINUE nor IGNORECOUPLE are specified and the FILE list specifies a coupled file without specifying its mate, ADARES terminates and issues an error message.

IGNOREEXP: Ignore Expanded Component Files

If the FILE list includes Adabas expanded component files, ADARES BACKOUT normally checks to ensure that all additional component files *related to the listed component files* are also in the list; if not, ADARES ends the BACKOUT operation and issues an error message. Specifying IGNOREEXP (or CONTINUE) stops the checking for related component files.

MTR: Multithreaded Regenerate Switch

MTR=YES activates the multithreaded regenerate feature; MTR=NO disables it.

When the multithreaded regenerate feature is active, multiple buffers containing PLOG information are sent to the Adabas nucleus in parallel to improve performance. When the feature is not active, only one buffer is sent to Adabas at a time.

If the nucleus ADARUN parameter MODE=SINGLE, MTR is automatically set to NO. Multiple threads are not available to Adabas running in single user mode.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for MTR is YES. In these cases, multithreaded regenerate has exclusive control of the whole database and is generally effective.

Otherwise, the default value is NO. If it only has exclusive control of some files, as is the case when FILE is specified *without* CONTINUE, multithreaded regenerate can run in parallel with normal applications accessing different files and has the potential to negatively impact the performance of production applications.

NOAUTOBACKOUT: Prevent Incomplete Transaction Backout

If several consecutive BACKOUT runs are necessary in order to process multiple protection logs resulting from a single Adabas session, an automatic backout should be performed only for the last input log. The NOAUTOBACKOUT parameter should therefore be specified for each BACKOUT run except the run in which the last input log is used.

Notes:

1. NOAUTOBACKOUT cannot be specified in single-user mode.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NPCALLS: Maximum Number of Parallel Calls

When MTR=YES, the NPCALLS parameter may be specified to limit the number of parallel calls sent to the Adabas nucleus.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for NPCALLS is the nucleus ADARUN parameter NT+1 or NC, whichever is smaller.

If the FILE parameter is specified without CONTINUE, the default value is the nucleus ADARUN parameter NT+1 or NC/2, whichever is smaller.

NPCALLS is primarily used to reduce the number of parallel calls allowed by the default value. Fewer parallel calls mean a smaller nucleus workload produced by ADARES. This is especially useful for increasing the resources available to application programs running in parallel with BACKOUT FILE.

PARALLELREAD: Enable Read-Only File Usage for Other Users

The PARALLELREAD parameter provides for concurrent read-only access to the files being processed by ADARES BACKOUT both for database-wide and file-oriented functions:

- for file-oriented functions, specifying PARALLELREAD causes ADARES to issue an OPEN call with "EXU=file-list" in the record buffer. This allows read-only access to the files for other users while ADARES is active.
- when FILE is not specified or when CONTINUE is specified, the PARALLELREAD parameter is effective for a database-wide session backout. The parameter makes it possible for read-only users to access the database at the same time the database session is being backed out.

Update commands are rejected.

If parallel access users read records that were updated in the database session being backed out, they may see record images that are logically wrong in the sense of the application, or response codes such as 113 that indicate inconsistencies.

Notes:

1. During ADARES operation with PARALLELREAD, temporary differences between the Associator and Data Storage may cause nucleus responses 113 or 199 to occur.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

PLOGDBID: Alternate Protection Log ID

When performing a backout operation using a protection log from a database other than that specified by the ADARUN statement's DBID parameter, PLOGDBID specifies the database ID of the alternate protection log. The default is the database ID from the ADARUN-specified database.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TOBLK: Ending TOCP Block

TOBLK specifies the block number containing the TOCP checkpoint entry. This block number, which can be obtained from the database status report, refers to either PLOGNUM or FROMPLOG, or to TOPLOG, if specified. TOBLK can only be specified if TOCP is specified.

TOCP: Ending Checkpoint Block for Backout

TOCP specifies the checkpoint at which the backout process is to be terminated. If backout processing is to continue until the beginning of the log, do not specify TOCP. The checkpoint identification (name), which can be obtained from the database status report, refers to either TOPLOG, if specified, or to PLOGNUM or FROMPLOG.

TOPLOG: Ending PLOG Session for Backout

TOPLOG specifies the last session to be processed by the specified ADARES function. If ADARES finds a session on the PLOG input (DD/SIIN) file whose session number is outside the inclusive range defined by FROMPLOG/TOPLOG, that session is excluded from ADARES processing. TOPLOG can only be specified if FROMPLOG is also specified. If TOPLOG is not specified, the FROMPLOG session becomes the default. To define the ending point more precisely, specify the TOCP and TOBLK parameters.

Examples

Example 1:

```
ADARES BACKOUT PLOGNUM=3
```

All files are to be included in backout processing. The protection log number is 3. Backout processing is to begin at the end of the log and is to end at the beginning of the log. At the end of the backout processing, an automatic backout (but moving forward) of incomplete transactions occurs.

Example 2:

```
ADARES BACKOUT
FILE=4,7,PLOGNUM=11,FROMCP=CH18,FROMBLK=1864,
ADARES                TOCP=CH01,TOBLK=1
```

The backout is to be limited to files 4 and 7. All updates applied to files 4 and 7 between the taking of checkpoints CH01 and CH18 are to be removed. CH01 is located in block 1 of data protection log 11. Checkpoint CH18 is located in block 1864 of data protection log 11. No automatic backout of incomplete transactions occurs.

140 BACKOUT DPLOG or MPLOG: Back Out Updates Using the Dual or Multiple Protection Log

- Executing the Function 868
- Syntax 869
- Essential Parameter 870
- Optional Parameters 870
- Example 874

The BACKOUT {DPLOG | MPLOG} function removes all the updates applied between two checkpoints contained on the same Adabas dual or multiple protection log data set, respectively.

The BACKOUT {DPLOG | MPLOG} function is not valid for a cluster database. This is disallowed because a merged PLOG is required in order to perform the BACKOUT.

Executing the Function

The following sequence is recommended for executing the BACKOUT DPLOG or MPLOG function:

1. Issue the operator or Online System command FEOFPL.

   Force EOF on the current protection log data set and switch to a new one. The new protection log data set will contain all information required for BACKOUT DPLOG or MPLOG.
2. Run the user application.

   All protection log data written by the nucleus for this application must fit on a single protection log data set. No protection log switch may occur while the application is running. Here, you should assume that the application program has failed, and must be backed out.
3. Issue again the operator or Online System command FEOFPL.

   Close the protection log data set. The closed data set contains all information required for BACKOUT DPLOG or MPLOG.
4. Run ADARES PLCOPY.

   Copy the content of the protection log data set to a sequential data set. This can be done by running ADARES PLCOPY or by using user exit 2 for DPLOG or user exit 12 for MPLOG.
5. Run ADARES BACKOUT DPLOG or MPLOG.

   This backs out the session to the status of step 1.

   An interrupted BACKOUT DPLOG or MPLOG run must be reexecuted from the beginning. If the data on the protection log data set that is to be used is unavailable (the nucleus uses this data protection log again), a BACKOUT from the sequential copy must be done.

Syntax

In general, the parameters FROMCP/TOCP/FROMBLK/TOBLK should not be specified. Software AG recommends that you back out using the entire contents of one protection log data set.

During backout, the nucleus writes new protection log information to the protection log data set currently available. This is the only data set that can be used by the nucleus. In case of a protection log switch during BACKOUT DPLOG or MPLOG, the nucleus waits until the complete data set has been copied with ADARES PLCOPY, and then resumes the backout run.

By default, all files in the specified input data set are included in the BACKOUT processing. You have the option to identify specific files to be included.

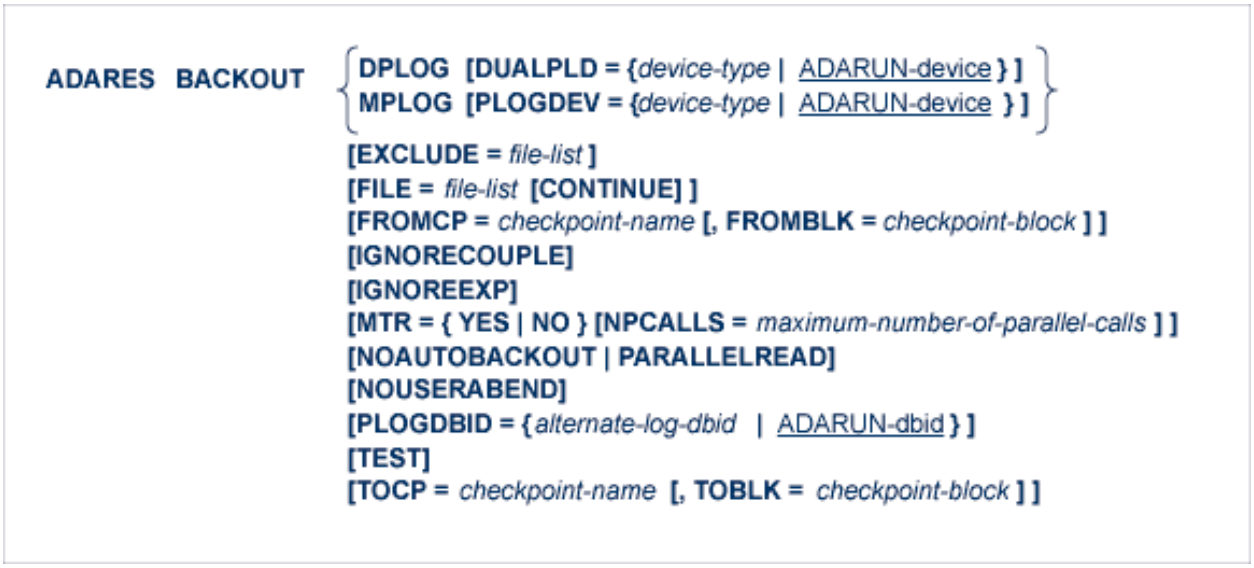
CONTINUE allows autobackout of incomplete transaction changes, if any, during file backout. If specified, all changes made by incomplete transactions are backed out of the database data sets specified by the FILE parameter. If the file list contains coupled or expanded component files and CONTINUE is specified, the usual default checking of the list for all coupled and/or remaining component files does not occur; in this case, IGNORECOUPLE or IGNOREEXP does not have to be specified to stop the checking.

Specifying CONTINUE locks the complete database for exclusive use by the BACKOUT function during file backout.

At the end of BACKOUT processing, ADARES automatically backs out all incomplete logical transactions when BACKOUT is specified for the entire database and continues until the end of the input data set is reached. This also occurs if

- the FILE parameter and the CONTINUE parameter are both specified; and
- the TOCP parameter is not specified.

You can override this process by specifying the NOAUTOBACKOUT parameter.



Essential Parameter

DPLOG | MPLOG: Dual or Multiple PLOG Source

DPLOG indicates that a dual protection log data set is to be used as input; MPLOG indicates that a multiple protection log data set is to be used as input.

Optional Parameters

CONTINUE: Continue File Recovery with Autobackout

When FILE is specified, CONTINUE locks the complete database for exclusive use by the BACKOUT function.

It allows autobackout of incomplete transaction changes, if any, during file backout. If specified, all changes made by incomplete transactions are backed out of the database data sets specified by the FILE parameter.

If the file list contains coupled or expanded component files and CONTINUE is specified, the usual default checking of the list for all coupled and/or remaining component files does not occur; in this case, IGNORECOUPLE or IGNOREEXP does not have to be specified to stop the checking.

DUALPLD | PLOGDEV: PLOG Device Type

DUALPLD specifies the device type used for the dual protection log data sets; PLOGDEV specified the device type used for the multiple protection log data sets. The default is the device type specified by the ADARUN DEVICE parameter.

EXCLUDE: Exclude Specified Files from Backout

EXCLUDE lists the numbers of the files to be excluded from BACKOUT processing; that is, the files that are not to be backed out. Any protection records that pertain to these files are ignored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

When the FILE parameter is specified, all files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter has no bearing on whether the BACKOUT is performed with or without transaction logic.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

Excluded files are listed in the extended CPLIST of the ADAREP report.

FILE: Files to be Included

FILE specifies the files to be included in the backout process. If all files are to be included, this parameter should be omitted. If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified here. If a specified file is coupled to other files, the coupled files must also be specified.

Note: Before beginning, ADARES locks all specified files for the duration of BACKOUT execution. If the FILE parameter is omitted, the entire database will be locked. Other users can have read-only access to the specified files if the UTYPE=EXU parameter is specified or to the database if the PARALLELREAD parameter is specified.

FROMCP: Beginning Checkpoint for Backout

FROMCP specifies the checkpoint before which the backout process is to begin. The checkpoint ID may be obtained from the database status report. If backout processing is to begin at the end of the log, do not specify the FROMCP parameter.

FROMBLK: Beginning FROMCP Block Number

FROMBLK is the block number containing the FROMCP checkpoint entry. This block number may be obtained from the database status report. FROMBLK can be specified only if FROMCP is specified.

IGNORECOUPLE: Ignore Unspecified Coupled Files

IGNORECOUPLE (or CONTINUE) stops the BACKOUT function from checking the FILE list for complete coupled file pairs. If neither CONTINUE nor IGNORECOUPLE are specified and the FILE list specifies a coupled file without specifying its mate, ADARES terminates and issues an error message.

IGNOREEXP: Ignore Expanded Component Files

If the FILE list includes any Adabas expanded component files, ADARES BACKOUT normally checks to ensure that all additional component files *related to the listed component files* are also

in the list; if not, ADARES ends the BACKOUT operation and issues an error message. Specifying IGNOREEXP (or CONTINUE) stops the checking for related component files.

MTR: Multithreaded Backout Switch

MTR=YES activates the multithreaded backout feature; MTR=NO disables it.

When the multithreaded backout feature is active, multiple buffers containing PLOG information are sent to the Adabas nucleus in parallel to improve performance. When the feature is not active, only one buffer is sent to Adabas at a time.

If the nucleus ADARUN parameter MODE=SINGLE, MTR is automatically set to NO. Multiple threads are not available to Adabas running in single user mode.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for MTR is YES. In these cases, multithreaded backout has exclusive control of the whole database and is generally effective.

Otherwise, the default value is NO. If it only has exclusive control of some files, as is the case when FILE is specified *without* CONTINUE, multithreaded backout can run in parallel with normal applications accessing different files and has the potential to negatively impact the performance of production applications.

NOAUTOBACKOUT: Prevent Incomplete Transaction Backout

If several consecutive BACKOUT runs are necessary in order to process multiple protection logs resulting from a single Adabas session, an automatic backout should be performed only for the last input log. The NOAUTOBACKOUT parameter should therefore be specified for each BACKOUT run except the run in which the last input log is used.

Notes:

1. NOAUTOBACKOUT cannot be specified in single-user mode.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NPCALLS: Maximum Number of Parallel Calls

When MTR=YES, the NPCALLS parameter may be specified to limit the number of parallel calls sent to the Adabas nucleus.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for NPCALLS is the nucleus ADARUN parameter NT+1 or NC, whichever is smaller.

If the FILE parameter is specified without CONTINUE, the default value is the nucleus ADARUN parameter NT+1 or NC/2, whichever is smaller.

NPCALLS is primarily used to reduce the number of parallel calls allowed by the default value. Fewer parallel calls mean a smaller nucleus workload produced by ADARES. This is especially useful for increasing the resources available to application programs running in parallel with BACKOUT DPLOG FILE.

PARALLELREAD: Enable Read-Only File Usage by Other Users

The PARALLELREAD parameter provides for concurrent read-only access to the files being processed by ADARES BACKOUT DPLOG both for database-wide and file-oriented functions:

- for file-oriented functions, specifying PARALLELREAD causes ADARES to issue an OPEN call with "EXU=file-list" in the record buffer. This allows read-only access to the files for other users while ADARES is active.
- when FILE is not specified or when CONTINUE is specified, the PARALLELREAD parameter is effective for a database-wide DPLOG backout. The parameter makes it possible for read-only users to access the database at the same time the database DPLOG is being backed out.

Update commands are rejected.

If parallel access users read records that were updated in the database DPLOG being backed out, they may see record images that are logically wrong in the sense of the application, or response codes such as 113 that indicate inconsistencies.

Notes:

1. During ADARES operation with PARALLELREAD, temporary differences between the Associator and Data Storage may cause nucleus responses 113 or 199 to occur.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

PLOGDBID: Alternate Protection Log ID

When performing a backout operation using a protection log from a database other than that specified by the ADARUN statement's DBID parameter, PLOGDBID specifies the database ID of the alternate protection log. The default is the database ID from the ADARUN-specified database.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TOBLK: Ending TOCP Block Number

TOBLK specifies the block number containing the TOCP checkpoint entry. TOBLK can only be specified if TOCP is specified.

TOCP: Ending Checkpoint for Backout

TOCP specifies the checkpoint at which the backout process is to stop. Backout processing continues up to the specified checkpoint. If backout processing is to continue until the beginning of the log, do not specify TOCP or NOAUTOBACKOUT.

Example

1. ADADBS OPERCOM FEOFPL
2. User application on files 20 and 21 fails
3. ADADBS OPERCOM FEOFPL
4. ADARES BACKOUT DPLOG,FILE=20,21

This example assumes that the PLCOPY function is performed with user exit 2. Whenever a protection log switch occurs, this user exit submits a job to copy the content of the dual protection log to a sequential data set.

1. Switch to a new PLOG.
2. Run the user session creating PLOG data on the new PLOG data set.
3. Close the PLOG data set. User exit 2 submits a job which copies the contents of the PLOG data set just closed.
4. Perform a BACKOUT from that PLOG for the files 20 and 21 up to the beginning of the PLOG.

141 CLCOPY: Copy Dual Command Log

- Optional Parameters 876
- Examples 877

The CLCOPY function is used only if dual logging of command information was specified for the Adabas session. This function copies the data set that has the earlier time stamp to a sequential data set. Once the CLCOPY function is completed successfully, the copied data set is marked as empty. This function may, therefore, be used only once for any given data set.

The CLCOPY function is not allowed in single-user mode.

```
ADARES  CLCOPY  [DUALCLD = {device-type | ADARUN-device } ]
                [NOUSERABEND]
                [OPENOUT]
                [TEST]
                [TWOCOPIES]
```

Optional Parameters

ADARES CLCOPY can be specified with no parameters.

DUALCLD: Dual Command Log Device Type

DUALCLD specifies the device type used for the dual command log data sets. This parameter is required if the device type used for the command log data set is different from that specified with the ADARUN DEVICE parameter.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OPENOUT: Open DDSIAUS1/2 or SIAUS1/2 Output Data Sets

The OPENOUT parameter indicates that the DD/SIAUS1/2 output data sets are to be opened by ADARES, even if no data is actually to be copied. Without OPENOUT, the sequential output data sets are not opened if ADARES detects an end-of-file condition while attempting to read the first input record; this may cause problems in some operating system environments. With OPENOUT, the output data sets are opened before the first input record is read.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TWOCOPIES: Create Two Copies of Output

TWOCOPIES causes two copies of the output to be created.

Examples

Example 1:

```
ADARES CLCOPY
```

Dual command log is to be copied.

Example 2:

```
ADARES CLCOPY TWOCOPIES
```

Dual command log is to be copied. Two copies of the output are to be created.

142 COPY: Copy a Sequential Protection Log or Save Tape

- Optional Parameters 881
- Examples 882

The COPY function copies an Adabas sequential protection log data set. If the Adabas session that created the sequential protection log data set was terminated abnormally, the COPY function must be executed before the data set can be used as input to any other ADARES function.

ADARES COPY

- must be used to copy a data protection log data set from disk to a tape data set before it can be used as input to the ADARES BACKOUT function.
- may be used even if subsequent Adabas sessions have created other data protection log data sets.
- also accepts ADASAV SAVE output (DD/SAVEn) as input. Only one ADASAV SAVE input volume can be copied in a single ADARES COPY run. A SAVE output tape must be assigned to the DD/SIIN job control file.
- may be executed any number of times for a given input data set.

```
ADARES  COPY    [PLOGNUM = protection-log-number
                 FROMPLOG = start-session  [, TOPLOG = stop-session ]]
                [NONUC]
                [NOUSERABEND]
                [OPENOUT]
                [RLOGDEV = { device-type | ADARUN-device } ]
                [TEST]
                [TWOCOPIES]
                [UTICPLIST]
```

The COPY function has special uses if you are using the Adabas Delta Save Facility. Refer to the *Adabas Delta Save Facility* documentation for more information.

ADARES COPY can be specified with no parameters. If ADARES COPY is specified without either PLOGNUM or FROMPLOG, the whole input protection log is copied.

Optional Parameters

FROMPLOG: Beginning Session for Copy

FROMPLOG specifies the session number at which the specified ADARES function is to start. ADARES searches the PLOG input (DD/SIIN) file for the correct starting session.

NONUC: Ignore Nucleus Response

The NONUC parameter may be used to ignore the nucleus response code in cases where the Adabas nucleus is unable to perform an autorestart. ADARES then attempts to physically copy the input tape as far as possible.

This parameter should only be specified after all other corrective actions have failed; however, NONUC *must* be specified if the PLOG sequential input (DD/SIIN) file contains several nucleus sessions to be copied.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OPENOUT: Open DDSIAUS1/2 or SIAUS1/2 Output Data Sets

The OPENOUT parameter indicates that the DD/SIAUS1/2 output data sets are to be opened by ADARES, even if no data is actually to be copied. Without OPENOUT, the sequential output data sets are not opened if ADARES detects an end-of-file condition while attempting to read the first input record; this may cause problems in some operating system environments. With OPENOUT, the output data sets are opened before the first input record is read.

PLOGNUM: Protection Log Number

The Adabas protection log number of the data set to be copied. This number may be obtained from the database status report produced by the ADAREP utility. The output of the COPY function will be assigned the same log number.

RLOGDEV: Device Type for RLOG Data Set

The RLOGDEV parameter is used if the Adabas Recovery Aid (ADARAI) is active to specify a device-type for the recovery log (RLOG) data set.

If RLOGDEV is not specified (the default), the recovery log device-type is assumed to be the same as the ADARUN DEVICE parameter.

If the specified or default value for RLOGDEV is incorrect, ADARES COPY terminates with error 149, "missing or mismatching RLOGDEV parameter".

The RLOGDEV parameter makes it possible for ADARES to record its function for ADARAI, even if the GCBs of the database have been destroyed.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TOPLOG: Ending PLOG Session for Backout

TOPLOG specifies the last session to be processed by the specified ADARES function. If ADARES finds a session on the PLOG input (DD/SIIN) file that is greater than the specified TOPLOG session, that session is excluded from ADARES processing.

TWOCOPIES: Create Two Output Copies

TWOCOPIES causes two copies of the output to be created. If TWOCOPIES is not specified, the default is one copy.

UTICPLIST: Print All Utility Checkpoints

The UTICPLIST parameter causes ADARES to select and print all SYNP, SYNV, and SYNS checkpoints found on the data protection log during the COPY function.

Examples

Example 1:

```
ADARES COPY PLOGNUM=6
```

Data protection log 6 is to be copied.

Example 2:

```
ADARES COPY PLOGNUM=8,TWOCOPIES
```

Data protection log 8 is to be copied. Two copies of the output are to be created.

143 MERGE CLOG: Merge Nucleus Cluster Command Logs

- Essential Parameter 884

In an Adabas cluster environment, you can merge command logs (CLOGs) across a cluster in one of two ways:

- If your system is set up appropriately (CLOGMRG=YES and user exit 2), CLOGs are merged automatically.
- Otherwise, you can merge CLOGs manually using the ADARES MERGE CLOG utility function.

```
ADARES  MERGE  CLOG, NUMLOG = nn
```

Sequential data sets are expected as input to the MERGE CLOG function; therefore, the ADARES CLCOPY function must be executed prior to the ADARES MERGE function.

The timestamp contained in the CLOGLAYOUT=5 is required for the proper merging of command logs records.

Essential Parameter

NUMLOG: Number of Command Log Data Sets

The NUMLOG parameter is required: it specifies the number of command log data sets to be included in the merge process. The maximum number is 32.

144 PLCOPY: Copy Protection Log to Sequential Data Set

- Optional Parameters 886
- Examples 888

The PLCOPY function is used only if dual/multiple logging of protection information was specified for the Adabas session. This function copies the data set that has the earlier time stamp to a sequential data set. Once the PLCOPY function is successfully completed, the copied data set is marked as empty. This function may, therefore, be used only once in an Adabas session for any given data set.

The use of hardware compression (IDRC) is *not* recommended for protection log files. The ADARES BACKOUT function is not supported for hardware-compressed data on VSE and z/VM systems. On z/OS systems, the BACKOUT function will take at least twice as long to run when processing compressed data.

The PLCOPY function is not allowed in single-user mode.

```
ADARES  PLCOPY  [PLOGDEV = { device-type  |  ADARUN-device } ]
                [NOPPT]
                [NOUSERABEND]
                [OPENOUT]
                [RLOGDEV = { device-type  |  ADARUN-device } ]
                [SBLKNUM = starting-block-num ]
                [TEST]
                [TWOCOPIES]
                [UTICPLIST]
```

The PLCOPY function has special uses if you are using the Adabas Delta Save Facility. Refer to the *Adabas Delta Save Facility* documentation for more information.

ADARES PLCOPY can be specified with no parameters.

Optional Parameters

PLOGDEV: PLOG Device Type

PLOGDEV specifies the device type used for dual/multiple protection log data sets. This parameter is required if the device type used for the dual/multiple protection log data set is different from that specified with the ADARUN DEVICE parameter.

NOPPT (Clustered Nucleus Environments Only)

The parallel participant table (PPT) tells ADARES PLCOPY which data sets to copy. If the PPT is destroyed, the ADARES NOPPT function allows the DBA to specify the PLOG data sets that are to be copied and merged.

If ADARAI is used, the PLOG data sets are written to the RLOG at nucleus initialization. In the event of a failure and a final PLCOPY is still needed, ADARAI can construct the PLCOPY NOPPT JCL from the PLOG data sets written to the RLOG.

NOPPT is intended only for emergency use when the PPT has been overwritten. It specifies that the PPT is to be ignored and DD/PLOG data sets are to be supplied with JCL.

Caution: Use this parameter cautiously since it ignores the PPT and all control-type information typically provided by the PPT.

When you use this parameter, you must supply

- the correct intermediate data set; and
- the correct input protection logs from all nuclei in the form of DD/PLOG01-nn.

Caution: Without the PPT, ADARES cannot perform any extensive validations on the input data sets.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OPENOUT: Open DDSIAUS1/2 or SIAUS1/2 Output Data Sets

The OPENOUT parameter indicates that the DD/SIAUS1/2 output data sets are to be opened by ADARES, even if no data is actually to be copied. Without OPENOUT, the sequential output data sets are not opened if ADARES detects an end-of-file condition while attempting to read the first input record; this may cause problems in some operating system environments. With OPENOUT, the output data sets are opened before the first input record is read.

RLOGDEV: Device Type for RLOG Data Set

The RLOGDEV parameter is used if the Adabas Recovery Aid (ADARAI) is active to specify a device-type for the recovery log (RLOG) data set.

If RLOGDEV is not specified (the default), the recovery log device type is assumed to be the same as the ADARUN DEVICE parameter.

If the specified or default value for RLOGDEV is incorrect, ADARES PLCOPY terminates with error 149, "missing or mismatching RLOGDEV parameter".

The RLOGDEV parameter makes it possible for ADARES to record its function for ADARAI, even if the GCBs of the database have been destroyed.

SBLKNUM

The SBLKNUM parameter can only be specified in conjunction with the NOPPT parameter and only for the PLCOPY function.

SBLKNUM allows the user to specify the starting block number for the sequential merge output. If this parameter is omitted, an attempt will be made to read the PPT and obtain the block number from there. If this read fails, the output will start with block one.

To determine the value for this parameter, the user must look at the output from the previous PLCOPY and use the next block number in sequence.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TWOCOPIES: Create Two Copies of Output

TWOCOPIES causes two copies of the output to be created. If TWOCOPIES is not specified, the default is one copy.

UTICPLIST: Print All Utility Checkpoints

The UTICPLIST parameter causes ADARES to select and print all SYNP, SYNV, and SYNS checkpoints found on the data protection log during the PLCOPY function.

Examples

Example 1:

Copy the dual/multiple protection log.

```
ADARES PLCOPY
```

Example 2:

Create two copies of the dual/multiple protection log.

```
ADARES PLCOPY TWOCOPIES
```

Example 3:

Copy the dual/multiple protection log. The Adabas Recovery Aid (ADARAI) is active. The recovery log (RLOG) device type is 8390.

```
ADARES PLCOPY RLOGDEV=8390
```

145 REGENERATE: Regenerate Updates

- Syntax 890
- Essential Parameters 891
- Optional Parameters and Subparameters 891
- Examples 896

The REGENERATE function reapplies all the updates performed between two checkpoints.

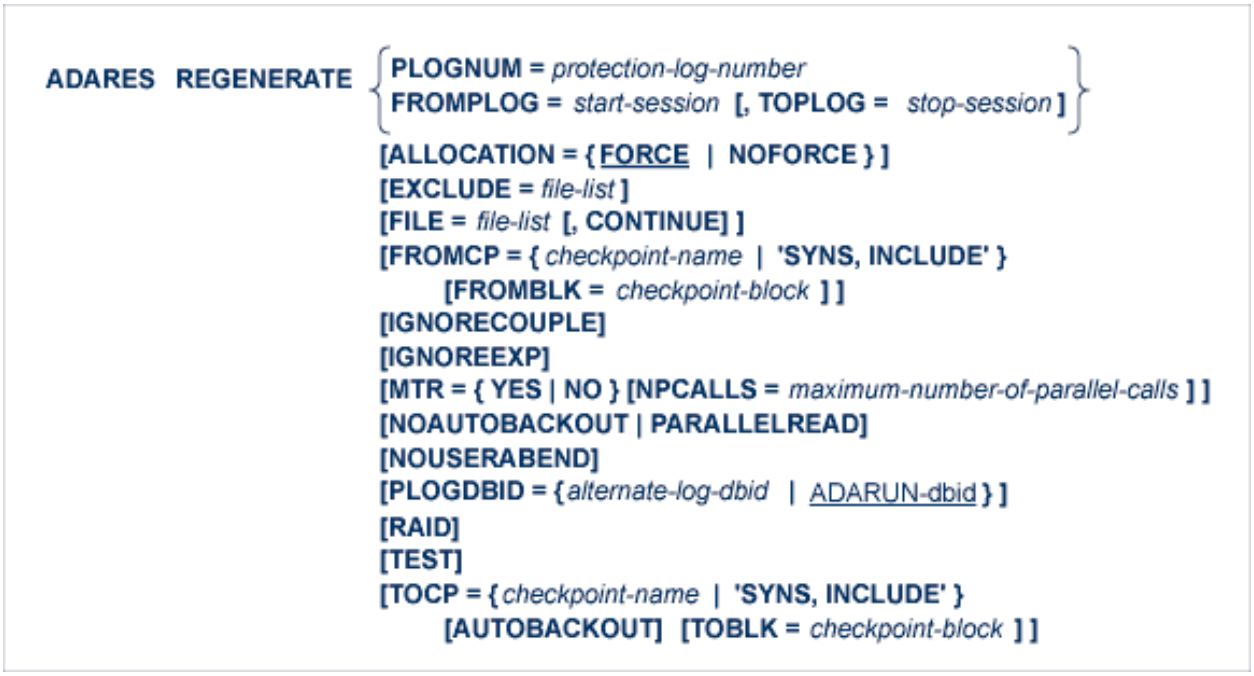
In addition to restoring normal updates, ADARES REGENERATE also restores any of the following ADADBS utility (or Adabas Online System) function updates that were performed between the specified checkpoints for the selected file or files:

ALLOCATE	DELETE	NEWFIELD	RELEASE
CHANGE	DSREUSE	PRIORITY	RENAME
DEALLOCATE	ISNREUSE	RECOVER	RENUMBER
DELCP	MODFCB	REFRESH	UNCOUPLE

For the database, all file-related operations listed above are performed, plus any of the following ADADBS (or Adabas Online System) database-related functions:

ADD	INCREASE (data set size)
DECREASE (data set size)	RECOVER

Syntax



Essential Parameters

You can specify either the log number (PLOGNUM) or the session number (FROMPLOG) of the protection log as a starting point for REGENERATE processing. If you specify a session number, you can also specify a range of sessions to be processed using the TOPLOG parameter.

FROMPLOG: Beginning Session for Regeneration

FROMPLOG specifies the session number at which the specified ADARES function is to start. ADARES searches the PLOG input file for the correct starting session. To define the starting point more precisely, specify the FROMCP and FROMBLK parameters.

PLOGNUM: Protection Log Number

PLOGNUM is the log number of the data protection log to be used as input for regenerate processing. The log number may be obtained from the database status report.

Optional Parameters and Subparameters

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

ALLOCATION concerns the following operations, which are replayed as part of the regeneration:

- ADADBS ALLOCATE
- Adabas Online System "Define File"
- Adabas Online System "Install/Change DLOG Area"

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

AUTOBACKOUT: Back Out Transactions from TOCP Checkpoint

When the TOCP parameter is specified, incomplete transactions are not normally backed out at the end of processing. This allows you to reexecute the utility function that corresponds to the TOCP checkpoint, followed by another ADARES operation with FROMCP specifying the starting checkpoint.

In situations where a REGENERATE/BACKOUT should end at the TOCP checkpoint, using the AUTOBACKOUT parameter to back out incomplete transactions ensures the logical consistency of the database. Note that AUTOBACKOUT is allowed only if TOCP is specified.

CONTINUE: Continue File Recovery with Autobackout

CONTINUE allows AUTOBACKOUT of any incomplete transaction changes during file regeneration. If specified, all changes made by incomplete transactions are backed out of the database data sets specified by the FILE parameter.

If the file list contains either coupled or expanded component files and CONTINUE is specified, the usual checking of the list for inclusion of complete coupled pairs and/or component file sets is not performed; in this case, IGNORECOUPLE or IGNOREEXP does not have to be specified to stop the respective file list check.

If CONTINUE is specified, the complete database is locked for use by the REGENERATE function only.

EXCLUDE: Exclude Specified Files from Regenerate

EXCLUDE lists the numbers of the files to be excluded from REGENERATE processing; that is, the files that are not to be regenerated. Any protection records that pertain to these files are ignored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

When the FILE parameter is specified, all files specified in the EXCLUDE parameter must also be specified in the FILE parameter.

The EXCLUDE parameter has no bearing on whether the REGENERATE is performed with or without transaction logic.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

Excluded files are listed in the extended CPLIST of the ADAREP report.

FILE: Files to Be Included

FILE specifies the files to be included in the regeneration process. If all files are to be included, do not specify the FILE parameter. If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified here. If a specified file is coupled to other files, the coupled files must also be specified.

Note: Before beginning, ADARES locks all specified files for the duration of REGENERATE execution. If the FILE parameter is omitted, the entire database will be locked.

FROMBLK: Starting Block for Regeneration

FROMBLK specifies the block number in which the FROMCP checkpoint entry is contained. This block number may be obtained from the previous ADASAV restore output or database status report. It refers to PLOGNUM or FROMPLOG. FROMBLK can be specified only if FROMCP is specified.

FROMCP: Starting Checkpoint for Regeneration

FROMCP defines the checkpoint after which the REGENERATE process is to begin. Processing begins with the information following the specified checkpoint. The checkpoint name may be obtained from the previous ADASAV restore output (SYN2/5), the database status report, or the ADARES COPY/PLCOPY output resulting from specifying UTICPLIST. If processing is to begin at the beginning of the log, do not specify the FROMCP parameter. FROMCP refers to the protection log specified by PLOGNUM or FROMPLOG.

For information about the 'SYNS,INCLUDE' option, see the section *INCLUDE: Include Checkpoint in Regeneration*.

IGNORECOUPLE: Ignore Unspecified Coupled Files

IGNORECOUPLE (or CONTINUE) stops the REGENERATE function from checking the FILE list for complete coupled file pairs. If neither CONTINUE nor IGNORECOUPLE is specified and the FILE list specifies a coupled file without specifying its mate, ADARES terminates and issues an error message.

IGNOREEXP: Ignore Expanded Component Files

If the FILE list includes any Adabas expanded component files, ADARES BACKOUT normally checks to ensure that all *related* component files are also in the list; if not, ADARES ends the REGENERATE operation and issues an error message. Specifying IGNOREEXP (or CONTINUE) stops the checking for related component files.

INCLUDE: Include Checkpoint in Regeneration

The optional keyword INCLUDE specified for FROMCP and/or TOCP includes the checkpoint where the regenerate starts/stops in the operation; that is, the function associated with the checkpoint is reexecuted. The checkpoint name must be SYNS, since ADARES can reexecute only functions associated with SYNS checkpoint. The checkpoint name and parameter combination 'SYNS,INCLUDE' must be enclosed in apostrophes.

If INCLUDE is not specified (the default), the REGENERATE operation starts immediately *after* the checkpoint specified by FROMCP and stops immediately *before* the checkpoint specified by TOCP.

The INCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

MTR: Multithreaded Regenerate Switch

MTR=YES activates the multithreaded regenerate feature; MTR=NO disables it.

When the multithreaded regenerate feature is active, multiple buffers containing PLOG information are sent to the Adabas nucleus in parallel to improve performance. When the feature is not active, only one buffer is sent to Adabas at a time.

If the nucleus ADARUN parameter MODE=SINGLE, MTR is automatically set to NO. Multiple threads are not available to Adabas running in single user mode.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for MTR is YES. In these cases, multithreaded regenerate has exclusive control of the whole database and is generally effective.

Otherwise, the default value is NO. If it only has exclusive control of some files, as is the case when FILE is specified *without* CONTINUE, multithreaded regenerate can run in parallel with normal applications accessing different files and has the potential to negatively impact the performance of production applications.

NOAUTOBACKOUT: Prevent Incomplete Transaction Backout

NOAUTOBACKOUT stops the normal backout of incomplete transactions at the end of REGENERATE operation. Normally, ADARES performs an automatic backout of all incomplete logical transactions at the end of the function if both of the following are true:

- The REGENERATE was for the entire database (FILE parameter omitted), or the CONTINUE parameter was specified; and
- The TOCP parameter was omitted, which implies that processing is to be performed until the end of the input data set is reached.

If several consecutive REGENERATE runs are needed to process multiple protection logs resulting from a single Adabas session, an automatic backout should be performed only for the last input log. The NOAUTOBACKOUT parameter should therefore be specified for each REGENERATE run except for the run in which the last input log is used.

Notes:

1. NOAUTOBACKOUT cannot be specified in single-user mode.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

NOUSERABEND: Terminate without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NPCALLS: Maximum Number of Parallel Calls

When MTR=YES, the NPCALLS parameter may be specified to limit the number of parallel calls sent to the Adabas nucleus.

If the FILE parameter is not specified, or is specified with CONTINUE, the default value for NPCALLS is the nucleus ADARUN parameter NT+1 or NC, whichever is smaller.

If the FILE parameter is specified without CONTINUE, the default value is the nucleus ADARUN parameter NT+1 or NC/2, whichever is smaller.

NPCALLS is primarily used to reduce the number of parallel calls allowed by the default value. Fewer parallel calls mean a smaller nucleus workload produced by ADARES. This is especially useful for increasing the resources available to application programs running in parallel with REGENERATE FILE.

PARALLELREAD: Enable Read-Only File Usage by Other Users

The PARALLELREAD parameter provides for concurrent read-only access to the files being processed by ADARES REGENERATE both for database-wide and file-oriented functions:

- for file-oriented functions, specifying PARALLELREAD causes ADARES to issue an OPEN call with "EXU=file-list" in the record buffer. This allows read-only access to the files for other users while ADARES is active.
- when FILE is not specified or when CONTINUE is specified, the PARALLELREAD parameter is effective for database-wide session regeneration. The parameter makes it possible for read-only users to access the database at the same time the database session is being regenerated.

Update commands are rejected.

If parallel access users read records that were updated in the database session being regenerated, they may see record images that are logically wrong in the sense of the application, or response codes such as 113 that indicate inconsistencies.

Notes:

1. During ADARES operation with PARALLELREAD, temporary differences between the Associator and Data Storage may cause nucleus responses 113 or 199 to occur.
2. NOAUTOBACKOUT is mutually exclusive with PARALLELREAD; only one of these parameters may be specified in an ADARES BACKOUT run.

PLOGDBID: Alternate Protection Log ID

PLOGDBID specifies an alternate DBID from which the PLOG has been taken. When regenerating with a protection log from a database other than that specified by the ADARUN statement's DBID parameter, use PLOGDBID to specify the database ID of the alternate protection log. The default is the database ID (DBID) from the ADARUN-specified database.

RAID: Action to Follow Receipt of Nucleus Response Code or Utility Checkpoint

The RAID parameter terminates a regeneration with error 146 whenever a file is to be excluded because a utility checkpoint (other than ADADBS or Adabas Online System checkpoints) was encountered or a nucleus response code was received for the file.

If RAID is not specified (the default), ADARES continues processing the other files after a file is excluded from REGENERATE processing.

RAID is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TOBLK: Ending TOCP Block

TOBLK specifies the block number in which the TOCP checkpoint entry is contained. TOBLK, which can be specified only if TOCP is also specified, refers to the protection log specified by TOPLOG, if specified, or else by PLOGNUM or FROMPLOG.

TOCP: Ending Checkpoint Block for Regenerate

TOCP specifies the checkpoint before which the REGENERATE process is to stop. Processing continues up to, but not including, the specified checkpoint. If REGENERATE processing is to continue until the end of the log, do not specify TOCP. TOCP refers to the protection log specified by TOPLOG, if specified, or else PLOGNUM or FROMPLOG.

For information about the 'SYNS,INCLUDE' option, see the section *INCLUDE: Include Checkpoint in Regeneration*.

TOPLOG: Ending PLOG Session for Regenerate

TOPLOG specifies the last session to be processed by the specified ADARES function. If ADARES finds a session on the PLOG input file that is greater than the specified TOPLOG session, that session is excluded from ADARES processing. If TOPLOG is not specified, the FROMPLOG session becomes the default.

Examples

Example 1:

```
ADARES REGENERATE PLOGNUM=4
```

All files are to be included in regenerate processing. The protection log number is 4. Regenerate processing is to begin at the beginning of the log and is to end at the end of the log. At the end of REGENERATE processing, incomplete transactions are automatically backed out.

Example 2:

```
ADARES REGENERATE
FILE=4,7,FROMPLOG=11,FROMCP=CH01,FROMBLK=106,
ADARES TOPLOG=12,TOCP=CH05,TOBLK=2031
```

Regenerate processing is to be limited to files 4 and 7. All updates applied to files 4 and 7 between the taking of checkpoints CH01 and CH05 are to be reapplied. CH01 is located in block 106 of data protection log 11. Checkpoint CH05 is located in block 2031 of data protection log 12. No automatic backout of incomplete transactions occurs following REGENERATE processing, as in the previous example.

Example 3:

```
ADARES REGENERATE EXCLUDE=10,11,12
```

Files 10 through 12 are excluded from the REGENERATE database function. No changes to these files are replayed.

Example 4:

```
ADARES REGENERATE
ADARES FROMCP='SYNS,INCLUDE',FROMBLK=123
ADARES TOCP=SYNP,TOBLK=234
```

1. ADARES regenerates the database.
2. The REGENERATE starts at the SYNS checkpoint in PLOG block 123; ADARES reexecutes the associated ADADBS/Adabas Online System function.
3. The REGENERATE stops just before the SYNP checkpoint in block 234; ADARES does *not* replay the associated utility function.

Example 5:

```
ADARES REGENERATE FILE=10
ADARES FROMCP='SYNS,INCLUDE',FROMBLK=345
ADARES TOCP='SYNS,INCLUDE',TOBLK=456
```

1. ADARES regenerates file 10.
2. The REGENERATE starts at the SYNS checkpoint in PLOG block 345; ADARES reexecutes the associated ADADBS/Adabas Online System function if it pertains to file 10.
3. The REGENERATE stops at the SYNS checkpoint in block 456; ADARES replays the associated ADADBS/Adabas Online System function if it pertains to file 10.

Example 6:

```
ADARES REGENERATE
ADARES RAID
```

1. ADARES regenerates the database.
2. ADARES reexecutes all database updates found on the input PLOG.
3. ADARES immediately terminates with error 146 if it receives a nucleus response code or encounters a utility checkpoint other than from ADADBS or Adabas Online System.

146 REPAIR: Repair Data Storage Blocks

- Syntax 900
- Essential Parameter 900
- Optional Parameters 901
- Examples 901

Caution: The REPAIR function can cause data loss if not used correctly. It should only be used with guidance from your Software AG technical support representative.

The REPAIR function may be used to repair one or more Data Storage blocks, using the protection log and the output of the ADASAV utility.

Notes:

1. An interrupted REPAIR function must be reexecuted from the beginning.
2. The REPAIR function should *not* be run if any of the following utility functions have changed the RABN ranges since the last ADASAV SAVE operation: `ADAORD`, `ADALOD`, `ADADBS DEALLOCATE`, `ADASAV RESTORE FMOVE`
3. The DDSIIN/SIIN input must be concatenated in the following sequence: `ADASAV SAVE (DD/SAVEn) output;,protection log.`

Syntax

```
ADARES  REPAIR   DSRABN = { rabn  |  rabn - rabn }
                 [FILE = { locked-file-list  |  entire-database } ]
                 [NOUSERABEND]
                 [TEST]
```

Essential Parameter

DSRABN: Data Storage RABN or RABNs to Be Repaired

DSRABN specifies one or more Data Storage RABNs to be repaired. Either a single RABN or a range of RABNs (for example, 1000-1234) can be specified.

Optional Parameters

FILE: Locked File List

FILE locks one or more files so that they cannot be read or updated by any user during REPAIR execution. Only the files specified are locked for the exclusive use of ADARES REPAIR. Files not included in the list remain available to other users of the database. If FILE is not specified, the entire database is locked; the user queue must be empty.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADARES REPAIR DSRABN=1434,FILE=20
```

Repair Data Storage block 1434. Only file 20 is locked during file processing.

Example 2:

```
ADARES REPAIR DSRABN=1462-2543
```

Repair Data Storage blocks 1462 through 2543.

147 Multithreaded Processing Statistics

When running ADARES BACKOUT, BACKOUT DPLOG, or REGENERATE with MTR=YES, a table with processing statistics is printed to DDDRUCK after successful completion of the utility. For example:

```
MULTI - THREADING PROCESSING STATISTIC
--------------------------------------

 PLOG BLOCKS READ FROM INPUT          20472
 PLOG RECORDS SENT TO ADABAS         764554
 COMMANDS PROCESSED                  302273
 TRANSACTION PROCESSED                55045
 NUMBER OF ADABAS CALLS               56450
 MAXIMUM CALLS IN PARALLEL               71
 AVERAGE RECORD BUFFER SIZE            1403
```

Field	Description
PLOG blocks read from input	Number of PLOG blocks read from the input protection log
PLOG records sent to Adabas	Number of PLOG records selected for backout or regenerate processing
Commands processed	Number of update commands processed (N1, E1, ...)
Transactions processed	Number of transactions backed out or regenerated
Number of Adabas calls	Number of Adabas calls issued to perform the backout or regenerate
Maximum calls in parallel	Maximum number of backout or regenerate calls processed in parallel by the nucleus
Average calls in parallel	Average number of backout or regenerate calls processed in parallel by the nucleus during this ADARES run
Average record buffer size	Average size of the record buffers used for the backout or regenerate calls

148 JCL/JCS Requirements and Examples

- BS2000 906
- z/OS 913
- z/VM 920
- VSE 925

This section describes the job control information required to run ADARES with BS2000, z/OS, VSE, and z/VM systems and shows examples of each of the job streams.

Notes:

1. The DD/SIAUS1/2 device type used to copy the protection log may not support the BACKOUT function if it is an IDRC (hardware compression) device. For more information, see the description of the PLCOPY function earlier in this document.
2. When running with the optional Recovery Aid (ADARAI), all temporary data sets must also be cataloged in the job control.

BS2000

Data Set	Link Name	Storage	More Information
Sequential protection log or ADASAV DDSAVEn output	DDSIIN	tape/ disk	Input log for COPY, REGENERATE, and REPAIR functions.
Multiple protection log	DDPLOGRn	disk	Input logs for PLCOPY function, and BACKOUT DPLOG.
Multiple command log	DDCLOGRn	disk	Input logs for CLCOPY function.
Sequential protection log	DDBACK	tape	Input log for BACKOUT function (not BACKOUT DPLOG).
Copied log	DDSIAUS1	tape/ disk	Output of COPY, CLCOPY, PLCOPY functions.
Extra copied log	DDSIAUS2	tape/ disk	Required only if two copies are to be produced by a copy function (with TWOCOPIES).
Data Storage	DDDATARn	disk	Required only for REGENERATE if FROMCP=SYN1 or SYN4.
Associator	DDASSORn	disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADARES parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADARES messages	SYSLST DDDRUCK		*Messages and Codes*

ADARES JCL Examples (BS2000)

Copy Dual/Multiple Command Log

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY DUAL/MULTIPLE COMMAND LOG
 /REMARK *
 /DELETE-FILE ADAyyyyy.AUS1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.AUS1,PUB(SPACE=(960,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.RES.CLCO
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDCLOGR1,ADAyyyyy.CLOGR1,SHARE-UPD=YES
 /SET-FILE-LINK DDCLOGR2,ADAyyyyy.CLOGR2,SHARE-UPD=YES
 /SET-FILE-LINK DDSIAUS1,ADAyyyyy.AUS1
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES CLCOPY
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY DUAL/MULTIPLE COMMAND LOG
 /REMARK *
 /SYSFILE SYSLST=L.RES.CLCO
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.CLOGR1 ,LINK=DDCLOGR1,SHARUPD=YES
 /FILE ADAyyyyy.CLOGR2 ,LINK=DDCLOGR2,SHARUPD=YES
 /FILE ADAyyyyy.AUS1 ,LINK=DDSIAUS1,SPACE=(960,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES CLCOPY
 /LOGOFF NOSPOOL
```

Copy Sequential Protection Log

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY SEQUENTIAL PLOG
 /REMARK *
 /DELETE-FILE ADAyyyyy.SIBA.COP1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.SIBA.COP1,PUB(SPACE=(960,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.RES.COPY
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDSIIN,ADAyyyyy.SIBA
 /SET-FILE-LINK DDSIAUS1,ADAyyyyy.SIBA.COP1
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES COPY PLOGNUM=ppp
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY SEQUENTIAL PLOG
 /REMARK *
 /SYSFILE SYSLST=L.RES.COPY
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.SIBA ,LINK=DDSIIN
 /FILE ADAyyyyy.SIBA.COP1,LINK=DDSIAUS1,SPACE=(960,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES COPY PLOGNUM=ppp
 /LOGOFF NOSPOOL
```

Copy Dual/Multiple Protection Log

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY DUAL/MULTIPLE PROTECTION LOG
 /REMARK *
 /DELETE-FILE ADAyyyyy.AUS1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.AUS1,PUB(SPACE=(960,480))
 /SET-JOB-STEP
 /DELETE-FILE ADAyyyyy.AUS2
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.AUS2,PUB(SPACE=(960,480))
 /SET-JOB-STEP
 /ASS-SYSLST L.RES.PLCO
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
 /SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
 /SET-FILE-LINK DDSIAUS1,ADAyyyyy.AUS1
 /SET-FILE-LINK DDSIAUS2,ADAyyyyy.AUS2
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES PLCOPY TWOCOPIES
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S COPY DUAL/MULTIPLE PROTECTION LOG
 /REMARK *
 /SYSFILE SYSLST=L.RES.PLCO
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.PLOGR1 ,LINK=DDPLOGR1,SHARUPD=YES
 /FILE ADAyyyyy.PLOGR2 ,LINK=DDPLOGR2,SHARUPD=YES
 /FILE ADAyyyyy.AUS1 ,LINK=DDSIAUS1,SPACE=(960,480)
 /FILE ADAyyyyy.AUS2 ,LINK=DDSIAUS2,SPACE=(960,480)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES PLCOPY TWOCOPIES
 /LOGOFF NOSPOOL
```

Backout Using a Sequential Protection Log

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S BACKOUT FROM SEQUENTIAL PLOG
 /REMARK *
 /ASS-SYSLST L.RES.BACK
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDBACK,ADAyyyyy.BACK
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES BACKOUT
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S BACKOUT FROM SEQUENTIAL PLOG
 /REMARK *
 /SYSFILE SYSLST=L.RES.BACK
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.BACK ,LINK=DDBACK
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES BACKOUT
 /LOGOFF NOSPOOL
```

Backout Using a Dual/Multiple Protection Log

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S BACKOUT FROM DUAL/MULTIPLE PLOG
 /REMARK *
 /ASS-SYSLST L.RES.BADP
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
```

```
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
 /SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES BACKOUT DPLOG
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S BACKOUT FROM DUAL/MULTIPLE PLOG
 /REMARK *
 /SYSFILE SYSLST=L.RES.BADP
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.PLOGR1 ,LINK=DDPLOGR1,SHARUPD=YES
 /FILE ADAyyyyy.PLOGR2 ,LINK=DDPLOGR2,SHARUPD=YES
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES BACKOUT DPLOG
 /LOGOFF NOSPOOL
```

Regenerate Function

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S REGENERATE
 /REMARK *
 /ASS-SYSLST L.RES.REGE
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDSIIN,ADAyyyyy.SIBA
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES REGENERATE FILE=1,CONTINUE,PLOGNUM=ppp
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S REGENERATE
 /REMARK *
 /SYSFILE SYSLST=L.RES.REGE
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.SIBA ,LINK=DDSIIN
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES REGENERATE FILE=1,CONTINUE,PLOGNUM=ppp
 /LOGOFF NOSPOOL
```

Repair Data Storage

In SDF Format:

```
/.ADARES LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A R E S REPAIR DATASTORAGE
 /REMARK *
 /ASS-SYSLST L.RES.REPA
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDSIIN,ADAyyyyy.SAVE
 /SET-FILE-LINK DDSIIN01,ADAyyyyy.PLOG5
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
 ADARES REPAIR DSRABN=3456 3490
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADARES LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A R E S REPAIR DATASTORAGE
 /REMARK *
 /SYSFILE SYSLST=L.RES.REPA
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.SAVE ,LINK=DDSIIN
```

```
/FILE ADAyyyyy.PLOG5 ,LINK=DDSIIN01
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADARES,DB=yyyyy,IDTNAME=ADABAS5B
ADARES REPAIR DSRABN=3456 3490
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Sequential protection log or ADASAV DDSAVEn output	DDSIIN	tape/ disk	Input log for COPY, REGENERATE, and REPAIR functions.
Multiple protection log	DDPLOGRn	disk	Input logs for PLCOPY function, and BACKOUT DPLOG/MPLOG .
Multiple command log	DDCLOGRn	disk	Input logs for CLCOPY function.
Sequential protection log	DDBACK	tape	Input log for BACKOUT function (not BACKOUT DPLOG).
Copied log	DDSIAUS1	tape/ disk	Output of COPY, CLCOPY, PLCOPY functions.
Extra copied log	DDSIAUS2	tape/ disk	Required only if two copies are to be produced by a copy function (with TWOCOPIES).
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
Data Storage	DDDATARn	disk	Required only for REGENERATE if FROMCP=SYN1 or SYN4.
Associator	DDASSORn	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADARES parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARES messages	DDDRUCK	printer	*Messages and Codes*

ADARES JCL Examples (z/OS)

Copy Sequential Protection Log

```
//ADARESCP  JOB
//*
//*    ADARES: COPY SEQUENTIAL PROTECTION LOG
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD        <=== ADABAS LOAD
//*

//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1  <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1  <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1  <=== WORK
//DDSIIN    DD   DSN=EXAMPLE.DByyyyy.SIBA,            <=== PLOG
//            VOL=SER=vvvvvv,DISP=OLD,UNIT=TAPE
//DDSIAUS1  DD   DSN=EXAMPLE.DByyyyy.PLOG(+1),        <=== PLOG COPY
//            VOL=SER=vvvvvv,UNIT=TAPE,DISP=(NEW,CATLG)
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADARES COPY
/*
```

Refer to ADARESCP in the JOBS data set for this example.

Copy Dual/Multiple Protection Log

```
//ADARESCD  JOB
//*
//*    ADARES: COPY DUAL/MULTIPLE PROTECTION LOG
//*             TWO COPIES OF OUTPUT ARE TO BE CREATED
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1  <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1  <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1  <=== WORK
//DDPLOGR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR1  <=== PLOG1
//DDPLOGR2  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR2  <=== PLOG2
//DDSIAUS1  DD   DSN=EXAMPLE.DByyyyy.PLOG1(+1),       <=== PLOG COPY 1
//            VOL=SER=vvvvvv,UNIT=TAPE,DISP=(NEW,CATLG)
//DDSIAUS2  DD   DSN=EXAMPLE.DByyyyy.PLOG2(+1),       <=== PLOG COPY 2
```

```
//            VOL=SER=vvvvvv,UNIT=TAPE,DISP=(NEW,CATLG)
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X


//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADARES PLCOPY TWOCOPIES
/*
```

Refer to ADARESCD in the JOBS data set for this example.

Automatically Copy/Merge Nucleus Cluster Protection Logs

Note: Note that when intermediate data sets are used for both CLCOPY and PLCOPY, the data set names must be unique so that they are not overwritten.

Following is sample JCL for allocating the required intermediate data sets MERGINT1 and MERGINT2:

```
//ALLOC JOB
//*
//* Example to allocate the MERGINT1 and the MERGINT2 data sets
//*
//ALLOC   EXEC PGM=IEFBR14
//MERGINT1 DD  DISP=(NEW,CATLG,DELETE),DSN=EXAMPLE.PINTERI,
//             SPACE=(CYL,(1,10,0)),UNIT=3390,VOL=SER=volser,
//             RECFM=VB,BLKSIZE=27998,LRECL=27994
//MERGINT2 DD  DISP=(NEW,CATLG,DELETE),DSN=EXAMPLE.PINTERO,
//             SPACE=(CYL,(1,10,0)),UNIT=3390,VOL=SER=volser,
//             RECFM=VB,BLKSIZE=27998,LRECL=27994
```

Refer to ADARESMP in the JOBS data set for this example.

Automatically Copy/Merge Nucleus Cluster Protection Logs Ignoring PPT

```
//ADARESIP  JOB

//*
//*    ADARES: COPY/MERGE DUAL/MULTIPLE PROTECTION LOGS FROM ALL
//*             NUCLEI IN AN ADABAS CLUSTER
//*             PPT IS TO BE IGNORED
//*             THIS IS ONLY FOR EMERGENCY USE WHEN THE PPT HAS BEEN
//*             OVER-WRITTEN - USE CAUTION WHEN SUBMITTING
//*
```

```
//RES      EXEC PGM=ADARUN

//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.ASSOR1   <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.DATAR1   <=== DATA
//DDPLOG01  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR1   <=== PLOG1 NUC1
//DDPLOG02  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR2   <=== PLOG2 NUC1
//DDPLOG03  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR1A  <=== PLOG1 NUC2
//DDPLOG04  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR2A  <=== PLOG2 NUC2
//DDPLOG05  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR1B  <=== PLOG1 NUC3
//DDPLOG06  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.PLOGR2B  <=== PLOG2 NUC3
//MERGINT2  DD   DISP=SHR,DSN=EXAMPLE.INTERO           <=== INTER
//MERGINT1  DD   DISP=SHR,DSN=EXAMPLE.INTERI           <=== INTER
//DDSIAUS1  DD   DSN=EXAMPLE.DBYYYYY.PLOG1(+1),        <=== PLOG COPY
//            VOL=SER=ADAXXX,UNIT=TAPE,DISP=(NEW,CATLG)
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=XXX,DEVICE=3390,DBID=YYYYY
/*
//DDKARTE   DD   *
ADARES PLCOPY NOPPT
/*
//
```

Refer to ADARESIP in the JOBS data set.

Copy Dual/Multiple Command Log

```
//ADARESCC  JOB
//*
//*     ADARES: COPY DUAL/MULTIPLE COMMAND LOG
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDCLOGR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.CLOGR1 <=== CLOG1
//DDCLOGR2  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.CLOGR2 <=== CLOG2
//DDSIAUS1  DD   DSN=EXAMPLE.DByyyyy.CLOG,           <=== OUTPUT OF
//            VOL=SER=vvvvvv,UNIT=TAPE,DISP=(NEW,CATLG)       CLCOPY

//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
```

```
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADARES CLCOPY
/*
```

Refer to ADARESCC in the JOBS data set for this example.

Automatically Copy/Merge Nucleus Cluster Command Logs

Note: Note that when intermediate data sets are used for both CLCOPY and PLCOPY, the data set names must be unique so that they are not overwritten.

Following is sample JCL for allocating the required intermediate data sets MERGINT1 and MERGINT2:

```
//ALLOC JOB
//*
//* Example to allocate the MERGINT1 and the MERGINT2 data sets
//*
//ALLOC   EXEC PGM=IEFBR14
//MERGINT1 DD  DISP=(NEW,CATLG,DELETE),DSN=EXAMPLE.CINTERI,
//             SPACE=(CYL,(1,10,0)),UNIT=3390,VOL=SER=volser,
//             RECFM=VB,BLKSIZE=27998,LRECL=27994
//MERGINT2 DD  DISP=(NEW,CATLG,DELETE),DSN=EXAMPLE.CINTERO,
//             SPACE=(CYL,(1,10,0)),UNIT=3390,VOL=SER=volser,
//             RECFM=VB,BLKSIZE=27998,LRECL=27994
```

Refer to ADARESMC in the JOBS data set for this example.

Manually Merge Sequential Command Logs in a Nucleus Cluster Environment

```
//ADARESCM  JOB
//*
//*    ADARES: MERGE SEQUENTIAL COMMAND LOGS
//*             FOR USE WITH AN ADABAS NUCLEUS CLUSTER
//*

//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.ASSOR1  <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.DATAR1  <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.WORKR1  <=== WORK
//DDCLOG01  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.CLOGR1A <=== CLOG1
//DDCLOG02  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.CLOGR1B <=== CLOG2
//DDCLOG03  DD   DISP=SHR,DSN=EXAMPLE.DBYYYYY.CLOGR1C <=== CLOG3
//DDSIAUS1  DD   DSN=EXAMPLE.DBYYYYY.CLOGM,           <=== OUTPUT OF
```

```
//            VOL=SER=ADAXXX,UNIT=TAPE,DISP=(NEW,CATLG)       CLOG MERGE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=XXX,DEVICE=3390,DBID=YYYYY
/*
//DDKARTE   DD   *
ADARES MERGE CLOG,NUMLOG=3
/*
//
```

Refer to ADARESCM in the JOBS data set for this example.

Backout from a Sequential Protection Log

```
//ADARESSP  JOB
//*
//*    ADARES: BACKOUT FROM A SEQUENTIAL PLOG
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDBACK    DD   DSN=EXAMPLE.DByyyyy.PLOG(-5),       <=== PLOG TAPE
//               UNIT=TAPE,DISP=OLD
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*

//DDKARTE   DD   *
ADARES BACKOUT PLOGNUM=nnn
/*
```

Refer to ADARESSP in the JOBS data set for this example.

Backout from Dual/Multiple Protection Log

```
//ADARESB   JOB
//*
//*    ADARES: BACKOUT FROM DUAL/MULTIPLE PLOG
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDPLOGR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR1 <=== PLOG1
//DDPLOGR2  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.PLOGR2 <=== PLOG2
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADARES BACKOUT DPLOG
/*
```

Refer to ADARESB in the JOBS data set for this example.

Regenerate Function

```
//ADARESR   JOB
//*
//*    ADARES: REGENERATE
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*

//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDSIIN    DD   DSN=EXAMPLE.DByyyyy.PLOG(-5),       <=== PLOG TAPE
//            UNIT=TAPE,DISP=OLD
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
```

```
ADARES REGENERATE FILE=1
/*
```

Refer to ADARESR in the JOBS data set for this example.

Repair Data Storage

```
//ADARESRP  JOB
//*
//*    ADARES:  REPAIR DATASTORAGE
//*
//RES      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDSIIN    DD   DSN=EXAMPLE.DByyyyy.SAVE,           <=== SAVE
OUTPUT
//             DISP=OLD,UNIT=TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(-5),       <=== PLOG TAPE
//             DISP=OLD,UNIT=TAPE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADARES REPAIR DSRABN=3456-3490
```

Refer to ADARESRP in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Sequential protection log or ADASAV DDSAVEn output	DDSIIN	tape/ disk	Input log for COPY, REGENERATE and REPAIR functions.
Multiple protection log	DDPLOGRn	disk	Input logs for PLCOPY and BACKOUT DPLOG.
Multiple command log	DDCLOGRn	disk	Input logs for CLCOPY.
Sequential protection log	DDBACK	tape	Input log for BACKOUT (not BACKOUT DPLOG).
Copied log	DDSIAUS1	tape/ disk	Output of CLCOPY and PLCOPY.

Data Set	DD Name	Storage	More Information
Extra copied log	DDSIAUS2	tape/ disk	Required for TWOCOPIES.
Data Storage	DDDATARn	disk	Required for REGENERATE if FROMCP=SYN1 or SYN4.
Associator	DDASSORn	disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using ADARAI.
ADARUN parameters	DDCARD	disk/ terminal/reader	*Operations*
ADARES parameters	DDKARTE	disk/ terminal/reader	
ADARUN messages	DDPRINT	disk/ terminal/printer	*Messages and Codes*
ADARES messages	DDDRUCK	disk/ terminal/printer	*Messages and Codes*

ADARES JCL Examples (z/VM)

Copy Sequential Protection Log

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDSIIN,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDSIAUS1,DSN=ADABASVv.SIAUS1,UNIT=181,VOL=SIBA01
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADARES.CONTROL,MODE=A
ADARUN
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of ADARES CONTROL A1

```
ADARES COPY PLOGNUM=ppp
```

Copy Dual/Multiple Protection Log

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOG1,VOL=PLOGV1
DATADEF DDPLOGR2,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDSIAUS1,DSN=ADABASVv.SIAUS1,UNIT=181,VOL=SIBA01
DATADEF DDSIAUS2,DSN=ADABASVv.SIAUS2,UNIT=182,VOL=SIBA02
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADARES.CONTROL,MODE=A
ADARUN
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of ADARES CONTROL A1

```
ADARES PLCOPY TWOCOPIES
```

Copy Dual/Multiple Command Log

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDCLOGR1,DSN=ADABASVv.CLOG1,VOL=CLOGV1
DATADEF DDCLOGR1,DSN=ADABASVv.CLOG1,VOL=CLOGV2
DATADEF DDSIAUS1,DSN=ADABASVv.CLOG1,UNIT=181,VOL=CLOG01
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADARES.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of ADARES CONTROL A1

```
ADARES CLCOPY
```

Backout Using a Sequential Protection Log

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDBACK,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=BACKOUT.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of BACKOUT CONTROL A1

```
ADARES BACKOUT
```

Backout Using a Dual/Multiple Protection Log

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOG1,VOL=PLOGV1
DATADEF DDPLOGR2,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=BACKOUT.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of BACKOUT CONTROL A1

```
ADARES BACKOUT DPLOG
```

Regenerate Function

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDSIIN,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=REGENER.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of REGENER CONTROL A1

```
ADARES REGENERATE FILE=1
```

Repair Data Storage

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDSIIN,DSN=ADABASVv.SAVE,MODE=A
DATADEF DDSIIN,DSN=ADABASVv.SIBA,MODE=A,CONCAT=1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOG1,VOL=PLOGV1
DATADEF DDPLOGR2,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDPRINT,DSN=ADARES.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADARES.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNRES.CONTROL,MODE=A
DATADEF DDKARTE,DSN=BACKOUT.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNRES CONTROL A1

```
ADARUN PROG=ADARES,DEVICE=dddd,DB=yyyyy
```

Contents of BACKOUT CONTROL A1

```
ADARES REPAIR DSRABN=3456-3490
```

VSE

Data Set	Symbolic	Storage	Logical Unit	More Information
Sequential protection log or ADASAV SAVEn output	SIIN	tape disk	SYS020 See note	Input log for COPY, REGENERATE, and REPAIR.
Multiple protection log	PLOGRn	disk	See note	Input logs for PLCOPY and BACKOUT DPLOG.
Multiple command log	CLOGRn	disk	See note	Input logs for CLCOPY.
Sequential protection log	BACK	tape	SYS020	Input log for BACKOUT (not BACKOUT DPLOG).
Copied log	SIAUS1	tape disk	SYS021 See note	Output of COPY, CLCOPY, and PLCOPY.
Extra copied log	SIAUS2	tape disk	SYS022 See note	Required for TWOCOPIES.
Data Storage	DATARn	disk		Required for REGENERATE if FROMCP=SYN1 or SYN4.
Associator	ASSORn		See note	
Recovery log (RLOG)	RLOGR1	disk		Required when using ADARAI.
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 See note	
ADARES parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	
ADARES messages		printer	SYS009	

Note: Any programmer logical unit can be used.

ADARES JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to the following members for these examples:

Example	Member
Copy sequential protection log	ADARESCP.X
Copy dual/multiple protection log	ADARESCD.X
Copy dual/multiple command log	ADARESCC.X
Regenerate	ADARESR.X
Backout from a sequential protection log	ADARESSP.X
Backout from a dual protection log	ADARESB.X
Repair Data Storage	ADARESRP.X

Copy Sequential Protection Log

```
* $$ JOB JNM=ADARESCP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESCP
*       COPY SEQUENTIAL PLOG(TAPE)
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS020,TAPE,DO
// PAUSE MOUNT INPUT TAPE ON TAPE cuu
// MTC REW,SYS020
// TLBL SIIN,'ADABAS.ADAvrs.SIBA'
// ASSGN SYS022,TAPE,DO
// PAUSE MOUNT SCRATCH TAPE ON TAPE cuu
// MTC REW,SYS022
// MTC WTM,SYS022,5
// MTC REW,SYS022
// TLBL SIAUS1,'ADABAS.ADAvrs.SIAUS1'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES COPY
/*
/&
* $$ EOJ
```

Copy Dual/Multiple Protection Log

```
* $$ JOB JNM=ADARESCD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESCD
*       CLCOPY WITH OPTION TWOCOPIES(TAPE)
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS021,TAPE,DO
// ASSGN SYS022,TAPE,DO
// PAUSE MOUNT SCRATCH TAPE ON TAPES cu1 AND cu2
// MTC REW,SYS022
// MTC WTM,SYS022,5
// MTC REW,SYS022
// MTC REW,SYS021
// MTC WTM,SYS021,5
// MTC REW,SYS021
// TLBL SIAUS1,'ADABAS.ADAvrs.PLOGC1'
// TLBL SIAUS2,'ADABAS.ADAvrs.PLOGC2'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES PLCOPY TWOCOPIES
/*
/&
* $$ EOJ
```

Copy Dual/Multiple Command Log

```
* $$ JOB JNM=ADARESCC,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESCC
*       COPY DUAL/MULTIPLE COMMAND LOG
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS022,DISK,VOL=vvvvvv,SHR
// DLBL SIAUS1,'ADABAS.ADAvrs.CLOG',0,SD
// EXTENT SYS022,vvvvvv,1,0,sssss,nnnnn
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES CLCOPY
/*
/&
* $$ EOJ
```

Regenerate

```
* $$ JOB JNM=ADARESR,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESR
*       REGENERATE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS020,DISK,VOL=vvvvvv,SHR
// DLBL SIIN,'EXAMPLE.DByyyyy.PLOG'
// EXTENT SYS020,vvvvvv
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES REGENERATE FILE=1
/*
/&
* $$ EOJ
```

Backout from a Sequential Protection Log

```
* $$ JOB JNM=ADARESSP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESSP
*       BACKOUT FROM A SEQUENTIAL PLOG
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS020,TAPE
// PAUSE  ***  PLEASE MOUNT TAPE ***
// MTC REW,SYS020
// TLBL BACK,'DByyyyy.PLCOPY.TAPE'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES BACKOUT PLOGNUM=ppp
/*
/&
* $$ EOJ
```

Backout from a Dual Protection Log

```
* $$ JOB JNM=ADARESB,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESB
*       BACKOUT FROM DUAL PLOG
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES BACKOUT DPLOG
/*
/&
* $$ EOJ
```

Repair Data Storage

```
* $$ JOB JNM=ADARESRP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADARESRP
*       REPAIR DATASTORAGE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS020,TAPE
// PAUSE MOUNT INPUT TAPE ON cuu
// MTC REW,SYS020
// TLBL SIIN,'EXAMPLE.ADAyyyyy.SAVE1'
// TLBL SIIN01,'ADABAS.ADAyyyyy.PLOG5'       (*)
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADARES,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADARES REPAIR DSRABN=3456-3490
/*
/&
* $$ EOJ
```

(*) See **Sequential Input Files**, **VSE concatenation**.

XIX ADASAV Utility: Save/Restore Database or Files

The ADASAV utility is used to saves and restores the contents of the database, specific files, or a file to or from a sequential data set. This chapter covers the following topics:

Functional Overview
RESTONL: Restore Database from Online Source
RESTONL FILES: Restore Files to Original RABNs from Online Source
RESTONL FMOVE: Restore Files to Any RABNs from Online Source
RESTONL GCB: Restore Database Incremental from Online Source
RESTORE: Restore Database from Offline Source
RESTORE FILES: Restore Files to Original RABNs from Offline Source
RESTORE FMOVE: Restore Files to Any RABNs from Offline Source
RESTORE GCB: Restore Database Incremental from Offline Source
RESTPLOG: Restore Protection Log Only
SAVE: Save Database
SAVE FILES: Save Specified Files
JCL/JCS Requirements and Examples

149 Functional Overview

- RESTONL and RESTORE Functions 934
- Adabas Release Support 935

The ADASAV utility saves and restores the contents of the database, specific files, or a file to or from a sequential data set.

ADASAV should be run as often as required for the number and size of the files contained in the database, and the amount and type of updating.

For large databases, ADASAV functions may be run in parallel for the various disk packs on which the database is contained.

Special ADASAV functions are available for use with the Adabas Delta Save Facility. For more information, see the *Adabas Delta Save Facility* documentation.

RESTONL and RESTORE Functions

For either RESTORE or RESTONL function operations, the Associator and Data Storage data sets must first be formatted. If either operation is interrupted, no database update activity should be attempted until the function has been successfully reexecuted.

RESTONL functions restore from a SAVE data set created while the Adabas nucleus was *active* (that is, online); RESTORE functions restore from a SAVE data set created while the Adabas nucleus was *inactive* (that is, offline).

RESTONL and RESTORE have the subfunctions GCB, FILES, and FMOVE:

- Without a subfunction, RESTONL and RESTORE restore entire databases.
- With the GCB subfunction, they restore the general control blocks (GCBs), Associator RABNs 3-30 of the database, and specified files.
- With the FILES subfunction, they restore one or more files into an existing database to their original RABNs.
- With the FMOVE subfunction, they restore one or more files into an existing database to any free space, allowing changes to extent sizes.

This section covers the following topics:

- RESTPLOG and RESTONL Functions

- Online and Offline SAVEs

RESTPLOG and RESTONL Functions

If changes occurred during the online SAVE, the RESTONL function is followed automatically by the RESTPLOG function. RESTPLOG applies the updates that occurred during, and therefore were not included in, the online SAVE.

RESTPLOG is also executed following a RESTONL or RESTONL FILES function that ended before completing restoration of protection log (PLOG) updates. RESTPLOG applies the database updates not applied by the unsuccessful RESTONL function.

Online and Offline SAVEs

The SAVE function to save a database, or one or more files may be executed while the Adabas nucleus is active (online) or inactive (offline). If the Recovery Aid option is active, a SAVE database operation begins a new RLOG generation.

Adabas Release Support

You can restore entire Adabas databases only using the same Adabas release used to create the save data set. However, you can restore individual files from save data sets created by earlier Adabas versions (down to version 5.1) using the RESTORE FILES, RESTORE FMOVE, RESTONL FILES, or RESTONL FMOVE functions.

Considerations when Restoring Files from Adabas version 5

When restoring from an Adabas 5 save data set, the RESTORE FILE function discards the unused RABN chains that may be present for the normal index or upper index. This makes all blocks of these chains “unreachable index blocks” as reported by the ADAICK ICHECK function in WARNING-163. These blocks will not be reused until they are reordered by the ADAORD REORFASSO or other reorder functions. The RESTORE FMOVE function does not discard the unused RABN chains, but rather transforms them to the new Adabas version structure.

If the database contains different device types for Data Storage and Work, restoring from an Adabas 5 save data set might be difficult if the Data Storage block size is larger than the Work block size. ADASAV may reject the restore because the maximum compressed record length of the file exceeds the length allowed by the Work block size. This is due to the increase in the length of protection record headers in later Adabas versions. To restore the file in this case, a new Work device type with a larger block size must be installed using the ADADEF NEWWORK function.

150 RESTONL: Restore Database from Online Source

- Conditions 938
- Result 939
- Syntax 939
- Optional Parameters and Subparameters 939
- Examples 941

The RESTONL function restores a database from a database SAVE data set created while the Adabas nucleus was *active*.

Notes:

1. An interrupted RESTONL (database) operation must be reexecuted from the beginning. If the interruption occurred while RESTONL (database) was restoring the PLOG, the restore operation can be completed using the RESTPLOG function. Until successful completion or reexecution of the restore operation, the database is inaccessible.
2. If the ADASAV RESTONL (database) job control contains the DD names, symbolic names, or link names for DD/WORKnn, these data sets are reset.

Conditions

To use the RESTONL (database) function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It must have been created by an online database SAVE operation with the same version of Adabas as is used for the RESTONL.

2 The output database must have the same physical layout (device types, extent sizes) as the original database. The Associator and Data Storage data sets must be present and must have been previously formatted. The SAVE data set to be restored may have originated for this or from a different database.

3 No Adabas nucleus may be active on the output database or on a database with the DBID of the output database.

4 The protection log (PLOG) data set containing information written by the nucleus session at the time of the SAVE operation (see output of SAVE run) must be supplied. PLOG data sets from other sessions may also be included.

5 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

Result

The result of this function is a database with the same physical status it had at the end of the ADASAV SAVE operation.

Syntax

```
ADASAV  RESTONL  [BUFNO = { number-of-buffers  |  1 } ]
                 [CLOGDEV = { CLOG1-device-type  |  ADARUN-device } ]
                 [DRIVES = { count  |  1 } ]
                 [EXCLUDE = file-list ]
                 [NEWDBID = new-database-id ]
                 [NEWDBNAME = new-database-name ]
                 [NOUSERABEND]
                 [OVERWRITE]
                 [PLOGDEV = { PLOG-device-type  |  ADARUN-device } ]
                 [PLOGNUM = PLOG-number  [, SYN1 = PLOG-block-number ] ]
                 [TEST]
```

Optional Parameters and Subparameters

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTONL operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

CLOGDEV: Command Log Device Type

The device type of the dual/multiple command log (CLOG). This parameter is required only if the device type of the CLOG is different from that specified by the ADARUN DEVICE parameter, which is the default.

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored. For a database restore:

- no files specified in the EXCLUDE parameter will exist in the restored database; and
- all files specified in the EXCLUDE parameter must exist on the save data set.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

NEWDBID: New ID for Restored Database

NEWDBID may be used to assign a different database ID to the restored database. The ID can be in the range 1-65,535; if Adabas Online System Security is installed, DBID 999 is reserved.

If NEWDBID is specified, the ADARUN DBID parameter must specify the ID of the database on the SAVE data set.

No Adabas nucleus may be active with the DBID specified on NEWDBID.

NEWDBNAME: New Name for Restored Database

NEWDBNAME assigns a new name to the restored database. If NEWDBNAME is not specified, the restored database keeps its old name.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing Database

If the restore operation is to overwrite an existing database, the OVERWRITE parameter must be specified. No Adabas nucleus may be active on the database to be overwritten.

PLOGDEV: Protection Log Device Type

The device type to be assigned to the dual/multiple protection log (PLOG). This parameter is required only if the device type of the PLOG is different from that specified by the ADARUN DEVICE parameter.

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the nucleus protection log used while the ADASAV SAVE operation was active (see output listing of the online SAVE function). Sequential protection (SIBA) logs from more than one nucleus session can be concatenated. ADASAV skips protection logs with a number lower than the PLOGNUM value. PLOGNUM is optional.

If PLOGNUM is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

SYN1: Beginning Block Number

SYN1 specifies the block number containing the SYN1 checkpoint at which the corresponding SAVE operation began (see output listing of the online SAVE function). This parameter is optional.

If SYN1 is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTONL
```

Restore the database saved when the nucleus was active (online). The protection log number and SYN1 block number required for the restore operation are determined automatically by ADASAV.

Example 2:

```
ADASAV RESTONL
ADASAV EXCLUDE=255
ADASAV EXCLUDE=400
```

Files 255 and 400 are excluded from the restore of the database from an online-save data set.

151 RESTONL FILES: Restore Files to Original RABNs from Online Source

- Conditions 944
- Result 945
- Syntax 945
- Optional Parameters and Subparameters 946
- Examples 949

The RESTONL FILES function restores files from a file or database SAVE data set created while the Adabas nucleus was *active*. One or more files can be restored. The files are restored into an existing database to their original RABNs.

Notes:

1. An interrupted RESTONL FILES operation must be reexecuted from the beginning. If the interruption occurred while RESTONL FILES was restoring the PLOG, the restore operation can be completed using the RESTPLOG function. Until successful completion or reexecution of the restore operation, the files to be restored are inaccessible.
2. Checkpoint and security files from Adabas version 5 cannot be restored.

Conditions

To use the RESTONL FILES function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It can be a database or file SAVE data set and must contain the files to be restored. SAVE data sets from version 5.1 or above can be used.

2 A file may be restored using a SAVE tape created from a different database; however, the device types must be identical.

3 An existing database must be present. The files to be restored may have originated from this or from a different database.

4 All RABNs originally used by the file(s) to be restored must either be free (available according to the free space table) or be occupied by files to be overwritten.

5 The Adabas nucleus may be active or inactive on the output database.

If the Adabas nucleus is active for restoring the checkpoint or security files, the ADASAV utility requires exclusive database control; that is, no user may be active on the database.

6 The protection log (PLOG) data set containing information written by the nucleus session at the time of the SAVE operation (see output of SAVE run) must be supplied. PLOG data sets from other sessions may also be included. If none of the files to be restored were modified during the online SAVE operation, the protection log data set(s) can be omitted.

7 If the SAVE tape was created with Adabas version 5.1, the location of the SYN1/SYN4 checkpoint written by the Adabas nucleus at the beginning of the online SAVE operation must be specified.

8 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

9 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

10 Expanded files and coupled files can only be restored or overwritten as a whole. That is, if one file in an expanded file is specified, all other files in the expanded file must be specified. If one file in a coupled relationship is specified, all other files in that relationship must be specified.

11 A checkpoint, security, trigger, or user-defined system file can be overwritten only by another checkpoint, security, trigger, or user-defined system file, respectively. A checkpoint, security, or trigger file cannot be restored if such a file already exists in the database with a different file number.

12 New file numbers can be assigned to the files to be restored using the NEWFILES parameter.

Result

The result of this function is the specified files with the same physical status they had at the end of the ADASAV SAVE operation.

Syntax

```
ADASAV  RESTONL  FILES = file-list  [ALLOCATION = { FORCE | NOFORCE } ]
                                    [BUFNO = { number-of-buffers | 1 } ]
                                    [DRIVES = { count | 1 } ]
                                    [EXCLUDE = file-list ]
                                    [NEWFILES = file-list ]
                                    [NOUSERABEND]
                                    [OVERWRITE]
                                    [PASSWORD = 'password-list' ]
                                    [PLOGNUM = protection-log-number ]
                                       [ { SYN1 | SYN4 } = PLOG-block-number ]
                                    [TEST]
```

The FILES file list specifies the file or files to be restored. If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified here. If a specified file is coupled to other files, the coupled files must also be specified.

The file list specified need not correspond to a file list used in the corresponding SAVE function. A file list may be specified even if no file list was used for the corresponding SAVE function.

A file may also be restored using a SAVE tape created from a different database; however, the device types must be identical.

Optional Parameters and Subparameters

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

ALLOCATION pertains to the implicit RABN specifications derived from the files on the save data set.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

Note: An ADASAV RESTONL FILES operation with ALLOCATION=NOFORCE specified *cannot* be completed with RESTPLOG if the function fails after completing the restore from the save data set but before completing the restore from the protection log.

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTONL operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

If the NEWFILES parameter

- is *not* specified, all files specified in the EXCLUDE parameter must also be specified in the FILES parameter.

- *is* specified, all files specified in the EXCLUDE parameter must also be specified in the NEWFILES parameter. In this case, the file numbers specified in the EXCLUDE parameter refer to the new file numbers in NEWFILES, not to the old file numbers in the FILES parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

NEWFILES: New File Numbers

The NEWFILES parameter specifies the new file number to be assigned to each file specified by FILES. The parameter is optional; if no new file number is assigned to a file, the file retains its original number. NEWFILES may not be specified for expanded files, physically coupled files, or replicated files.

If a file with a number specified by NEWFILES already exists in the database, the corresponding file will not be restored unless the OVERWRITE parameter is also specified. If the file to be overwritten is password-protected, the corresponding PASSWORD parameter must also be specified.

If several files are to be restored, the list of file numbers in the NEWFILES parameter must correspond to the list of files in the FILES parameter. If no new file number is to be assigned to a file, its entry in the file number list of NEWFILES must be specified as zero. See the **Examples**.

You can use NEWFILES to renumber a *base file* or *LOB file* only if both files of the *LOB file group* are restored. In this case, ADASAV assigns both files the new file numbers specified by the NEWFILES parameter and adjusts the links between the two files accordingly. However, if only one file of a *LOB file group* is restored, it cannot be assigned a new file number using the NEWFILES parameter; use the ADADBS or AOS RENUMBER function instead.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing File

This parameter causes an existing file to be deleted and then restored. If a file which is to be restored is already present in the database, ADASAV will skip this file unless the OVERWRITE parameter is supplied.

Note: To avoid unintentionally overwriting the database, Software AG recommends that you always specify the OVERWRITE parameter after, and not before, the FILES file list.

PASSWORD: File Password/Passwords

PASSWORD specifies one password or a list of passwords if one or more files in the FILES file list are password protected. This only applies to files already in the database that are to be overwritten. If the NEWFILES parameter is specified, the PASSWORD parameter must specify the passwords related to the new file numbers.

When restoring more than one password-protected file, the correct passwords must be specified as positional values corresponding to the protected file numbers' positions in the FILES list. Refer to the **Examples** for more information about the PASSWORD parameter. The Adabas nucleus must be active if password-protected files are being overwritten.

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the nucleus protection log (PLOG) used while the ADASAV SAVE operation was active (see output listing of the online SAVE function). This parameter is optional when restoring a SAVE tape created by ADASAV version 5.2 or above, or when none of the files to be restored were changed during the SAVE operation. Sequential protection (SIBA) logs from more than one nucleus session can be concatenated. ADASAV skips PLOGs with a number lower than the PLOGNUM value.

If PLOGNUM is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

Note: This is not possible when restoring from a version 5.1 SAVE data set.

SYN1|SYN4: Starting Block Number

The block number containing the SYN1/SYN4 checkpoint at which the restore operation is to begin (refer to the output listing of the online SAVE function for the block number). When restoring a SAVE tape created by ADASAV version 5.2 or above, this parameter is optional.

If SYN1/SYN4 is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

Note: This is not possible when restoring from a version 5.1 SAVE data set.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTONL FILES=3,4,5, OVERWRITE
ADASAV      PASSWORD='PWDFIL3,,PWDFIL5'
ADASAV      PLOGNUM=15,SYN1=20
```

Files 3, 4, and 5 are to be restored. Files 3 and 5 are password-protected and their passwords are PWDFIL3 and PWDFIL5. The PLOG number is 15 and the block containing the SYN1 checkpoint is 20. The old files are to be overwritten.

Example 2:

```
ADASAV RESTONL FILES=11,12,13,14,OVERWRITE
ADASAV      NEWFILES=16,0,17
```

Files 11, 12, 13, and 14 are to be restored. Files 11 and 13 are to be restored as files 16 and 17, respectively. The file numbers of files 12 and 14 will not be changed because the corresponding NEWFILES parameter values are specified as zero or omitted. Files 12, 14, 16, and 17 are to be overwritten, if already present in the database.

152 RESTONL FMOVE: Restore Files to Any RABNs from Online Source

- Conditions 952
- Result 953
- Syntax 953
- Optional Parameters 954
- Examples 961

The RESTONL FMOVE function restores files from a file or database SAVE data set created while the Adabas nucleus was *active*. One or more files can be restored. The files are restored into an existing database to any free space. Their extent sizes may be changed.

Notes:

1. An interrupted RESTONL FMOVE operation must be reexecuted from the beginning. It is not possible to use the RESTPLOG function to recover from an interrupted RESTONL FMOVE operation that ended while restoring the PLOG. Until successful completion or reexecution of the restore operation, the files to be restored are inaccessible.
2. Checkpoint and security files from Adabas version 5 cannot be restored.

Conditions

To use the RESTONL FMOVE function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It can be a database or file SAVE data set and must contain the files to be restored. SAVE data sets from Adabas version 5.1 or above can be used.

2 An existing database must be present. The files to be restored may have originated from this or from a different database.

3 The FMOVE file list specifies the file or files to be restored using new RABNs. The RABNs must be located on the same device type as used originally.

4 For the file(s) to be restored, sufficient space, either free space (according to the free space table) or space occupied by files to be overwritten, must be available in the database.

5 The Adabas nucleus may be active or inactive on the output database.

If the Adabas nucleus is active for restoring the checkpoint or security files, the ADASAV utility requires exclusive database control; that is, no user may be active on the database.

6 The protection log (PLOG) data set containing information written by the nucleus session at the time of the SAVE operation (see output of SAVE run) must be supplied. PLOG data sets from other sessions may also be included. If none of the files to be restored were modified during the online SAVE operation, the protection log data set(s) can be omitted.

7 If the SAVE tape was created with Adabas version 5.1, the location of the SYN1/SYN4 checkpoint written by the Adabas nucleus at the beginning of the online SAVE operation must be specified.

8 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

9 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

10 Expanded files and coupled files can only be restored or overwritten as a whole. That is, if one file in an expanded file is specified, all other files in the expanded file must be specified. If one file in a coupled relationship is specified, all other files in that relationship must be specified.

11 A checkpoint, security, trigger, or user-defined system file can be overwritten only by another checkpoint, security, trigger, or user-defined system file, respectively. A checkpoint, security, or trigger file cannot be restored if such a file already exists in the database with a different file number.

12 New file numbers can be assigned to the files to be restored using the NEWFILES parameter.

Result

The result of this function is the specified files with the same contents they had at the end of the ADASAV SAVE operation but not necessarily in the same database blocks.

Syntax

```
ADASAV  RESTONL  FMOVE = file-list  [ACRABN = AC-start-rabn-list ]
                                   [AC2RABN = AC2-start-rabn-list ]
                                   [ALLOCATION = { FORCE  |  NOFORCE } ]
                                   [ASSOVOLUME = 'Associator-extent-volume' ]
                                   [BUFNO = { number-of-buffers  |  1 } ]
                                   [DATAVOLUME = 'Data-Storage-extent-volume' ]
                                   [DRIVES = { count  |  1 } ]
                                   [DSRABN = DS-start-rabn-list ]
                                   [DSSIZE = DS-size-list ]
                                   [EXCLUDE = file-list ]
                                   [MAXISN = isn-count-list ]
                                   [MAXISN2 = isn-count-list ]
                                   [NEWFILES = file-list ]
                                   [NIRABN = NI-start-rabn-list ]
                                   [NISIZE = NI-size-list ]
                                   [NOUSERABEND]
                                   [OVERWRITE]
                                   [PASSWORD = 'password-list' ]
                                   [PLOGNUM = protection-log-number ]
                                      [ { SYN1 | SYN4 } = PLOG-block-number ]
                                   [TEST]
                                   [UIRABN = UI-start-rabn-list ]
                                   [UISIZE = UI-size-list ]
```

The FMOVE file list specifies the file or files to be restored using new RABNs. The RABNs must be located on the same device type as used originally.

If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified. If a specified file is coupled to other files, the coupled files must also be specified.

Optional Parameters

ACRABN: Starting Address Converter RABN/RABN List

ACRABN specifies the starting address converter RABN for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and ACRABN omitted, the location of the address converter is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the ACRABN parameter must correspond to the list of files in the FMOVE parameter. If no ACRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **examples**.

AC2RABN: Starting Secondary Address Converter RABN/RABN List

AC2RABN specifies the starting secondary address converter RABN for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and AC2RABN omitted, the location of the secondary address converter is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally. If the file contains no secondary address converter extents, this parameter is ignored.

If several files are to be restored, the list of RABNs in the AC2RABN parameter must correspond to the list of files in the FMOVE parameter. If no AC2RABN value is to be given for a file, its entry in the RABN list must be specified as zero.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOVOLUME: Associator Extent Volume

Note: The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated. If the requested number of blocks cannot be found on the specified volume, ADASAV retries the allocation while disregarding the ASSOVOLUME parameter.

If ACRABN, UIRABN, or NIRABN is specified, ADASAV ignores the ASSOVOLUME value when allocating the corresponding extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADASAV's default allocation rules.

If several files are to be restored, the list of volumes in the ASSOVOLUME parameter must correspond to the list of files in the FMOVE parameter. If no volume is to be given for a file, its entry in the volume list must be left empty. See the **Examples** .

BUFNO: Count of Buffers Per Drive

The BUFNO value allocates fixed buffers for RESTONL operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DATAVOLUME: Data Storage Extent Volume

Note: The value for DATAVOLUME must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) is to be allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADASAV retries the allocation while disregarding the DATAVOLUME value.

If DSRABN is specified, DATAVOLUME is ignored for the related file. If DATAVOLUME is not specified, the Data Storage space is allocated according to ADASAV's default allocation rules.

If several files are to be restored, the list of volumes in the DATAVOLUME parameter must correspond to the list of files in the FMOVE parameter. If no volume is to be given for a file, its entry in the volume list must be left empty. See the **Examples** .

DRIVES: Tape Drives for Parallel Restore

ADASAV is able to restore files from multiple save data set volumes in parallel to RABNs that are different from their original RABNs in the database. DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

DSRABN: Starting Data Storage RABN/RABN List

DSRABN specifies the starting Data Storage RABN for each file specified by FMOVE. DSRABN can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and DSRABN omitted, the location of the file's Data Storage is chosen by ADASAV from the free areas in Data Storage that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the DSRABN parameter must correspond to the list of files in the FMOVE parameter. If no DSRABN value is specified for a file, its entry in the RABN list must be specified as zero. See the **examples** .

DSSIZE: New Data Storage Size

DSSIZE is the new size to be allocated for Data Storage for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original Data Storage.

If DSSIZE is omitted, the original Data Storage size is used.

If several files are to be restored, the list of sizes in the DSSIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **examples**.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

If the NEWFILES parameter

- is *not* specified, all files specified in the EXCLUDE parameter must also be specified in the FMOVE parameter.
- *is* specified, all files specified in the EXCLUDE parameter must also be specified in the NEWFILES parameter. In this case, the file numbers specified in the EXCLUDE parameter refer to the new file numbers in NEWFILES, not to the old file numbers in the FMOVE parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

MAXISN: New Maximum ISN

MAXISN is the new number of ISNs to be allocated for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The value must be at least as large as the original highest allocated ISN (MAXISN).

If MAXISN is omitted, the original ISN count is used.

If several files are to be restored, the list of ISN counts in the MAXISN parameter must correspond to the list of files in the FMOVE parameter. If no ISN count is to be given for a file, its entry in the ISN count list must be specified as zero. See the **examples**.

If the database consists of several Associator extents with different device types, ERROR-171 may occur if MAXISN is specified and the nucleus allocated an additional address converter extent during the online save operation. If this happens, remove the MAXISN parameter for the file indicated in the error message and rerun RESTONL FMOVE.

MAXISN2: New Maximum Secondary ISN

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. It can only be used in conjunction with the FMOVE parameter. The secondary address converter is used to map secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

The value must be at least as large as the original highest allocated ISN (MAXISN2).

If MAXISN2 is omitted, the original ISN count is used. If the file contains no secondary address converter extents, this parameter is ignored.

If several files are to be restored, the list of ISN counts in the MAXISN2 parameter must correspond to the list of files in the FMOVE parameter. If no ISN count is to be given for a file, its entry in the ISN count list must be specified as zero.

If the database consists of several Associator extents with different device types, ERROR-171 may occur if MAXISN2 is specified and the nucleus allocated an additional address converter extent during the online save operation. If this happens, remove the MAXISN2 parameter for the file indicated in the error message and rerun RESTONL FMOVE.

NEWFILES: New File Numbers

The NEWFILES parameter specifies the new file number to be assigned to each file specified by FMOVE. The parameter is optional: if no new file number is assigned to a file, the file retains its original number. NEWFILES may not be specified for expanded files, physically coupled files, or replicated files.

If a file with a number specified by NEWFILES already exists in the database, the corresponding file will not be restored unless the OVERWRITE parameter is also specified. If the file to be overwritten is password-protected, the corresponding PASSWORD parameter must also be specified.

If several files are to be restored, the list of file numbers in the NEWFILES parameter must correspond to the list of files in the FMOVE parameter. If no new file number is to be assigned to a file, its entry in the file number list of NEWFILES must be specified as zero. See the **Examples**.

You can use NEWFILES to renumber a *base file* or *LOB file* only if both files of the *LOB file group* are restored. In this case, ADASAV assigns both files the new file numbers specified by the NEWFILES parameter and adjusts the links between the two files accordingly. However, if only one file of a *LOB file group* is restored, it cannot be assigned a new file number using the NEWFILES parameter; use the ADADBS or AOS RENUMBER function instead.

NIRABN: Starting Normal Index RABN/RABN List

NIRABN specifies the starting RABN for the normal index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and NIRABN omitted, the location of the normal index is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the NIRABN parameter must correspond to the list of files in the FMOVE parameter. If no NIRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **Examples**.

NISIZE: New Size for Normal Index

NISIZE is the new size to be allocated for the normal index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original normal index.

If NISIZE is omitted, the original normal index size is used.

If several files are to be restored, the list of sizes in the NISIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **examples**.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing File

This parameter causes an existing file to be deleted and then restored. If a file which is to be restored is already present in the database, ADASAV will skip this file unless the OVERWRITE parameter is supplied.

Note: To avoid unintentionally overwriting the database, Software AG recommends that you always specify the OVERWRITE parameter after, and not before, the FMOVE file list.

PASSWORD: Adabas Security File Password

PASSWORD specifies one password or a list of passwords if one or more files in the FILES or FMOVE file list are password-protected. This only applies to files already in the database that are to be overwritten. If the NEWFILES parameter is specified, the PASSWORD parameter must specify the passwords related to the new file numbers.

When restoring more than one password-protected file, the correct passwords must be specified as positional values corresponding to the protected file numbers' positions in the FILES or FMOVE list. Refer to the **examples** for more information about the PASSWORD parameter. When overwriting password-protected files, the Adabas nucleus must be active.

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the nucleus protection log (PLOG) used while the ADASAV SAVE operation was active (see output listing of the online SAVE function). This parameter is optional when restoring a SAVE tape created by ADASAV version 5.2 or above, or when none of the files to be restored were changed during the SAVE operation. Sequential protection (SIBA) logs from more than one nucleus session can be concatenated. ADASAV skips PLOGs with a number lower than the PLOGNUM value.

If PLOGNUM is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

Note: This is not possible when restoring from a version 5.1 SAVE data set.

SYN1|SYN4: Beginning Block Number

The block number containing the SYN1/SYN4 checkpoint at which the restore operation is to begin (refer to the output listing of the online SAVE function for the block number). When restoring a SAVE tape created by ADASAV version 5.2 or above, this parameter is optional.

If SYN1/SYN4 is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

Note: This is not possible when restoring from a version 5.1 SAVE data set.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables *cannot* be tested; only the syntax of the specified parameters can be tested.

UIRABN: Starting Upper Index RABN/RABN List

UIRABN specifies the starting RABN for the upper index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and UIRABN omitted, the location of the upper index is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the UIRABN parameter must correspond to the list of files in the FMOVE parameter. If no UIRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **examples**.

UISIZE: New Upper Index Size

UISIZE is the new size to be allocated for the upper index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original upper index.

If UISIZE is omitted, the original upper index size is used.

If several files are to be restored, the list of sizes in the UISIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **Examples**.

Examples

Example 1:

```
ADASAV RESTONL   PLOGNUM=25,SYN1=160,OVERWRITE
ADASAV              FMOVE=1,2
ADASAV              ACRABN=2100,2300
ADASAV              DSRABN=1500,2000
ADASAV              NIRABN=0,2380
ADASAV              UIRABN=2190
```

Protection log 25 is to be used. The block containing the SYN1 checkpoint is 160. Files 1 and 2 are to be deleted and restored. File 1 is to be restored using starting RABNs:

Address Converter	2100
Data Storage	1500
Normal Index	(chosen by ADASAV)
Upper Index	2190

File 2 is to be restored using starting RABNs:

Address Converter	2300
Data Storage	2000
Normal Index	2380
Upper Index	(chosen by ADASAV)

Example 2:

```
ADASAV RESTONL
PLOGNUM=4711,SYN4=99,FMOVE=3,4,5,OVERWRITE
ADASAV      PASSWORD='PWD3,,PWD5'
```

The files specified by the FMOVE file list may possibly be restored to different RABNs than they had before. Files 3 and 5 are password-protected and their passwords are PWD3 and PWD5.

Example 3:

```
ADASAV RESTONL FMOVE=11,12,13,14,OVERWRITE
ADASAV          NEWFILES=16,0,17
```

Files 11, 12, 13, and 14 are to be restored. Files 11 and 13 are to be restored as files 16 and 17, respectively. The file numbers of files 12 and 14 will not be changed because the corresponding NEWFILES parameter values are specified as zero or omitted. Files 12, 14, 16, and 17 are to be overwritten, if already present in the database.

153 RESTONL GCB: Restore Database Incremental from Online Source

- Conditions 964
- Result 965
- Syntax 965
- Optional Parameters and Subparameters 966
- Examples 968

From a database SAVE data set created while the Adabas nucleus was *active,* the RESTONL GCB function restores:

- the general control blocks (GCBs);
- Associator RABNs 3-30 of the database;
- the checkpoint file;
- the security file (if present); and
- all files specified with the FILES parameter.

Notes:

1. An interrupted RESTONL GCB operation must be reexecuted from the beginning. If the interruption occurred while RESTONL GCB was restoring the PLOG, the restore operation can be completed using the RESTPLOG function. Until successful completion or reexecution of the restore operation, the database is inaccessible.
2. If the ADASAV RESTONL GCB job control contains the DD names, symbolic names, or link names for DD/WORKnn, these data sets are reset.

Conditions

To use the RESTONL GCB function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It must have been created by an online database SAVE operation with the same version of Adabas as is used for the RESTONL and must contain the file(s) to be restored.

2 The output database must have the same physical layout (device types, extent sizes) as the original database. The Associator and Data Storage data sets must be present and must have been previously formatted. The SAVE data set to be restored may have originated from this or from a different database.

3 No Adabas nucleus may be active on the output database or on a database with the DBID of the output database.

4 The protection log (PLOG) data set containing information written by the nucleus session at the time of the SAVE operation (see output of SAVE run) must be supplied. PLOG data sets from other sessions may also be included.

5 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

6 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

Result

The result of this function is a database containing the specified files and the checkpoint and security files with the same physical status they had at the end of the ADASAV SAVE operation.

This operation is equivalent to a RESTONL (database), but excludes any files not specified in the FILES parameter.

Important: Any existing database in the target Associator and Data Storage data sets is completely overwritten and any files in that database are lost.

Syntax

```
ADASAV  RESTONL  GCB  [BUFNO = { number-of-buffers | 1 } ]
                      [CLOGDEV = {CLOG1-device-type | ADARUN-device } ]
                      [DRIVES = {count | 1 } ]
                      [EXCLUDE = file-list ]
                      [FILES =  file-list ]
                      [NEWDBID = new-database-id ]
                      [NEWDBNAME = new-database-name ]
                      [NOUSERABEND]
                      [OVERWRITE]
                      [PLOGDEV = {PLOG-device-type | ADARUN-device } ]
                      [PLOGNUM = PLOG-number  [, SYN1 = PLOG-block-number ] ]
                      [TEST]
```

Optional Parameters and Subparameters

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTONL operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

CLOGDEV: Command Log Device Type

The device type of the command log (CLOG). This parameter is required only if the device type of the CLOG is different from that specified by the ADARUN DEVICE parameter.

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored.

No files specified in the EXCLUDE parameter will exist in the restored database.

All files specified in the EXCLUDE parameter must exist on the save data set.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

FILES: Files to Be Restored

FILES specifies the file or files to be included in the database restore operation. The checkpoint and security files (if any) are always restored. If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified here. If a specified file is coupled to other files, the coupled files must also be specified.

NEWDBID: New ID for Restored Database

NEWDBID may be used to assign a different database ID to the restored database. The ID can be in the range 1-65,535; if Adabas Online System Security is installed, DBID 999 is reserved.

If NEWDBID is specified, the ADARUN DBID parameter must specify the ID of the database on the SAVE data set.

No Adabas nucleus may be active with the DBID specified on NEWDBID.

NEWDBNAME: New Database Name

NEWDBNAME assigns a new name to the restored database. If NEWDBNAME is not specified, the restored database keeps its old name.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing Database

If the restore operation is to overwrite an existing database, the OVERWRITE parameter must be specified.

No Adabas nucleus may be active on the database to be overwritten.

PLOGDEV: Protection Log Device Type

The device type of the dual/multiple protection log (PLOG). This parameter is required only if the device type of the PLOG is different from that specified by the ADARUN DEVICE parameter.

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the nucleus protection log used while the ADASAV SAVE operation was active (see output listing of the online SAVE function). Sequential protection (SIBA) logs from more than one nucleus session can be concatenated. ADASAV skips protection logs with a number lower than the PLOGNUM value. The PLOGNUM parameter is optional.

If PLOGNUM is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

SYN1: Beginning Block Number

SYN1 specifies the protection log block number containing the SYN1 checkpoint at which the corresponding SAVE operation began (see output listing of the online SAVE function). This parameter is optional.

If SYN1 is not specified, ADASAV automatically determines the correct value from information stored in the SAVE data set.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTONL GCB
```

Restore the database GCBs saved when the nucleus was active (online). The checkpoint and security files are also restored. ADASAV determines the protection log number and SYN1 block number required for the restore operation automatically.

Example 2:

```
ADASAV RESTONL   GCB,FILES=3,4,5,OVERWRITE
ADASAV          PLOGNUM=15,SYN1=20
```

Files 3, 4 and 5 as well as the checkpoint and security files are restored. The protection log number is 15 and the block containing the SYN1 checkpoint is 20. The old database is to be overwritten.

154 RESTORE: Restore Database from Offline Source

- Conditions 970
- Result 970
- Syntax 971
- Optional Parameters 971
- Examples 973

The RESTORE function restores a database from a database SAVE data set created while the Adabas nucleus was *inactive*.

Notes:

1. An interrupted RESTORE (database) operation must be reexecuted from the beginning. Until successful completion or reexecution of the restore operation, the database is inaccessible.
2. If the ADASAV RESTORE (database) job control contains the DD names, symbolic names, or link names for DDWORKnn/ WORKnn, these data sets are reset.

Conditions

To use the RESTORE (database) function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It must have been created by an offline database SAVE operation with the same version of Adabas as is used for the RESTORE.

2 The output database must have the same physical layout (device types, extent sizes) as the original database. The Associator and Data Storage data sets must be present and must have been previously formatted. The SAVE data set to be restored may have originated for this or from a different database.

3 No Adabas nucleus may be active on the output database or on a database with the DBID of the output database.

4 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

Result

The result of this function is a database with the same physical status it had at the time of the ADASAV SAVE operation.

Syntax

```
ADASAV  RESTORE  [BUFNO = { number-of-buffers  |  1 } ]
                 [CLOGDEV = { CLOG1-device-type  |  ADARUN-device } ]
                 [DRIVES = { count  |  1 } ]
                 [EXCLUDE = file-list ]
                 [NEWDBID = new-database-id ]
                 [NEWDBNAME = new-database-name ]
                 [NOUSERABEND]
                 [OVERWRITE]
                 [PLOGDEV = { PLOG-device-type  |  ADARUN-device } ]
                 [TEST]
```

Optional Parameters

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTORE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

CLOGDEV: Command Log Device Type

The device type to be assigned to the dual/multiple command log (CLOG). This parameter is required only if the device type to be used for the CLOG is different from that specified by the ADARUN DEVICE parameter.

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored. For a database restore:

- no files specified in the EXCLUDE parameter will exist in the restored database; and
- all files specified in the EXCLUDE parameter must exist on the save data set.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

NEWDBID: New ID for Restored Database

NEWDBID may be used to assign a different database ID to the restored database. The ID can be in the range 1-65,535; if Adabas Online System Security is installed, DBID 999 is reserved.

If NEWDBID is specified, the ADARUN DBID parameter must specify the ID of the database on the SAVE data set.

No Adabas nucleus may be active with the DBID specified on NEWDBID.

NEWDBNAME: New Database Name

NEWDBNAME assigns a new name to the restored database. If NEWDBNAME is not specified, the restored database keeps its old name.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing Database

If the restore operation is to overwrite an existing database, the OVERWRITE parameter must be specified.

No Adabas nucleus may be active on the database to be overwritten.

PLOGDEV: Protection Log Device Type

The device type of the dual/multiple protection log (PLOG). This parameter is required only if the device type of the PLOG is different from that specified by the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTORE OVERWRITE
```

A database is to be restored. An existing database might be overwritten.

Example 2:

```
ADASAV RESTORE EXCLUDE=10,11,12
```

Files 10 through 12 are excluded from the restore of the database from an offline-save data set.

155 RESTORE FILES: Restore Files to Original RABNs from Offline Source

- Conditions 976
- Result 977
- Syntax 977
- Optional Parameters 978
- Examples 980

The RESTORE FILES function restores files from a file or database SAVE data set created while the Adabas nucleus was *inactive*, or from a file SAVE data set created with UTYPE=EXU. One or more files can be restored. The files are restored into an existing database to their original RABNs.

Notes:

1. An interrupted RESTORE FILES operation must be reexecuted from the beginning. Until successful completion or reexecution of the restore operation, the files to be restored are inaccessible.
2. Checkpoint and security files from Adabas version 5 cannot be restored.

Conditions

To use the RESTORE FILES function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It can be a database or file SAVE data set and must contain the files to be restored.

2 A file may be restored using a SAVE data set created using a different database as long as identical device types are used.

3 An existing database must be present. The files to be restored may have originated from this or from a different database. SAVE data sets from Adabas version 5.1 or above can be used.

4 All RABNs originally used by the file(s) to be restored must either be free (available according to the Free Space Table) or be occupied by files to be overwritten.

5 The Adabas nucleus may be active or inactive on the output database.

If the Adabas nucleus is active for restoring the checkpoint or security files, the ADASAV utility requires exclusive database control; that is, no user may be active on the database.

6 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

7 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

8 Expanded files and coupled files can only be restored or overwritten as a whole. That is, if one file in an expanded file is specified, all other files in the expanded file must be specified. If one file in a coupled relationship is specified, all other files in that relationship must be specified.

9 A checkpoint, security, trigger, or user-defined system file can be overwritten only by another checkpoint, security, trigger, or user-defined system file, respectively. A checkpoint, security, or trigger file cannot be restored if such a file already exists in the database with a different file number.

10 New file numbers can be assigned to the files to be restored using the NEWFILES parameter.

Result

The result of this function is the specified files with the same physical status they had at the time of the ADASAV SAVE operation.

Syntax

```
ADASAV  RESTORE  FILES = file-list  [ALLOCATION = { FORCE | NOFORCE } ]
                                   [BUFNO = { number-of-buffers | 1 } ]
                                   [DRIVES = { count | 1 } ]
                                   [EXCLUDE = file-list ]
                                   [NEWFILES = file-list ]
                                   [NOUSERABEND]
                                   [OVERWRITE]
                                   [PASSWORD = 'password-list' ]
                                   [TEST]
```

The FILES file list specifies the file or files to be restored.

For an Adabas expanded file, all component files of the expanded file including the anchor file must be specified. If a specified file is coupled to other files, the coupled files must also be specified.

The file list specified need not correspond to the file list used for the corresponding SAVE function. A file list may be specified even if no file list was used for the corresponding SAVE function.

A file may also be restored using a SAVE data set created using a different database as long as identical device types are used.

Optional Parameters

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

ALLOCATION pertains to the implicit RABN specifications derived from the files on the save data set.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTORE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

If the NEWFILES parameter

- is *not* specified, all files specified in the EXCLUDE parameter must also be specified in the FILES parameter.
- *is* specified, all files specified in the EXCLUDE parameter must also be specified in the NEWFILES parameter. In this case, the file numbers specified in the EXCLUDE parameter refer to the new file numbers in NEWFILES, not to the old file numbers in the FILES parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

NEWFILES: New File Numbers

The NEWFILES parameter specifies the new file number to be assigned to each file specified by FILES. The parameter is optional: if no new file number is assigned to a file, the file retains

its original number. NEWFILES may not be specified for expanded files, physically coupled files, or replicated files.

If a file with a number specified by NEWFILES already exists in the database, the corresponding file will not be restored unless the OVERWRITE parameter is also specified. If the file to be overwritten is password-protected, the corresponding PASSWORD parameter must also be specified.

If several files are to be restored, the list of file numbers in the NEWFILES parameter must correspond to the list of files in the FILES parameter. If no new file number is to be assigned to a file, its entry in the file number list of NEWFILES must be specified as zero. See the **examples**.

You can use NEWFILES to renumber a *base file* or *LOB file* only if both files of the *LOB file group* are restored. In this case, ADASAV assigns both files the new file numbers specified by the NEWFILES parameter and adjusts the links between the two files accordingly. However, if only one file of a *LOB file group* is restored, it cannot be assigned a new file number using the NEWFILES parameter; use the ADADBS or AOS RENUMBER function instead.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing File

This parameter causes an existing file to be deleted and then restored. If a file which is to be restored is already present in the database, ADASAV will skip this file unless the OVERWRITE parameter is supplied.

Note: To avoid unintentionally overwriting the database, Software AG recommends that you always specify the OVERWRITE parameter after, and not before, the FILES file list.

PASSWORD

PASSWORD specifies one password or a list of passwords if one or more files specified in FILES are password-protected. This only applies to files already in the database which are to be overwritten. If the NEWFILES parameter is specified, the PASSWORD parameter must specify the passwords related to the new file numbers.

When restoring more than one password-protected file, the correct passwords must be specified as positional values corresponding to the protected file numbers' positions in the FILES list. Refer to the **examples** for more information about the PASSWORD parameter. When overwriting password-protected files, the Adabas nucleus must be active.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTORE   FILES=3,4,5,OVERWRITE,
ADASAV      PASSWORD='PWD3,,PWD5'
```

Files 3, 4, and 5 are to be restored. Existing files 3, 4, and 5 are to be overwritten by the restored files. Passwords PWD3 and PWD5 are provided for files 3 and 5.

Example 2:

```
ADASAV RESTORE FILES=11,12,13,14,OVERWRITE
ADASAV          NEWFILES=16,0,17
```

Files 11, 12, 13, and 14 are to be restored. Files 11 and 13 are to be restored as files 16 and 17, respectively. The file numbers of files 12 and 14 will not be changed because the corresponding NEWFILES parameter values are specified as zero or omitted. Files 12, 14, 16, and 17 are to be overwritten, if already present in the database.

156 RESTORE FMOVE: Restore Files to Any RABNs from Offline Source

- Conditions 982
- Result 983
- Syntax 983
- Optional Parameters 984
- Examples 990

The RESTORE FMOVE function restores files from a file or database SAVE data set created while the Adabas nucleus was *inactive*, or from a file SAVE data set created with UTYPE=EXU. One or more files can be restored. The files are restored into an existing database to any free space. Their extent sizes may be changed.

Notes:

1. An interrupted RESTORE FMOVE operation must be reexecuted from the beginning. Until successful completion or reexecution of the restore operation, the files to be restored are inaccessible.
2. Checkpoint and security files from Adabas version 5 cannot be restored.

Conditions

To use the RESTORE FMOVE function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It can be a database or file SAVE data set and must contain the files to be restored. SAVE data sets from Adabas version 5.1 or above can be used.

2 An existing database must be present. The files to be restored may have originated from this or from a different database.

3 The FMOVE file list specifies a file or files to be restored using new RABNs (and sizes). The RABNs must be located on the same device type as used originally for the respective files. Files can be restored into other than the original database as long as device types are identical.

4 For the file(s) to be restored, sufficient space, either free space (according to the free space table) or space occupied by files to be overwritten, must be available in the database.

5 The Adabas nucleus may be active or inactive on the output database.

If the Adabas nucleus is active for restoring the checkpoint or security files, the ADASAV utility requires exclusive database control; that is, no user may be active on the database.

6 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

7 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

8 Expanded files and coupled files can only be restored or overwritten as a whole. That is, if one file in an expanded file is specified, all other files in the expanded file must be specified.

If one file in a coupled relationship is specified, all other files in that relationship must be specified.

9 A checkpoint, security, trigger, or user-defined system file can be overwritten only by another checkpoint, security, trigger, or user-defined system file, respectively. A checkpoint, security, or trigger file cannot be restored if such a file already exists in the database with a different file number.

10 New file numbers can be assigned to the files to be restored using the NEWFILES parameter.

Result

The result of this function is the specified files with the same contents they had at the time of the ADASAV SAVE operation but not necessarily in the same database blocks.

Syntax

The FMOVE file list specifies a file or files to be restored using new RABNs (and sizes).

For an Adabas expanded file, all component files of the expanded file including the anchor file must be specified. If a specified file is coupled to other files, the coupled files must also be specified.

The RABNs must be located on the same device type as used originally for the respective files. Files can be restored into other than the original database as long as device types are identical.

```
ADASAV  RESTORE  FMOVE = file-list  [ACRABN = AC-start-rabn-list ]
                                   [AC2RABN = AC2-start-rabn-list ]
                                   [ALLOCATION = { FORCE | NOFORCE } ]
                                   [ASSOVOLUME = 'Associator-extent-volume' ]
                                   [BUFNO = { number-of-buffers | 1 } ]
                                   [DATAVOLUME = 'Data-Storage-extent-volume' ]
                                   [DRIVES = { count | 1 } ]
                                   [DSRABN = DS-start-rabn-list ]
                                   [DSSIZE = DS-size-list ]
                                   [EXCLUDE = file-list ]
                                   [MAXISN = isn-count-list ]
                                   [MAXISN2 = isn-count-list ]
                                   [NEWFILES = file-list ]
                                   [NIRABN = NI-start-rabn-list ]
                                   [NISIZE = NI-size-list ]
                                   [NOUSERABEND]
                                   [OVERWRITE]
                                   [PASSWORD = 'password-list' ]
                                   [TEST]
                                   [UIRABN = UI-start-rabn-list ]
                                   [UISIZE = UI-size-list ]
```

Optional Parameters

ACRABN: Starting Address Converter RABN/RABN List

ACRABN specifies the starting address converter RABN for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and ACRABN omitted, the location of the address converter is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the ACRABN parameter must correspond to the list of files in the FMOVE parameter. If no ACRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **examples** .

AC2RABN: Starting Secondary Address Converter RABN/RABN List

AC2RABN specifies the starting secondary address converter RABN for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and AC2RABN omitted, the location of the secondary address converter is chosen by ADASAV from the free areas in the Associator that have the same device type as

used originally. If the file contains no secondary address converter extents, this parameter is ignored.

If several files are to be restored, the list of RABNs in the AC2RABN parameter must correspond to the list of files in the FMOVE parameter. If no AC2RABN value is to be given for a file, its entry in the RABN list must be specified as zero.

ALLOCATION: Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

ASSOVOLUME: Associator Extent Volume

Note: The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME identifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated. If the requested number of blocks cannot be found on the specified volume, ADASAV retries the allocation while disregarding the ASSOVOLUME parameter.

If ACRABN, UIRABN, or NIRABN is specified, ADASAV ignores the ASSOVOLUME value when allocating the corresponding extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADASAV's default allocation rules.

If several files are to be restored, the list of volumes in the ASSOVOLUME parameter must correspond to the list of files in the FMOVE parameter. If no volume is to be given for a file, its entry in the volume list must be left empty. See the **examples** .

BUFNO: Count of Buffers

The BUFNO value allocates fixed buffers for RESTORE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DATAVOLUME: Data Storage Extent Volume

Note: The value for DATAVOLUME must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) is to be allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADASAV retries the allocation while disregarding the DATAVOLUME value.

If DSRABN is specified, DATAVOLUME is ignored for the related file. If DATAVOLUME is not specified, the Data Storage space is allocated according to ADASAV's default allocation rules.

If several files are to be restored, the list of volumes in the DATAVOLUME parameter must correspond to the list of files in the FMOVE parameter. If no volume is to be given for a file, its entry in the volume list must be left empty. See the **examples**.

DRIVES: Tape Drives for Parallel Restore

ADASAV is able to restore files from multiple save data set volumes in parallel to RABNs that are different from their original RABNs in the database. DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

DSRABN: Starting Data Storage RABN/RABN List

DSRABN specifies the starting Data Storage RABN for each file specified by FMOVE. DSRABN can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and DSRABN omitted, the location of the file's Data Storage is chosen by ADASAV from the free areas in Data Storage that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the DSRABN parameter must correspond to the list of files in the FMOVE parameter. If no DSRABN value is specified for a file, its entry in the RABN list must be specified as zero. See the **examples**.

DSSIZE: New Data Storage Size

DSSIZE is the new size to be allocated for Data Storage for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original Data Storage.

If DSSIZE is omitted, the original Data Storage size is used.

If several files are to be restored, the list of sizes in the DSSIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **examples**.

EXCLUDE: Exclude Specified Files from Restore

If specified, EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored. A file number may be listed only once.

If not specified, no files are excluded.

If the NEWFILES parameter

- is *not* specified, all files specified in the EXCLUDE parameter must also be specified in the FMOVE parameter.

- *is* specified, all files specified in the EXCLUDE parameter must also be specified in the NEWFILES parameter. In this case, the file numbers specified in the EXCLUDE parameter refer to the new file numbers in NEWFILES, not to the old file numbers in the FMOVE parameter.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

MAXISN: New Maximum ISN

MAXISN is the new number of ISNs to be allocated for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter. The value must be at least as large as the original highest allocated ISN (MAXISN).

If MAXISN is omitted, the original ISN count is used.

If several files are to be restored, the list of ISN counts in the MAXISN parameter must correspond to the list of files in the FMOVE parameter. If no ISN count is to be given for a file, its entry in the ISN count list must be specified as zero. See the **examples**.

MAXISN2: New Maximum Secondary ISN

MAXISN specifies the desired size of the secondary address converter (AC2) in ISNs. It can only be used in conjunction with the FMOVE parameter. The secondary address converter is used to map the secondary ISNs of secondary spanned records to the RABNs of the Data Storage blocks where the secondary records are stored.

The value must be at least as large as the original highest allocated ISN (MAXISN2).

If MAXISN2 is omitted, the original ISN count is used. If the file contains no secondary address converter extents, this parameter is ignored.

If several files are to be restored, the list of ISN counts in the MAXISN2 parameter must correspond to the list of files in the FMOVE parameter. If no ISN count is to be given for a file, its entry in the ISN count list must be specified as zero.

If the database consists of several Associator extents with different device types, ERROR-171 may occur if MAXISN2 is specified and the nucleus allocated an additional address converter extent during the online save operations. If this happens remove the MAXISN2 parameter for the file indicated in the error message and rerun RESTONL FMOVE.

NEWFILES: New File Numbers

The NEWFILES parameter specifies the new file number to be assigned to each file specified by FMOVE. The parameter is optional: if no new file number is assigned to a file, the file retains its original number. NEWFILES may not be specified for expanded files, physically coupled files, or replicated files.

If a file with a number specified by NEWFILES already exists in the database, the corresponding file will not be restored unless the OVERWRITE parameter is also specified. If the file to be overwritten is password-protected, the corresponding PASSWORD parameter must also be specified.

If several files are to be restored, the list of file numbers in the NEWFILES parameter must correspond to the list of files in the FMOVE parameter. If no new file number is to be assigned to a file, its entry in the file number list of NEWFILES must be specified as zero. See the **examples** .

You can use NEWFILES to renumber a *base file* or *LOB file* only if both files of the *LOB file group* are restored. In this case, ADASAV assigns both files the new file numbers specified by the NEWFILES parameter and adjusts the links between the two files accordingly. However, if only one file of a *LOB file group* is restored, it cannot be assigned a new file number using the NEWFILES parameter; use the ADADBS or AOS RENUMBER function instead.

NIRABN: Starting Normal Index RABN/RABN List

NIRABN specifies the starting RABN for the normal index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and NIRABN omitted, the location of the normal index is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the NIRABN parameter must correspond to the list of files in the FMOVE parameter. If no NIRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **examples** .

NISIZE: New Size for Normal Index

NISIZE is the new size to be allocated for the normal index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original normal index.

If NISIZE is omitted, the original normal index size is used.

If several files are to be restored, the list of sizes in the NISIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **examples** .

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing File

This parameter causes an existing file to be deleted and then restored. If a file which is to be restored is already present in the database, ADASAV will skip this file unless the OVERWRITE parameter is supplied.

Note: To avoid unintentionally overwriting the database, Software AG recommends that you always specify the OVERWRITE parameter after, and not before, the FMOVE file list.

PASSWORD: Adabas Security File Password

PASSWORD specifies one password or a list of passwords if one or more files in the FILES or FMOVE file list are password-protected. This only applies to files already in the database that are to be overwritten. If the NEWFILES parameter is specified, the PASSWORD parameter must specify the passwords related to the new file numbers.

When restoring more than one password-protected file, the correct passwords must be specified as positional values corresponding to the positions of the protected file numbers in the FILES or FMOVE list. Refer to the **examples** for more information about the PASSWORD parameter. When overwriting password-protected files, the Adabas nucleus must be active.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UIRABN: Starting Upper Index RABN/RABN List

UIRABN specifies the starting RABN for the upper index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

If FMOVE is specified and UIRABN omitted, the location of the upper index is chosen by ADASAV from the free areas in the Associator that have the same device type as used originally.

If several files are to be restored, the list of RABNs in the UIRABN parameter must correspond to the list of files in the FMOVE parameter. If no UIRABN value is to be given for a file, its entry in the RABN list must be specified as zero. See the **examples**.

UISIZE: New Upper Index Size

UISIZE is the new size to be allocated for the upper index for each file specified by FMOVE. It can only be used in conjunction with the FMOVE parameter.

The size can be specified in cylinders, or in blocks (by appending a "B" to the number). It must be at least as large as the used area of the original upper index.

If UISIZE is omitted, the original upper index size is used.

If several files are to be restored, the list of sizes in the UISIZE parameter must correspond to the list of files in the FMOVE parameter. If no size is to be given for a file, its entry in the size list must be specified as zero. See the **examples**.

Examples

Example 1:

```
ADASAV RESTORE FMOVE=4,6, ACRABN=0,3820,
MAXISN=0,2000000
```

Three tape drives are available for parallel RESTORE processing. Files 4 and 6 are to be restored with new RABNs. The space allocation for file 4 is to be done using original sizes.

The address converter for file 6 is to begin at Associator RABN 3820, and the value for the file's MAXISN is to be increased to 2,000,000.

Example 2:

```
ADASAV RESTORE FMOVE=3,4,5, OVERWRITE
ADASAV        PASSWORD='PWD3,,PWD5'
```

Files 3, 4 and 5 are to be restored. If they already exist in the database, they are overwritten. Passwords are provided for files 3 and 5 to allow them to be overwritten. All original size values are used. The files might be restored to other than the original RABNs.

Example 3:

```
ADASAV RESTORE   FMOVE=1,2
ADASAV          FMOVE=3,4
```

Restore files 1 through 4.

Example 4:

```
ADASAV RESTORE FMOVE=11,12,13,14,OVERWRITE
ADASAV          NEWFILES=16,0,17
```

Files 11, 12, 13, and 14 are to be restored. Files 11 and 13 are to be restored as files 16 and 17, respectively. The file numbers of files 12 and 14 will not be changed because the corresponding NEWFILES parameter values are specified as zero or omitted. Files 12, 14, 16, and 17 are to be overwritten, if already present in the database.

157 RESTORE GCB: Restore Database Incremental from Offline Source

- Conditions 992
- Result 993
- Syntax 993
- Optional Parameters 993
- Examples 995

From a database SAVE data set created while the Adabas nucleus was *inactive,* the RESTORE GCB function restores

- the general control blocks (GCBs);
- Associator RABNs 3-30 of the database;
- the checkpoint file;
- the security file (if present); and
- all files specified with the FILES parameter.

Notes:

1. An interrupted RESTORE GCB operation must be reexecuted from the beginning. Until successful completion or reexecution of the restore operation, the database is inaccessible.
2. If the ADASAV RESTORE GCB job control contains the DD names, symbolic names, or link names for DDWORKnn/ WORKnn, these data sets are reset.

Conditions

To use the RESTORE GCB function, the following conditions must be met:

1 The correct SAVE data set must be supplied. It must have been created by an offline database SAVE operation with the same version of Adabas as is used for the RESTORE and must contain the file(s) to be restored.

2 The output database must have the same physical layout (device types, extent sizes) as the original database. The Associator and Data Storage data sets must be present and must have been previously formatted. The SAVE data set to be restored may have originated for this or from a different database.

3 No Adabas nucleus may be active on the output database or on a database with the DBID of the output database.

4 If the SAVE operation was performed with the DRIVES parameter, the SAVE data sets created can also be restored with the DRIVES parameter. In that case, the restore operation is performed from the different SAVE data sets in parallel. Alternatively, the SAVE data sets can be concatenated to a single SAVE data set for a restore operation without the DRIVES parameter.

5 For restoring just a few files from a multivolume database SAVE data set, only those tape volumes that actually contain data of the files to be restored need to be supplied in the ADASAV job control. The job protocol of the SAVE operation as well as the corresponding SYNV checkpoints indicate the files or parts of files contained on each volume.

Result

The result of this function is a database containing the specified files and the checkpoint and security files with the same physical status they had at the time of the ADASAV SAVE operation.

This operation is equivalent to a RESTORE (database), but excludes any files not specified in the FILES parameter.

Important: Any existing database in the target Associator and Data Storage data sets is completely overwritten and any files in that database are lost.

Syntax

```
ADASAV  RESTORE  GCB  [BUFNO = { number-of-buffers | 1 } ]
                      [CLOGDEV = { CLOG1-device-type | ADARUN-device } ]
                      [DRIVES = { count | 1 } ]
                      [EXCLUDE = file-list ]
                      [FILES = file-list ]
                      [NEWDBID = new-database-id ]
                      [NEWDBNAME = new-database-name ]
                      [NOUSERABEND]
                      [OVERWRITE]
                      [PLOGDEV = { PLOG-device-type | ADARUN-device } ]
                      [TEST]
```

Optional Parameters

BUFNO: Count of Buffers Per Drive

The BUFNO value, multiplied by the DRIVES parameter value, allocates fixed buffers for RESTORE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

CLOGDEV: Command Log Device Type

The device type of the command log (CLOG). This parameter is required only if the device type of the CLOG is different from that specified by the ADARUN DEVICE parameter.

DRIVES: Tape Drives for Parallel Restore

DRIVES is the number of tape drives to be used for parallel restore processing. The number can range 1 to 8, inclusively; the default is 1.

EXCLUDE: Exclude Specified Files from Restore

EXCLUDE lists the numbers of the files to be excluded from the restore operation; that is, the files that are not to be restored.

No files specified in the EXCLUDE parameter will exist in the restored database.

All files specified in the EXCLUDE parameter must exist on the save data set.

The parameter is optional: if not specified, no files are excluded. A file number may be listed only once.

The EXCLUDE parameter is provided for use in recovery jobs built by the Adabas Recovery Aid (ADARAI).

FILES: Files to Be Restored

FILES specifies the file or files to be included in the database restore operation. If the specified file is a component file of an Adabas expanded file, all other component files of the expanded file must also be specified here. If a specified file is coupled to other files, the coupled files must also be specified. The checkpoint and security files are always restored.

NEWDBID: New ID for Restored Database

NEWDBID may be used to assign a different database ID to the restored database. The ID can be in the range 1-65,535; if Adabas Online System Security is installed, DBID 999 is reserved.

If NEWDBID is specified, the ADARUN DBID parameter must specify the ID of the database on the SAVE data set.

No Adabas nucleus may be active with the DBID specified on NEWDBID.

NEWDBNAME: New Database Name

NEWDBNAME assigns a new name to the restored database. If NEWDBNAME is not specified, the restored database keeps its old name.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE: Overwrite Existing Database

If the restore operation is to overwrite an existing database, the OVERWRITE parameter must be specified.

No Adabas nucleus may be active on the database to be overwritten.

PLOGDEV: Protection Log Device Type

The device type of the dual/multiple protection log (PLOG). This parameter is required only if the device type of the PLOG is different from that specified by the ADARUN DEVICE parameter.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Examples

Example 1:

```
ADASAV RESTORE GCB,FILES=2,4,6,8
```

The database Associator GCBs, RABNs 3-30, the checkpoint and security files, and files 2,4,6, and 8 are to be restored.

Example 2:

```
ADASAV RESTORE GCB,FILES=3,4,5,OVERWRITE
```

The Associator's GCBs and files 3, 4, and 5 are to be restored; the existing database will be overwritten.

158 RESTPLOG: Restore Protection Log Only

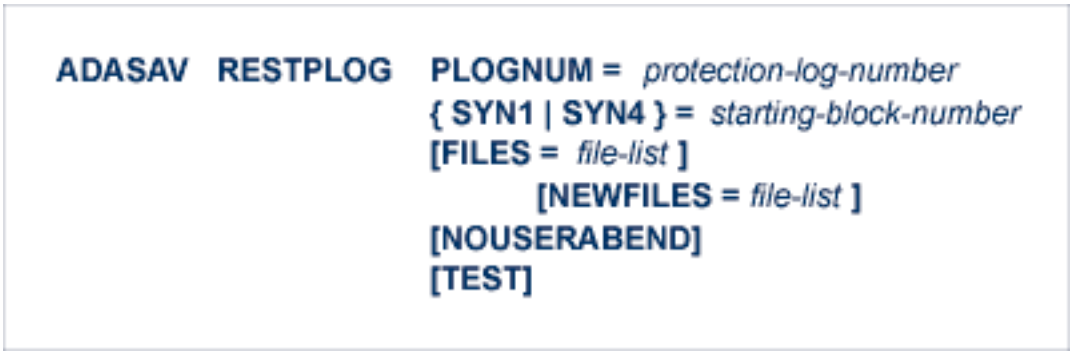
- Essential Parameters 998
- Optional Parameters 999
- Example 999

RESTPLOG restores changes contained in the PLOG to the already restored database or (if specified) files. RESTPLOG restores only the PLOG changes that were recorded during the related online SAVE (database or FILES) operation.

The RESTPLOG function is used when the following sequence occurs:

1. A SAVE data set is created online; that is, while the Adabas nucleus is active.
2. Using output created during the online SAVE, the RESTONL function is executed to restore the database or files, completes restoring the database or files from the SAVE tape, but ends due to an error condition before completing the updates recorded in PLOG.
3. The RESTPLOG function is executed to reapply all updates to the restored database or files that were recorded in PLOG. This avoids the need for restoring the complete database or files again with RESTONL.

RESTPLOG cannot be used to complete an ADASAV RESTONL FMOVE or an ADASAV RESTONL FILES with ALLOCATION=NOFORCE operation. These operations must be restarted.



Essential Parameters

PLOGNUM: Protection Log Number

PLOGNUM specifies the number of the protection log to be restored.

SYN1|SYN4: Starting Block Number

SYN1 or SYN4 specifies the block number containing the respective SYN1 or SYN4 checkpoint at which the restore operation is to begin.

Optional Parameters

FILES: List of Files to Restore

The FILES parameter specifies the files that were being restored in the RESTONL FILES or RESTONL GCB execution that was interrupted. For RESTPLOG, the same files must be specified that were specified for the interrupted function.

The FILES parameter must be omitted if a RESTONL (database) execution was interrupted. In this case, the RESTPLOG function is performed for all files of the database.

NEWFILES: New File Numbers

The NEWFILES parameter specifies the new file numbers to be assigned to each file listed in the FILES parameter. The same new file number assignments must be specified that were specified for the interrupted RESTONL FILE function that RESTPLOG is to complete.

If NEWFILES is not specified, the files to be restored retain their original numbers.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Example

```
ADASAV RESTPLOG PLOGNUM=30,SYN1=150
```

All updates contained on protection log 30 are to be reapplied to all affected files. The block containing the SYN1 checkpoint is 150.

159 SAVE: Save Database

- Syntax 1003
- Optional Parameters 1003
- Example 1005

The ADASAV SAVE (database) function saves the contents of the database to a sequential data set. It saves all blocks that are in use in the database.

The SAVE (database) function may be executed with the Adabas nucleus active or inactive. If executed while the Adabas nucleus is

- *active*, the RESTONL function must be used to later restore the database.
- *inactive*, the RESTORE function must be used to later restore the database.

In both cases, it is possible to restore just one or a few files from the database saved on the SAVE data set.

If the Adabas nucleus is *inactive*, it cannot be started while the SAVE function is executing, and no utility (such as ADALOD, for example) that makes changes to the database being saved can be run during the save. The SAVE function cannot be executed offline if a nucleus session autorestart is pending, or if another offline utility (such as ADALOD or ADASAV) is currently running.

If the Adabas nucleus is *active* during the execution of the save operation, users have full access to the database being saved. They can perform read, find, update, insert, and delete commands. However, utilities that make changes to the database to be saved (such as ADALOD, ADAINV, or ADADBS REFRESH, for example) must not be running and cannot be started while the save function is executing. An online save operation is also not possible if the nucleus is running without protection logging.

In an online save operation, the database to be saved may be changed while ADASAV is performing the save operation. Therefore, the Adabas nucleus writes all changed blocks to the protection log as well. This protection log must be supplied for a subsequent restore operation (that is, a RESTONL function).

The start of an online database save is marked by a SYN1 checkpoint. At the end of the online save, the nucleus synchronizes all currently active transactions. This means that Adabas performs no more update commands for users at ET status but allows the other active users to continue until they reach ET status. This status is then marked by a SYN2 checkpoint. The SYN2 checkpoint thus marks a consistent state of the database where no transactions are in progress. This state is reproduced when the database or files are restored from the SAVE data set later on.

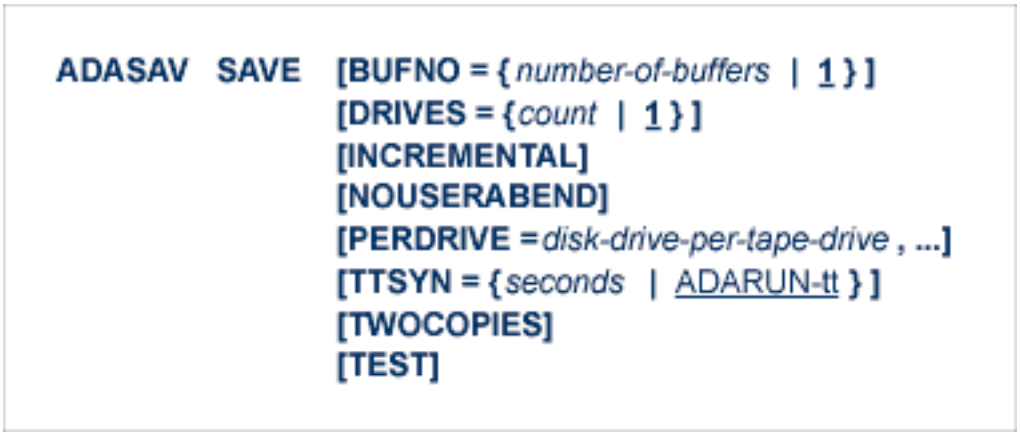
The maximum time required for the transaction synchronization can be limited by the TTSYN parameter.

Databases residing on several disk volumes are saved to several SAVE data sets in parallel when the DRIVES parameter is specified. This mode of operation may significantly reduce the duration of the save. The resulting SAVE data sets, when concatenated in the order of ascending drive number, are equivalent to a single SAVE data set produced without the DRIVES parameter.

The SAVE (database) function does not save files that are in invert, load, refresh, reorder, or restore status. In fact, it removes such files from the file list, prints message ADAU15, and performs the save operation for the remaining files. At the end, ADASAV terminates with return code 4.

If the Recovery Aid (RLOG) option is active, the SAVE (database) function starts a new RLOG generation.

Syntax



Optional Parameters

BUFNO: Count of Buffers

The BUFNO value allocates fixed buffers for the SAVE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DRIVES: Tape Drives for Parallel Save Processing

DRIVES is the number of sequential output data sets (usually on tape drives) to be used for parallel SAVE operations. A maximum of 8 drives may be specified. The default is 1.

INCREMENTAL: Save Changed Files Only

INCREMENTAL saves only those files that have been changed since the last ADASAV SAVE operation. If INCREMENTAL is not specified, the SAVE function saves all database files.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PERDRIVE: Disk Drives Per Tape Drive

PERDRIVE specifies the number of disk drives to be assigned to a single DRIVES tape drive. For example, if the database is contained on seven disk drives and three tape drives are available for SAVE processing, PERDRIVE=3,2,2 would cause the first three disk drives to be written to tape drive 1, the next two disk drives to be written to tape drive 2, and the next two disk drives to be written to tape drive 3. The drive sequence corresponds to the DD/SAVEn and DD/DUALn job control specifications, as described at the end of this document.

The total number of drives specified by PERDRIVE must equal the sum of all Associator (ASSO) and DATA disks; if both ASSO and DATA are on a single disk, this counts as two separate disks. If the DRIVES parameter is used and the PERDRIVE parameter is omitted, ADASAV determines the most efficient utilization of the tape drives.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TTSYN: SYN2 Checkpoint Control

TTSYN allows the user to decrease the ADARUN TT (maximum transaction time) of the nucleus during the synchronized checkpoint processing of the current ADASAV operation. The value specified is the approximate time in seconds (TT ⊚ 1.05 seconds), and must be less than the current ADARUN TT value. If TTSYN is not specified or if TTSYN is greater than the TT value of the nucleus, that TT value becomes the default.

If the Adabas nucleus is active while ADASAV SAVE is running, a synchronized SYN2 checkpoint is taken at the end of the SAVE operation. This ensures that there is a point in time where all users are at ET status. If a user is not at ET status, no new transactions can be started for other users; they must wait until the SYN2 checkpoint can be taken.

The ADARUN TT value controls the maximum elapsed time permitted for a logical transaction. This is the maximum wait time until the SYN2 checkpoint can be processed. The ADASAV SAVE TTSYN parameter allows the user to decrease the TT value only during the synchronized checkpoint processing. The original TT value becomes effective again when ADASAV ends the SAVE operation.

TWOCOPIES: Create Two Copies of Output

TWOCOPIES creates two physical copies of the ADASAV output.

Example

```
ADASAV SAVE DRIVES=4
```

The SAVE function is to be executed using four tape drives in parallel.

160 SAVE FILES: Save Specified Files

- Syntax 1009
- Optional Parameters 1010
- Examples 1011

The ADASAV SAVE FILES function saves the contents of one or more files to a sequential data set. It saves all blocks that are in use in the file(s).

The SAVE FILES function may be executed with the Adabas nucleus active or inactive. If executed while the Adabas nucleus is

- *active*, the RESTONL function must be used to later restore the file(s).
- *inactive*, the RESTORE function must be used to later restore the file(s).

In both cases, it is possible to restore just one or a few files from all files saved on the SAVE data set.

If the Adabas nucleus is *inactive*, it cannot be started while the SAVE function is executing, and no utility (such as ADALOD, for example) that makes changes to the file(s) being saved can be run during the save. The SAVE function cannot be executed offline if a nucleus session autorestart is pending, or if another offline utility (such as ADALOD or ADASAV) is currently running on the file(s) to be saved.

If the Adabas nucleus is *active* during the execution of the save operation, users have full access to the file(s) being saved. They can perform read, find, update, insert, and delete commands. However, utilities that make changes to the files to be saved (such as ADALOD, ADAINV, or ADADBS REFRESH, for example) must not be running and cannot be started while the save function is executing. An online save operation is also not possible if the nucleus is running without protection logging.

In an online save operation, the file(s) to be saved may be changed while ADASAV is performing the save operation. Therefore, the Adabas nucleus writes all changed blocks of the file(s) being saved to the protection log as well. This protection log must be supplied for a subsequent restore operation (that is, a RESTONL function).

The start of an online file save is marked by a SYN4 checkpoint. At the end of the online save, the nucleus synchronizes all currently active transactions. This means that Adabas performs no more update commands for users at ET status but allows the other active users to continue until they reach ET status. This status is then marked by a SYN5 checkpoint. The SYN5 checkpoint thus marks a consistent state of the database where no transactions are in progress. This state is reproduced when files are restored from the SAVE data set later on.

The maximum time required for the transaction synchronization can be limited by the TTSYN parameter.

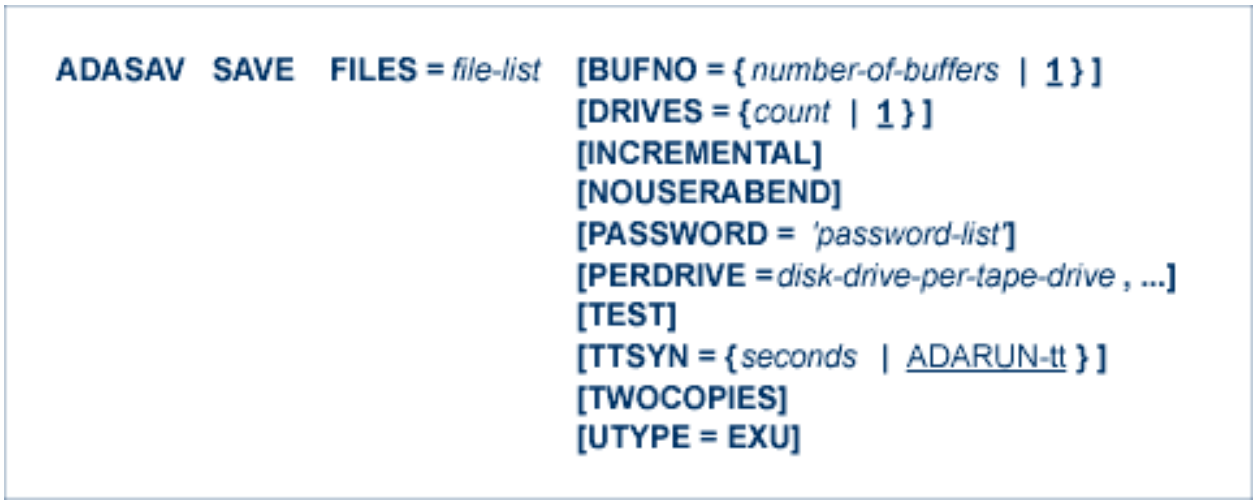
If the parameter UTYPE=EXU is specified and the Adabas nucleus is active, the save operation is performed like an offline save. ADASAV locks all files to be saved with an EXU-open against concurrent updates. The RESTORE function (rather than RESTONL) must be used for a later restore of the file(s).

Several offline file save operations, or file saves with UTYPE=EXU can be performed on different files in parallel. Only one online file save operation can be active at a time.

Files from databases residing on several disk volumes are saved to several SAVE data sets in parallel when the DRIVES parameter is specified. This mode of operation may significantly reduce the duration of the save. The resulting SAVE data sets, when concatenated in the order of ascending drive number, are equivalent to a single SAVE data set produced without the DRIVES parameter.

The SAVE FILES function does not save files that are in invert, load, refresh, reorder, or restore status. In fact, it removes such files from the file list, prints message ADAU15, and performs the save operation for the remaining files. At the end, ADASAV terminates with return code 4.

Syntax



The FILES file list specifies the file(s) to be saved. If a specified file is coupled to another file or is a component of an expanded file, and the ADASAV SAVE FILES function is executed

- with an *active* nucleus, ADASAV only accepts a SAVE FILES operation if all component files of an expanded file and all files coupled to the file are specified in the FILES file list.
- with an *inactive* nucleus, the SAVE FILES operation is accepted if any one component file of an expanded file or one of several coupled files is specified. ADASAV then extends the file list automatically.

Optional Parameters

BUFNO: Count of Buffers

The BUFNO value allocates fixed buffers for the SAVE operation. A value of 2 or 3 usually provides optimum performance; up to 255 is possible. A value greater than 5, however, provides little advantage and allocates a lot of space. The default is 1 (one buffer per drive).

DRIVES: Tape Drives for Parallel Save Processing

DRIVES is the number of tape drives to be used for parallel SAVE operation. A maximum of 8 drives can be specified. The default is 1.

INCREMENTAL: Save Changed Files Only

INCREMENTAL saves only those files in the FILES list that have been changed since the last ADASAV SAVE operation. If INCREMENTAL is not specified, the SAVE function saves all files in the FILES list.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: File Password

PASSWORD specifies one password or a list of passwords if one or more files in the FILES file list are password-protected. For more than one password-protected file, the correct passwords must be specified as positional values corresponding to the protected file numbers' positions in the FILES list. Refer to the examples at the end of this section for more information about the PASSWORD parameter. When saving password-protected files, the Adabas nucleus must be active.

PERDRIVE: Disk Drives per Tape Drive

PERDRIVE specifies the number of disk drives to be assigned to a single tape drive. For example, if the database is contained on seven disk drives, and three tape drives are available for SAVE processing, PERDRIVE=3,2,2 would cause the first three disk drives to be written to tape drive 1, the next two disk drives to be written to tape drive 2, and the next two disk drives to be written to tape drive 3. The drive sequence corresponds to the DDSAVEn/DDDUALn or SAVEn/DUALn job control specifications, as described at the end of this document.

The total number of drives specified by PERDRIVE must equal the sum of all Associator (ASSO) and DATA disks; if both ASSO and DATA are one a single disk, this counts as two separate disks. If the DRIVES parameter is used and the PERDRIVE parameter is omitted, ADASAV will determine the most efficient utilization of the tape drives.

TEST: Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

TTSYN: SYN5 Checkpoint Control

TTSYN allows the user to decrease the ADARUN TT (maximum transaction time) of the nucleus during the synchronized checkpoint processing of the current ADASAV operation. The value specified is the approximate time in seconds (TT ⊚ 1.05 seconds), and must be less than the current ADARUN TT value. If TTSYN is not specified or if TTSYN is greater than the TT value of the nucleus, the nucleus' TT value becomes the default.

Note: TTSYN is ineffective if UTYPE=EXU.

If the Adabas nucleus is active while ADASAV SAVE is running, a synchronized SYN5 checkpoint is taken at the end of the SAVE operation. This ensures that there is a point in time where all users are at ET status. If a user is not at ET status, no new transactions can be started for other users; they must wait until the SYN5 checkpoint can be taken.

The ADARUN TT value controls the maximum elapsed time permitted for a logical transaction. This is the maximum wait time until the SYN5 checkpoint can be processed. The ADASAV SAVE TTSYN parameter allows the user to decrease the TT value only during the synchronized checkpoint processing. The original TT value becomes effective again when ADASAV ends the SAVE operation.

TWOCOPIES: Create Two Copies of Output

TWOCOPIES creates two physical copies of the ADASAV output.

UTYPE=EXU: User Type for Open

ADASAV issues an Adabas open command with a record buffer "EXU=file-list". This enables a file to be saved where an Adabas nucleus is active with no protection log. No updates to the files being saved are permitted while the SAVE function is operating. The corresponding RESTORE file operation does not require a protection log.

Examples

Example 1:

```
ADASAV SAVE FILES=10,15
```

Files 10 and 15 are to be saved.

Example 2:

```
ADASAV SAVE FILES=3,4,5,
ADASAV      PASSWORD='PWD3,,PWD5'
```

Save files 3, 4, and 5. Files 3 and 5 are password protected and their passwords are PWD3 and PWD5.

161 JCL/JCS Requirements and Examples

- BS2000 1014
- z/OS 1020
- z/VM 1025
- VSE 1029

This section describes the job control information required to run ADASAV with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSOR*n*	disk	
Data Storage	DDDATAR*n*	disk	
Work	DDWORKR1 DDWORKR*n* (1-9) to DDWORK*nn* (10-32) in cluster environments	disk	A Work data set is *not* required for: ■ any ADASAV SAVE functions ■ ADASAV RESTONL FMOVE and ADASAV RESTONL FILES functions ■ ADASAV RESTORE FMOVE and ADASAV RESTORE FILES functions ■ ADASAV RESTPLOG functions A Work data set is *recommended* for: ■ ADASAV RESTONL and ADASAV RESTONL GCB functions ■ ADASAV RESTORE and ADASAV RESTORE GCB functions In these cases, if a Work data set is specified, the ADASAV function resets it; if a Work data set is not specified, no Work data sets are reset. **Note:** In an Adabas Cluster Services or Adabas Parallel Services cluster, where each cluster has its own Work data set, the Work data sets of all nuclei in the cluster should be specified for the RESTONL/RESTORE or RESTONL GCB/RESTORE GCB functions. If a database is restored and the Work data set has not been reset, nucleus error 72 (PARM ERROR 72) may occur. To resolve this, you can reset any Work data sets using the ADAFRM utility.
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
Backup copy	DDSAVE1-8	tape/ disk	Required for SAVE
Dual copy	DDDUAL1-8	tape/ disk	Required for SAVE with two backup copies

Data Set	Link Name	Storage	More Information
Backup copy (input to the RESTORE function)	DDREST1-8	tape/ disk	Required for RESTORE and RESTONL
Sequential protection log	DDPLOG	tape/ disk	Required for RESTONL and RESTPLOG
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADASAV parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADASAV messages	SYSLST/ DDDRUCK		*Messages and Codes*

Note: For RESTONL, the input SAVE tapes and the sequential protection log can be concatenated, using the name DDREST1.

ADASAV JCL Examples (BS2000)

Save Files, Save Database

In SDF Format:

```
/.ADASAV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A S A V SAVE FILES, SAVE DATABASE
 /REMARK *
 /DELETE-FILE ADAyyyyy.SAVE1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.SAVE1,SUP=TAPE(DEVICE=TAPE-C1,VOL=SAV101),-
 /  PROT=(USER-ACCESS=ALL-USERS)
 /SET-JOB-STEP
 /DELETE-FILE ADAyyyyy.DUAL1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.DUAL1,SUP=TAPE(DEVICE=TAPE-C1,VOL=SAV101),-
 /  PROT=(USER-ACCESS=ALL-USERS)
 /SET-JOB-STEP
 /ASS-SYSLST L.SAV.SAVE
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDSAVE1,ADAyyyyy.SAVE1,TAPE(FILE-SEQ=1)
 /SET-FILE-LINK DDDUAL1,ADAyyyyy.DUAL1,TAPE(FILE-SEQ=1)
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
```

```
 ADASAV SAVE TWOCOPIES
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASAV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A S A V SAVE FILES, SAVE DATABASE
 /REMARK *

/SYSFILE SYSLST=L.SAV.SAVE
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.SAVE1 ,LINK=DDSAVE1 ,DEVICE=TAPE-C1,VOLUME=SAV101
 /FILE ADAyyyyy.DUAL1 ,LINK=DDDUAL1 ,DEVICE=TAPE-C1,VOLUME=SAV201
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV SAVE TWOCOPIES
 /LOGOFF NOSPOOL
```

Restore Files from SAVE Data Sets Created Online

In SDF Format:

```
/.ADASAV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE FILES, RESTORE DATABASE
 /REMARK * FROM ONLINE CREATED SAVE DATASETS
 /REMARK *
 /ASS-SYSLST L.SAV.REON
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDREST1,ADAyyyyy.SAVE1
 /SET-FILE-LINK DDPLOG,ADAyyyyy.PLOG
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTONL FILES=2,PLOGNUM=ppp,SYN1=43
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASAV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE FILES, RESTORE DATABASE
 /REMARK * FROM ONLINE CREATED SAVE DATASETS
 /REMARK *

/SYSFILE SYSLST=L.SAV.REON
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.SAVE1 ,LINK=DDREST1
 /FILE ADAyyyyy.PLOG ,LINK=DDPLOG
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTONL FILES=2,PLOGNUM=ppp,SYN1=43
 /LOGOFF NOSPOOL
```

Restore Database

In SDF Format:

```
/.ADASAV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * ADASAV:
 /REMARK * EXAMPLE HOW TO USE ADASAV TO RESTORE THE
 /REMARK * ENTIRE DATABASE /REMARK *
 /REMARK *
 /DELETE-FILE ADAyyyyy.SAVE1
 /SET-JOB-STEP
 /IMPORT-FILE
SUP=TAPE(F-NAME=ADAyyyyy.SAVE1,DEV-TYPE=TAPE-C1,VOL=SAV101)
 /SET-JOB-STEP
 /ASS-SYSLST L.SAV.REST
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
 /SET-FILE-LINK DDREST1,ADAyyyyy.SAVE1,TAPE(FILE-SEQ=1),ACC-METH=SAM,-
 /  BUFF-LEN=32768,REC-FORM=V
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTORE OVERWRITE
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASAV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * ADASAV:
 /REMARK * EXAMPLE HOW TO USE ADASAV TO RESTORE THE
 /REMARK * ENTIRE DATABASE
 /REMARK *
 /SYSFILE SYSLST=L.SAV.REST
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
 /FILE ADAyyyyy.SAVE1 ,LINK=DDREST1,DEVICE=TAPE-C1,VOLUME=ADA001,-
 /     STATE=FOREIGN.-
 /     FCBTYPE=SAM,RECFORM=V,RECSIZE=,BLKSIZE=32768,LABEL=STD
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTORE OVERWRITE
 /LOGOFF NOSPOOL
```

Restore Protection Log after an Interrupted RESTONL Function

In SDF Format:

```
/.ADASAV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE PROTECTION LOG
 /REMARK *
 /ASS-SYSLST L.SAV.REPL
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDPLOG,ADAyyyyy.PLOG
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTPLOG FILES=2,PLOGNUM=ppp,SYN1=43
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASAV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE PROTECTION LOG
 /REMARK *
 /SYSFILE SYSLST=L.SAV.REPL
 /FILE ADA.MOD,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.PLOG ,LINK=DDPLOG
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTPLOG FILES=2,PLOGNUM=ppp,SYN1=43
 /LOGOFF NOSPOOL
```

Restore Files from SAVE Data Sets Created Offline

In SDF Format:

```
/.ADASAV LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE FILES, RESTORE DATABASE
 /REMARK * FROM OFFLINE CREATED SAVE DATASETS
 /REMARK *
 /ASS-SYSLST L.SAV.REFM
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
 /SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
 /SET-FILE-LINK DDREST1,ADAyyyyy.SAVE1
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTORE FMOVE=2
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASAV LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A S A V RESTORE FILES, RESTORE DATABASE
 /REMARK * FROM OFFLINE CREATED SAVE DATASETS
 /REMARK *
 /SYSFILE SYSLST=L.SAV.REFM
 /FILE ADA.MOD ,LINK=DDLIB
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
 /FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
 /FILE ADAyyyyy.SAVE1 ,LINK=DDREST1
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADASAV,DB=yyyyy,IDTNAME=ADABAS5B
 ADASAV RESTORE FMOVE=2
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSOR*n*	disk	
Data Storage	DDDATAR*n*	disk	
Work	DDWORKR1 DDWORKR*n* (1-9) to DDWORK*nn* (10-32) in cluster environments	disk	A Work data set is *not* required for: ■ any ADASAV SAVE functions ■ ADASAV RESTONL FMOVE and ADASAV RESTONL FILES functions ■ ADASAV RESTORE FMOVE and ADASAV RESTORE FILES functions ■ ADASAV RESTPLOG functions A Work data set is *recommended* for: ■ ADASAV RESTONL and ADASAV RESTONL GCB functions ■ ADASAV RESTORE and ADASAV RESTORE GCB functions In these cases, if a Work data set is specified, the ADASAV function resets it; if a Work data set is not specified, no Work data sets are reset.

Data Set	DD Name	Storage	More Information
			Note: In an Adabas Cluster Services or Adabas Parallel Services cluster, where each cluster has its own Work data set, the Work data sets of all nuclei in the cluster should be specified for the RESTONL/RESTORE or RESTONL GCB/RESTORE GCB functions. If a database is restored and the Work data set has not been reset, nucleus error 72 (PARM ERROR 72) may occur. To resolve this, you can reset any Work data sets using the ADAFRM utility.
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
Backup copy	DDSAVE1-8	tape/ disk	Required for SAVE
Dual copy	DDDUAL1-8	tape/ disk	Required for SAVE with two backup copies
Backup copy (input for RESTORE function)	DDREST1-8	tape/ disk	Required for RESTORE and RESTONL
Sequential protection log	DDPLOG	tape/ disk	Required for RESTONL and RESTPLOG
ADARUN parameters	DDCARD	reader	*Operations*
ADASAV parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADASAV messages	DDDRUCK	printer	*Messages and Codes*

Note: For RESTONL, the input SAVE tapes and the sequential protection log can be concatenated, using the name DDREST1.

ADASAV JCL Examples (z/OS)

Save Database

```
//ADASAV    JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO SAVE THE
//*       ENTIRE DATABASE
//*
//SAVE     EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD          <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1    <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1    <=== DATA
//DDSAVE1   DD   DSN=EXAMPLE.DByyyyy.DDSAVE1,UNIT=TAPE, <=== OUTPUT
//            DISP=(,CATLG),VOL=SER=ADABCK
```

```
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADASAV,SVC=xxx,DE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV SAVE
/*
```

Refer to ADASAV in the JOBS data set for this example.

Save Database with Two Copies of Output

```
//ADASAVT   JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO SAVE THE
//*       ENTIRE DATABASE CREATING TWO COPIES OF THE OUTPUT
//*
//SAVE     EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD          <=== ADABAS LOAD
//*

//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1    <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1    <=== DATA
//DDSAVE1   DD   DSN=EXAMPLE.DByyyyy.DDSAVE1,UNIT=TAPE, <=== OUTPUT
//            DISP=(,CATLG),VOL=SER=ADABCK
//DDDUAL1   DD   DSN=EXAMPLE.DByyyyy.DDSAVD1,UNIT=TAPE, <=== OUTPUT
//            DISP=(,CATLG),VOL=SER=ADABCK1
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADASAV,SVC=xxx,DE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV SAVE TWOCOPIES
/*
```

Refer to ADASAVT in the JOBS data set for this example.

Restore Database

```
//ADASAVR   JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO RESTORE THE
//*       ENTIRE DATABASE
//*
//RESTORE  EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDREST1   DD   DSN=EXAMPLE.DByyyyy.SAVE,           <=== SAVE OUTPUT
//          DISP=OLD,UNIT=TAPE,VOL=SER=ADABCK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X

//DDCARD    DD   *
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV RESTORE OVERWRITE
/*
```

Refer to ADASAVR in the JOBS data set for this example.

Restore Files From SAVE Data Sets Created Offline

```
//ADASAVRF  JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO RESTORE A FILE
//*       TO ANY RABNS FROM AN OFFLINE SAVE
//*
//RESTORE  EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABASLOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDREST1   DD   DSN=EXAMPLE.DByyyyy.SAVE,           <=== SAVE OUTPUT
//          DISP=OLD,UNIT=TAPE,VOL=SER=ADABCK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
```

```
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV RESTORE FMOVE=2
/*
```

Refer to ADASAVRF in the JOBS data set for this example.

Restore Files From SAVE Data Sets Created Online

```
//ADASAVRO  JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO RESTORE FILES
//*       FROM SAVE DATA SETS CREATED ONLINE
//*
//RESTORE  EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDREST1   DD   DSN=EXAMPLE.DByyyyy.SAVE,           <=== SAVE OUTPUT
//          DISP=OLD,UNIT=TAPE,VOL=SER=ADABCK
//DDPLOG    DD   DSN=EXAMPLE.DByyyyy.PLOG,           <=== PLOG OUTPUT
//          DISP=OLD,UNIT=TAPE,VOL=SER=PLOGD1
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV RESTONL FILES=2,PLOGNUM=ppp,SYN1=1
/*
```

Refer to ADASAVRO in the JOBS data set for this example.

Restore Protection Log after an Interrupted RESTONL Function

```
//ADASAVRP  JOB
//*
//*    ADASAV:
//*       EXAMPLE HOW TO USE ADASAV TO RESTORE THE
//*       PROTECTION LOG AFTER AN INTERRUPTED RESTONL
//*
//RESTORE  EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
```

```
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDPLOG    DD   DSN=EXAMPLE.DByyyyy.PLOG,           <=== PLOG INPUT
//          DISP=OLD,UNIT=TAPE,VOL=SER=PLOGD1
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADASAV RESTPLOG PLOGNUM=ppp,SYN1=1
/*
```

Refer to ADASAVRP in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Associator	DDASSOR*n*	disk	
Data Storage	DDDATAR*n*	disk	
Work	DDWORKR1 DDWORKR*n* (1-9) to DDWORK*nn* (10-32) in cluster environments	disk	A Work data set is *not* required for: ■ any ADASAV SAVE functions ■ ADASAV RESTONL FMOVE and ADASAV RESTONL FILES functions ■ ADASAV RESTORE FMOVE and ADASAV RESTORE FILES functions ■ ADASAV RESTPLOG functions A Work data set is *recommended* for: ■ ADASAV RESTONL and ADASAV RESTONL GCB functions ■ ADASAV RESTORE and ADASAV RESTORE GCB functions In these cases, if a Work data set is specified, the ADASAV function resets it; if a Work data set is not specified, no Work data sets are reset.

Data Set	DD Name	Storage	More Information
			Note: In an Adabas Cluster Services or Adabas Parallel Services cluster, where each cluster has its own Work data set, the Work data sets of all nuclei in the cluster should be specified for the RESTONL/RESTORE or RESTONL GCB/RESTORE GCB functions. If a database is restored and the Work data set has not been reset, nucleus error 72 (PARM ERROR 72) may occur. To resolve this, you can reset any Work data sets using the ADAFRM utility.
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
Backup copy	DDSAVE1-8	tape/ disk	Required for SAVE
Dual copy	DDDUAL1-8	tape/ disk	Required for SAVE with two backup copies
Backup copy (input to RESTORE function)	DDREST1-8	tape/ disk	Required for RESTORE and RESTONL
Sequential protection log	DDPLOG	tape/ disk	Required for RESTONL and RESTPLOG
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADASAV parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADASAV messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

Note: For RESTONL, the input SAVE tapes and the sequential protection log can be concatenated, using the name DDREST1.

ADASAV Job Control Examples (z/VM)

Save Database

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDSAVE1,DSN=ADABASVv.COPY1,MODE=A
DATADEF DDDUAL1,DSN=ADABASVv.COPY2,MODE=A
DATADEF DDPRINT,DSN=ADASAV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADASAV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNSAV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADASAV.CONTROL,MODE=A
ADARUN
```

Contents of RUNSAV CONTROL A1

```
ADARUN PROG=ADASAV,DEVICE=dddd,DB=yyyyy
```

Contents of ADASAV CONTROL A1

```
ADASAV SAVE TWOCOPIES
```

Restore Files from SAVE Data Sets Created Online

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDREST1,DSN=ADABASVv.SAVE,MODE=A
DATADEF DDPLOG,DSN=ADABASVv.SIBA,MODE=A
DATADEF DDPRINT,DSN=ADASAV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADASAV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNSAV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=RESTONL.CONTROL,MODE=A
ADARUN
```

Contents of RUNSAV CONTROL A1

```
ADARUN PROG=ADASAV,DEVICE=dddd,DB=yyyyy
```

Contents of RESTONL CONTROL A1

```
ADASAV RESTONL FILES=2,PLOGNUM=ppp,SYN1=1
```

Restore Protection Log After Interrupted RESTONL Function

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPLOG,DSN=ADABASVv.PLOG,MODE=A
DATADEF DDPRINT,DSN=ADASAV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADASAV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNSAV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=RESTFILE.CONTROL,MODE=A
ADARUN
```

Contents of RUNSAV CONTROL A1

```
ADARUN PROG=ADASAV,DEVICE=dddd,DB=yyyyy
```

Contents of RESTFILE CONTROL A1

```
ADASAV RESTPLOG FILES=2,PLOGNUM=ppp,SYN1=1
```

Restore Files from SAVE Data Sets Created Offline

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDREST1,DSN=ADABASVv.SAVE,MODE=A
DATADEF DDPRINT,DSN=ADASAV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADASAV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNSAV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=RESTORE.CONTROL,MODE=A
ADARUN
```

Contents of RUNSAV CONTROL A1

```
ADARUN PROG=ADASAV,DEVICE=dddd,DB=yyyyy
```

Contents of RESTORE CONTROL A1

```
ADASAV RESTORE FMOVE=2
```

VSE

File	Symbolic Name	Storage	Logical Unit	Information
Associator	ASSORn	disk	see note 1	
Data Storage	DATARn	disk	see note 1	
Work	WORKR1 WORKR*n* (1-9) to WORK*nn* (10-32) in cluster environments	disk	see note 1	A Work data set is *not* required for: ■ any ADASAV SAVE functions ■ ADASAV RESTONL FMOVE and ADASAV RESTONL FILES functions ■ ADASAV RESTORE FMOVE and ADASAV RESTORE FILES functions ■ ADASAV RESTPLOG functions A Work data set is *recommended* for: ■ ADASAV RESTONL and ADASAV RESTONL GCB functions ■ ADASAV RESTORE and ADASAV RESTORE GCB functions In these cases, if a Work data set is specified, the ADASAV function resets it; if a Work data set is not specified, no Work data sets are reset. **Note:** In an Adabas Cluster Services or Adabas Parallel Services cluster, where each cluster has its own Work data set, the Work data sets of all nuclei in the cluster should be specified for the RESTONL/RESTORE or RESTONL GCB/RESTORE GCB functions.

File	Symbolic Name	Storage	Logical Unit	Information
				If a database is restored and the Work data set has not been reset, nucleus error 72 (PARM ERROR 72) may occur. To resolve this, you can reset any Work data sets using the ADAFRM utility.
Recovery log (RLOG)	RLOGR1	disk		Required for ADARAI
Backup copy	SAVE1-8	tape disk	SYS011-SYS018 see note 1	Required for SAVE
Dual copy	DUAL1-8	tape disk	SYS021-SYS028 see note 1	Required for SAVE with two backup copies
Backup copy (input for RESTORE)	REST1-8	tape disk	SYS011-SYS018 see note 1	Required for RESTORE and RESTONL
Sequential protection log	PLOG	tape disk	SYS010 see note 1	Required for RESTONL and RESTPLOG
ADARUN parameters	SYSRDR CARD CARD	reader tape disk	SYSRDR SYS000 See note 1	
ADASAV parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	
ADASAV messages		printer	SYS009	

Notes:

1. Any programmer logical unit may be used.
2. For RESTONL, the input SAVE tapes and the sequential protection log can be concatenated, using the name REST1.

ADASAV JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Refer to the following members for these examples:

Example	Member
Save database	ADASAV.X
Save database with two copies of output	ADASAVT.X
Restore database	ADASAVR.X
Restore files from save data sets created online	ADASAVRO.X
Restore protection log after an interrupted RESTONL function	ADASAVRP.X
Restore files from save data sets created offline	ADASAVRF.X

Save Database

```
* $$ JOB JNM=ADASAV,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAV
*       SAVE THE ENTIRE DATABASE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS011,TAPE
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cuu
// TLBL SAVE1,'EXAMPLE.DByyyyy.SAVE'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV SAVE
/*
/&
* $$ EOJ
```

Save Database with Two Copies of Output

```
* $$ JOB JNM=ADASAVT,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAVT
*       SAVE THE ENTIRE DATABASE CREATING TWO COPIES OF THE OUTPUT
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// PAUSE MOUNT LOAD SAVE TAPES ON cu1 AND cu2
// ASSGN SYS011,TAPE
// TLBL SAVE1,'EXAMPLE.DByyyyy.SAVE'
// ASSGN SYS021,TAPE
// TLBL DUAL1,'EXAMPLE.DByyyyy.SAVE.COPY'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV SAVE TWOCOPIES
```

```
/*
/&
* $$ EOJ
```

Restore Database

```
* $$ JOB JNM=ADASAVR,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAVR
*       RESTORE THE ENTIRE DATABASE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS011,TAPE
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cuu
// TLBL REST1,'EXAMPLE.DByyyyy.SAVE'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV RESTORE OVERWRITE
/*
/&
* $$ EOJ
```

Restore Files from Save Data Sets Created Online

```
* $$ JOB JNM=ADASAVRO,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAVRO
*       RESTORE FILES FROM SAVE DATA SETS CREATED ONLINE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cu1 AND PLOG ON TAPE cu2
// ASSGN SYS011,TAPE
// TLBL REST1,'EXAMPLE.DByyyyy.SAVE'
// ASSGN SYS010,TAPE
// TLBL PLOG,'EXAMPLE.DByyyyy.PLOG5'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV RESTONL FILES=2,PLOGNUM=ppp,SYN1=1
/*
/&
* $$ EOJ
```

Restore Protection Log after an Interrupted RESTONL Function

```
* $$ JOB JNM=ADASAVRP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAVRP
*       RESTORE THE PROTECTION LOG AFTER AN INTERRUPTED RESTONL
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// PAUSE MOUNT LOAD PLOG FILE ON TAPE cuu
// ASSGN SYS010,TAPE
// TLBL PLOG,'EXAMPLE.DByyyyy.PLOG5'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV RESTPLOG PLOGNUM=ppp,SYN1=1
/*
/&
* $$ EOJ
```

Restore Files From Save Data Sets Created Offline

```
* $$ JOB JNM=ADASAVRF,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASAVRF
*       RESTORE A FILE TO ANY RABNS FROM AN OFFLINE SAVE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS011,TAPE
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cuu
// TLBL REST1,'EXAMPLE.DByyyyy.SAVE'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASAV,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADASAV RESTORE FMOVE=2
/*
/&
* $$ EOJ
```

XX ADASEL Utility: Select Protection Data

This chapter covers the following topics:

Functional Overview
ADASEL Syntax
Overriding ADASEL Defaults with Global Parameters
JCL/JCS Requirements and Examples

162 Functional Overview

The ADASEL utility selects information in the Adabas sequential (SIBA) or dual/multiple (PLOG) protection log. ADASEL decompresses the information and writes it to a print data set (DDDRUCK/DRUCK) or to a user-specified output data set.

The protection log contains information on all updates applied to the database during a given Adabas session. Information selected with ADASEL can be used for auditing or as input to a Natural or non-Adabas program.

You can select before-images, after-images, or both for new, updated, and deleted records. You can also select data written to the protection log with an Adabas C5 command.

Note: If the Adabas session used *dual/multiple protection logging*, use the ADARES PLCOPY function to copy the protection log before using it as input to ADASEL.

If the Adabas session used *sequential protection logging*, and if the session terminated abnormally, use the ADARES COPY function to copy the protection log before using it as input to ADASEL.

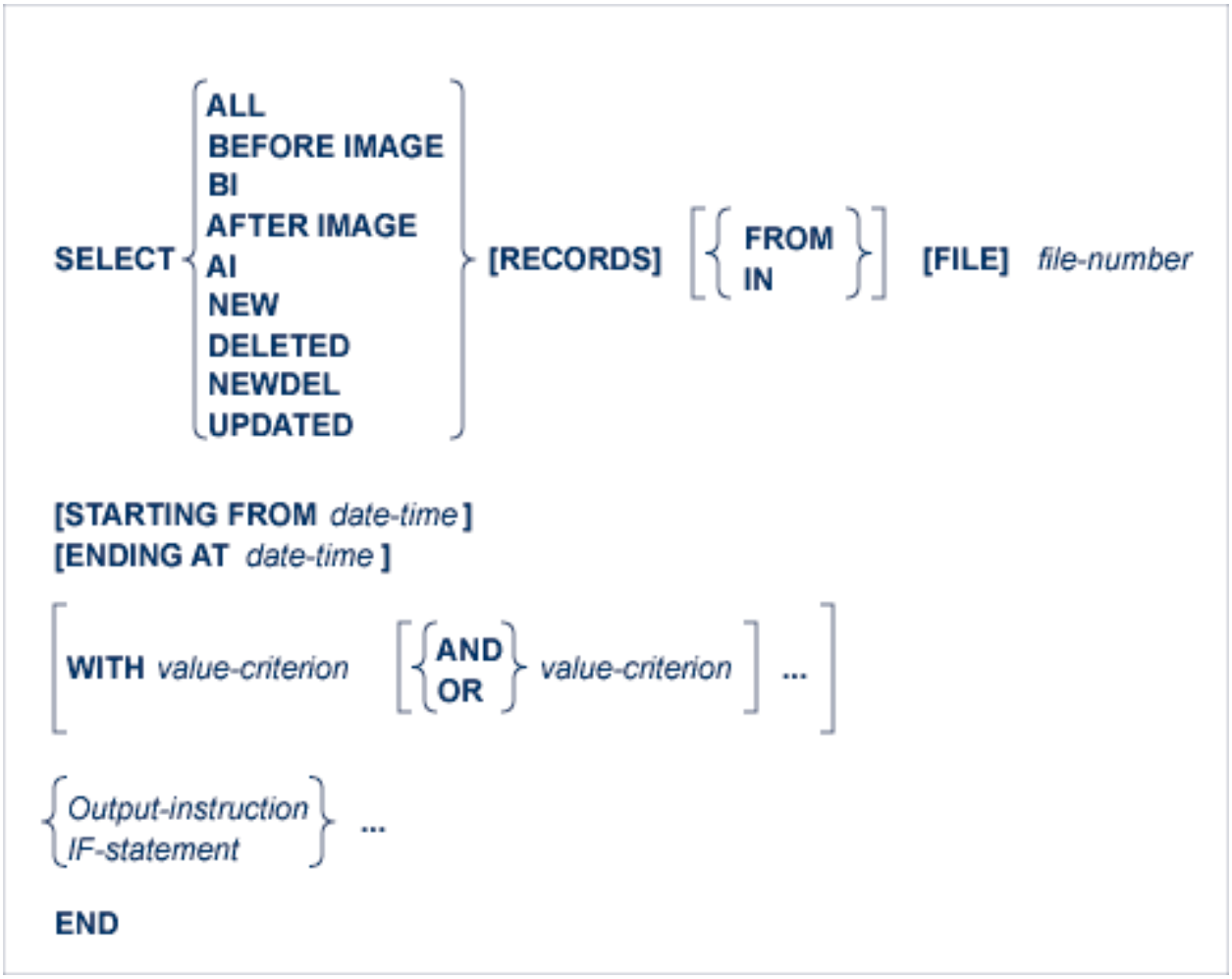
163 ADASEL Syntax

- SELECT Statement 1041
- Additional Selection Criteria 1042
- date-time 1043
- WITH Clause 1043
- IF Statement 1044
- value-criterion 1046
- output-instruction 1051

Unlike other Adabas utilities, ADASEL does not require the utility name at the beginning of each parameter line. A selection request includes the following parts:

- the keyword *SELECT*, followed by the selection option and the file number;
- optional clauses and statements that specify additional selection criteria;
- one or more output instructions;
- the keyword *END*.

An overview of the ADASEL syntax is shown below.



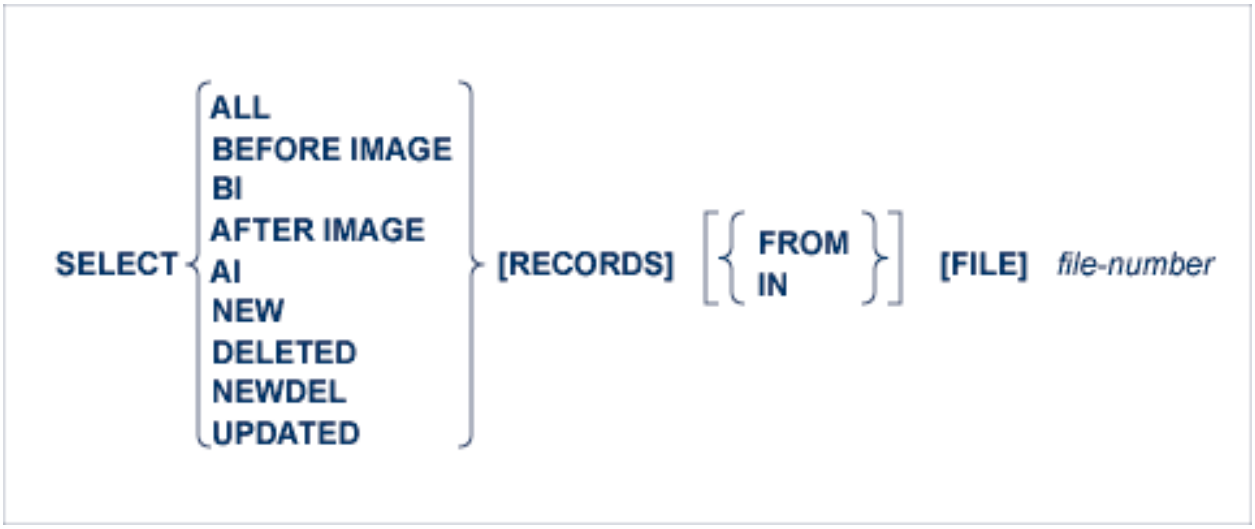
You can code multiple selection requests. Each request begins with SELECT and ends with END.

Example:

```
SELECT ALL RECORDS FROM FILE 1
   DISPLAY AA BB CC
END
```

```
SELECT BEFORE IMAGE FILE 2
   OUTPUT TO EXPA1
END
```

SELECT Statement



After the keyword SELECT, specify one of the following selection options:

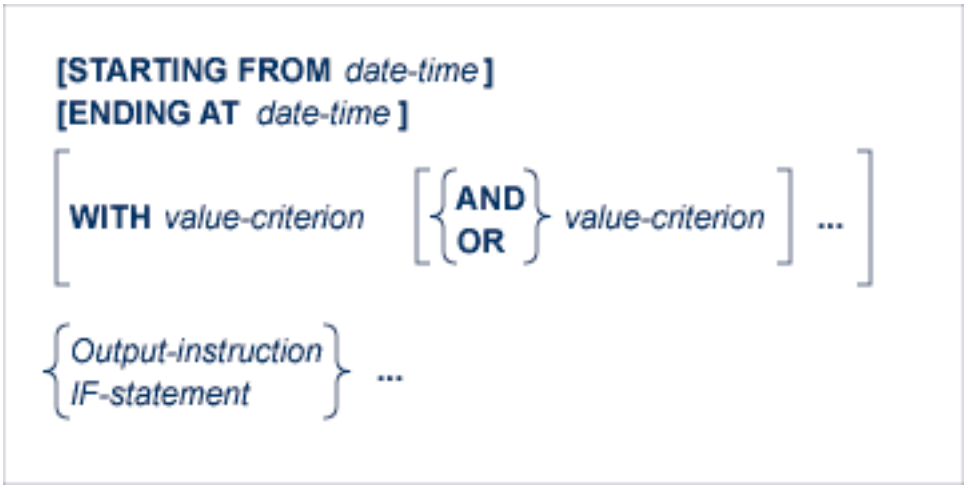
Option	Records Selected
ALL	Before-images derived from A1 (update) and E1 (delete) commands; after-images derived from A1 and N1 (add) commands.
BEFORE IMAGE | BI	Before-images derived from A1 and E1 commands.
AFTER IMAGE | AI	After-images derived from A1 and N1 commands.
NEW	After-images derived from N1 commands.
DELETED	Before-images derived from E1 commands.
NEWDEL	After-images derived from N1 commands and before-images derived from E1 commands.
UPDATED	Before-images and after-images derived from A1 commands.

file-number

Specify the Adabas file for which protection log data is to be selected. Valid file numbers are 0-5000 or 0 through one less than the ASSO block size, whichever is lower. To select user data written by a C5 command, specify the file number of the checkpoint file.

Additional Selection Criteria

You can use the STARTING FROM, ENDING AT, and WITH clauses and the IF statement to specify additional selection criteria. These optional clauses and statement take precedence over the selection option.



The STARTING FROM and ENDING AT clauses restrict selections to records added, updated, or deleted within a time range. The `date-time` variable is discussed below in section*date-time*.

The WITH clause is used to select records that satisfy the value-criteria specified. Multiple conditions can be specified using the logical operators AND and OR. The WITH clause is discussed in section *WITH Clause*.

The IF statement is used to select records and execute output instructions on a conditional basis. The IF statement is discussed in section *IF-statement*.

The syntax of the `value-criterion` variable used in both the WITH clause and the IF statement is described in section **value-criterion**.

Output instructions are described in section **output-instruction**.

date-time

The following are valid formats for the *date-time* variable:

Format	Description
yyyymmdd hhmmss	date/time
J(yyyyddd hhmmss)	Julian date/time
X ' *xxxxxxx* '	store-clock (STCK) representation

Note: The lowest valid value for *yyyy* is "1980".

Examples:

Select all records from file 1 that were added, deleted, or updated on or before midnight of May 12, 1996 (Julian date 132):

```
SELECT ALL RECORDS FROM FILE 1
   ENDING AT J(1996132/240000)
   DISPLAY AA BB CC
END
```

Select all records from file 112 that were added, deleted, or updated on or between January 1 and December 31, 1996:

```
SELECT ALL 112
   STARTING FROM 19960101/000000
   ENDING AT 19961231/240000
   OUTPUT TO EXPA1
END
```

WITH Clause

The WITH clause is used to select records that satisfy the value-criteria specified. Multiple conditions can be specified using the logical operators AND and OR.

- If value-criteria are connected by the AND operator, *each* condition must be satisfied in order for the record to be selected.
- If value-criteria are connected by the OR operator, the record is selected if *any* of the conditions is satisfied.

The syntax of the *value-criterion* variable is described in section ***value-criterion***.

Example:

The protection log contains before- and after-images for two updated records. The contents of the field BB in the records are shown below:

Before-Image	After-Image
BB = SMITH	BB = ZINN
BB = SMITH	BB = JONES

The SELECT statement includes a WITH clause that further qualifies the selection:

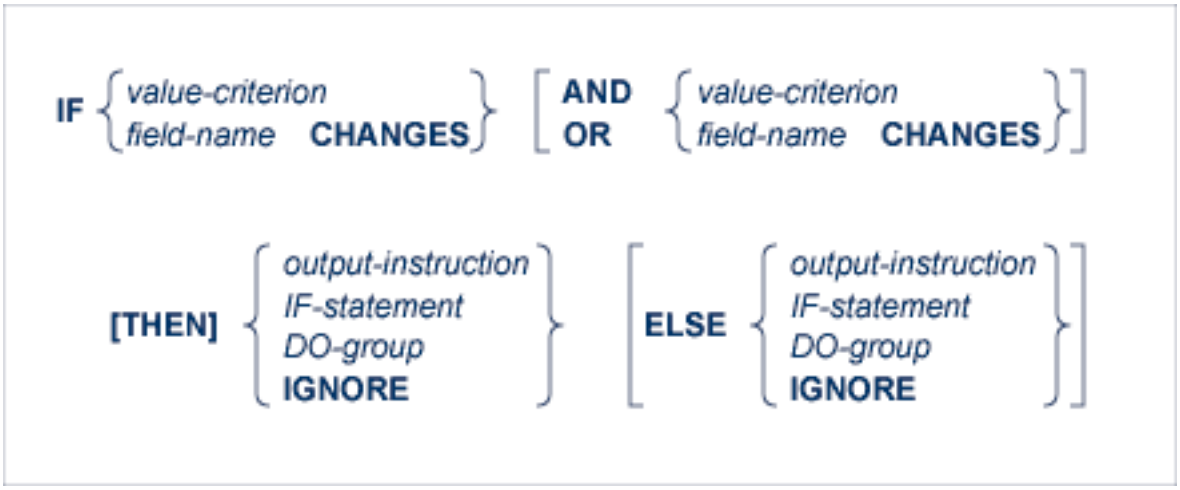
```
SELECT ALL RECORDS FROM FILE 1
   WITH BB ='SMITH'
   DISPLAY AA BB CC
END
```

In this example, despite the fact that the ALL option is used, only the two before-images are selected (because the BB field contains "SMITH" in the before-images). ADASEL ignores all records (in this case, the two after-images) in which the BB field has a value other than "SMITH". If the AFTER IMAGE option were specified, no records would be selected.

IF Statement

The IF statement is used to select records and execute output instructions on a conditional basis.

By default, ADASEL permits up to 20 nested IF statements. For information about changing the default, see the section Overriding ADASEL Defaults with Global Parameters **Overriding ADASEL Defaults with Global Parameters**.



The syntax of the *value-criterion* variable is described in section **value-criterion** . Output instructions are described in section **output-instruction**.

A "*field-name* CHANGES" criterion selects records in which the value of a specified field changed during an update. ADASEL detects the change between the before-image and the after-image. Thus, this criterion is valid only for the A1 (UPDATE) command, which writes both a before-image and an after-image to the protection log. The *field-name* must be the two-character Adabas name of an elementary field in the FDT. It *cannot* refer to a group, periodic group (PE), superdescriptor, subdescriptor, phonetic descriptor, or hyperdescriptor. However, it can refer to a multiple-value field (MU) or a member field of a periodic group (PE); see the section **value-criterion**, particularly in the subsection **Indexes for MUs and PE Member Fields** .

Note: Only the after-image is reported for "IF *field-name* CHANGES" criterion. If you want to report both the before-image and the after-images of a changed field using ADASEL, specify the LOGINFO or EXTENDED options on the OUTPUT instruction for the run. For more information, read ***OUTPUT Instruction***, elsewhere in thissection.

The syntax for DO group is as follows:

A DO group is a sequence of output instructions (NEWPAGE, SKIP, DISPLAY, and OUTPUT). The group must begin with the keyword DO and end with the keyword DOEND. A DO group cannot contain nested IF statements and cannot be nested within another DO group.

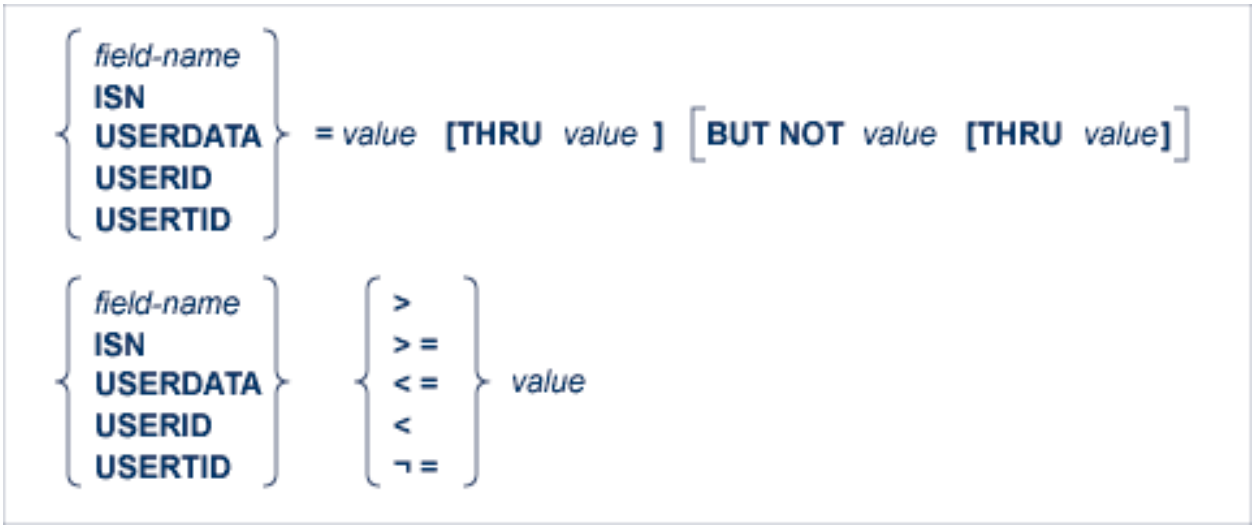
IGNORE instructs ADASEL not to display or output an item.

Example:

```
SELECT ALL FROM FILE 77
   IF AA ='SMITH' THEN
      IF BB CHANGES THEN DO
         DISPLAY 'Field BB changed:' BB AA CC
         SKIP 1 LINE
      DOEND
      ELSE DISPLAY AA BB CC
   ELSE IGNORE
END
```

value-criterion

The value-criterion is used in a WITH clause or an IF statement to select records on the basis of a value or values. It has the following syntax:



The BUT NOT clause excludes a value or subrange of values from the range specified in the equality (=).

Object of the Comparison

ADASEL can compare a value or range of values to the

- contents of the specified field. The `field-name` must be the two-character Adabas name of an elementary field in the FDT. It *cannot* refer to a group, periodic group (PE), superdescriptor, subdescriptor, phonetic descriptor, or hyperdescriptor. However, it can refer to a multiple-value field (MU) or a member field of a periodic group (PE); see **Indexes for MUs and PE Member Fields** .
- ISN; that is, the Adabas internal sequence number of the record.
- USERDATA; that is, the user data written by a C5 command.
- USERID; that is, the user ID (ETID) of the user who added, deleted, or updated the record.
- USERTID; that is, the terminal ID of the user who added, deleted, or updated the record.

Logical Operator

You can express logical operators for equalities and inequalities in words, abbreviations, or symbols as shown in the following table:

Comparison	Words	Abbreviation	Symbol
Equals	EQUAL	EQ	=
Greater than	GREATER THAN	GT	>
Greater than or equal to	GREATER EQUAL	GE	> =
Less than or equal to	LESS EQUAL	LE	< =
Less than	LESS THAN	LT	<
Not equal to	NOTEQUAL	NE	¬=

Note: The hexadecimal representation of the ¬= symbol is X'5F7E'.

Format of the Value

The format of the criterion value depends on the *default format* of the item that is the object of the comparison.

The default format of an Adabas field (*field-name*) is the format specified in the FDT. The following table shows the maximum length (in bytes) and valid formats for expressing the criterion value:

Criterion Value		Max. Bytes	Max. Digits
Field Format in FDT	Valid Formats		
Alphanumeric	Alphanumeric	253	
	Hexadecimal	253	506
Decimal (Packed or Unpacked)	Decimal digits (0-9)	29 *	
Binary	Decimal	4 *	10
	Hexadecimal	126	252
Floating-Point	Hexadecimal	8	16
Fixed-Point	Hexadecimal	4	8
Wide-character	Hexadecimal	253	506

* *Excluding minus sign*

The default formats and maximum lengths (in bytes) for other items are as follows:

Item	Default Format	Criterion Value	
		Valid Formats	Max. Length
ISN	Binary	Decimal, hexadecimal	4
USERDATA	Alphanumeric	Alphanumeric, hexadecimal	30
USERID	Binary, alphanumeric	Decimal, hexadecimal	8
USERTID	Binary, alphanumeric	Decimal, hexadecimal	8

Value Format Example 1:

If the default format is alphanumeric, the value can be expressed in alphanumeric or hexadecimal format.

```
BA EQ 'SMITH' or BA EQ X'E2D4C9E3C8'
```

Value Format Example 2:

If the default format is packed or unpacked decimal, the value is expressed in decimal digits (0-9). A leading minus sign indicates a negative value. Up to 29 digits (excluding the minus sign) are permitted. Other special characters ($, decimal points, commas, etc.) are not permitted.

```
NU = 123456789
NU = -987654321
```

Value Format Example 3:

If the default format is binary, the value can be expressed in hexadecimal or numeric format.

Up to 252 hexadecimal digits (126 bytes) are permitted for a binary Adabas field.

In numeric format, up to 10 decimal digits (4 binary bytes) are permitted. Thus, a binary value expressed in decimal digits can range from -2,147,483,648 through 2,147,483,647.

```
BB = 2147483647  or   BB = X'80000000'
BB = -2147483648 or   BB = X'7FFFFFFF'
```

Alphanumeric Values

Enclose an alphanumeric value in apostrophes:

```
AA ='SMITH'
```

To indicate an apostrophe within an alphanumeric string, use two successive apostrophes with no intervening space or character:

```
JJ ='Smith''s Market'
```

Hexadecimal Values

Begin a hexadecimal value with an "X" and enclose the value in apostrophes:

```
AA = X'E2D4C9E3C8'
```

A hexadecimal value must have an even number of hexadecimal characters:

```
JJ = X'04D2'
```

Continuation Lines

ADASEL treats columns 1-72 as the input line. To continue an alphanumeric or hexadecimal value on additional lines, place the closing apostrophe only at the end of the entire string. The value is concatenated until the closing apostrophe is found.

In an alphanumeric string, ADASEL includes leading and trailing spaces within apostrophes as part of the string; it ignores them in a hexadecimal string.

Example 1: Alphanumeric String

```
                                                                       7
1.....................................................................2

       AA ='THIS IS AN EXAMPLE OF HOW TO CONTINUE AN ALPHANUMERIC VALU
E. KEY THROUGH COLUMN 72 AND CONTINUE IN COLUMN 1 OF THE NEXT
LINE.'

                                                                       7
1.....................................................................2

       AA ='DO NOT CONTINUE AN ALPHA VALUE THIS WAY. LEADING AND
   TRAILING SPACES IN COLUMNS 1-72 ARE INCLUDED.'
```

ADASEL treats the second value above as follows:

```
'DO NOT CONTINUE AN ALPHA VALUE THIS WAY. LEADING AND         TRAILING BLANKS
IN COLUMNS 1-72 ARE INCLUDED.'
```

Example 2: Hexadecimal String

```
                                                                       7
1.....................................................................2

           XX = X'C1C2C3C4C5C6C7C8C9
                  D1D2D3D4D5D6D7D8D9'
```

ADASEL treats the hexadecimal value above as follows:
`X'C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7D8D9'`

Indexes for MUs and PE Member Fields

MU Field or a Member Field of a PE

If the *field-name* refers to an multiple-value field (MU) or to a member field of a periodic group (PE), you must include the index (occurrence number) immediately after the name:

AAi	where "AA" is the field name of an MU and *i* is the index
BBk	where "BB" is a member field of a PE and *k* is the index of the PE

Valid values for *i* and *k* range from "1" through "65,534" if you have Adabas 8 or later installed and if extended MU and PE counts are requested; otherwise the valid values range from "1" through "191".

Note: The use of more than 191 MU fields or PE groups in a file must be explicitly allowed for a file (it is not allowed by default). This is accomplished using the ADADBS MUPEX function or the ADACMP COMPRESS `MUPEX` and `MUPECOUNT` parameters.

Examples:

In file 12, the field JT is an MU. The following statement selects all before-images where the second occurrence of JT is "Programmer":

```
SELECT BI FROM FILE 12
   WITH JT2 = 'Programmer'
   DISPLAY NA
END
```

The field SA is a member of a PE. The following statement selects all records where SA in the third occurrence of the periodic group is greater than or equal to 35000:

```
SELECT ALL FROM 12
   WITH SA3 >= 35000
   DISPLAY NA SA3
END
```

MU Contained Within a PE

If an MU is contained within a PE, *both* indexes (PE and MU) must be specified:

ABk(i)	where "AB" is the name of an MU, *i* is the occurrence of AB, and *k* is the occurrence of the PE to which AB belongs

Example:

In file 211, the multiple-value field ST is a member of a PE. The following statement selects all records in which the third occurrence of ST in the second occurrence of the periodic group is "PAST DUE":

```
SELECT ALL FROM FILE 211
   WITH ST2(3) ='PAST DUE'
   DISPLAY AA BB ST2(3)
END
```

output-instruction

Output instructions include DISPLAY, OUTPUT, SKIP, and NEWPAGE. At least one output instruction is required. Multiple output instructions can be specified, and an output instruction can be included as part of an IF statement. The syntax is shown below:

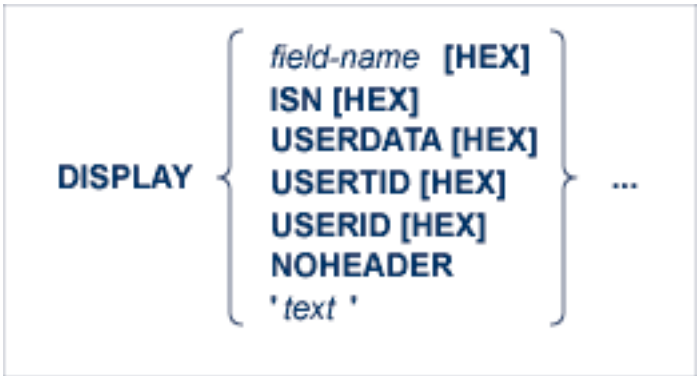
```
[DISPLAY item ... ]
[OUTPUT [{ [WITH] LOGINFO | EXTENDED }] [TO] EXPAn ]
[SKIP n { LINE | LINES } ]
[NEWPAGE]
```

The **DISPLAY instruction** is discussed below ; the OUTPUT instruction is discussed in section *Output Instruction*. See also the discussion *SKIP and NEWPAGE*.

DISPLAY Instruction

DISPLAY writes the output report to DDDRUCK/ DRUCK. The syntax specifies one or more output types. When specifying multiple output types, they are separated by at least one space:

```
DISPLAY { field-name [HEX]
          ISN [HEX]
          USERDATA [HEX]
          USERTID [HEX]
          USERID [HEX]
          NOHEADER
          'text ' } ...
```



where

field-name	displays the contents of the specified field. The *field-name* must be the two-character Adabas field name of an elementary field in the FDT. *field-name cannot* refer to a group, periodic group (PE), superdescriptor, subdescriptor, phonetic descriptor, or hyperdescriptor. However, it can refer to a multiple-value field (MU) or a member field of a PE; indexes for MUs and PE member fields are discussed in section **MU or PE Fields**.
HEX	displays the hexadecimal value corresponding to the type of output. HEX is especially useful if the output contains unprintable characters. Leave at least one space between the type of output and the following HEX keyword.
ISN	displays the ISN of each selected record.
USERDATA	displays records written to the protection log with a C5 command. The file number of the checkpoint file must be specified in the SELECT statement.
USERID	displays the user ID of the user who added, deleted, or updated the record.
USERTID	displays the TID of the user who added, deleted, or updated the record.
NOHEADER	suppresses the header.
'*text*'	displays the text string.

Examples:

Select records that have been modified. Display the text string "The following records were modified:". Then display the fields AA and CC in hexadecimal format and BB in the format defined in the FDT:

```
SELECT UPDATED RECORDS FROM FILE 117
   DISPLAY 'The following records were modified:'
   DISPLAY AA HEX BB CC HEX
END
```

Display the field AA of each new record, along with the user ID and terminal ID of the user who added the record; suppress the header:

```
SELECT NEW RECORDS FROM FILE 211
   DISPLAY AA USERID USERTID NOHEADER
END
```

Default Formats

A field is displayed according to its default format:

Alphanumeric	is displayed as entered, with unprintable characters converted to blanks.
Binary	is displayed in unsigned decimal digits (0-9) if the value is less than X'80000000'; otherwise, the value is displayed in hexadecimal notation.
Packed/unpacked	is displayed in decimal digits (0-9), with a leading minus sign if the value is negative.

MU or PE Fields

If *field-name* refers to an MU or a member field of a PE, you can display a single occurrence or a range of occurrences by specifying the index as part of the field name:

```
DISPLAY AA5
```

If you have Adabas 8 or later installed and if extended MU and PE counts are turned on for a file, valid index values range from "1" through "65534"; otherwise the valid index values range from "1" through "191". In addition, if you specify "N" as the upper limit of an index range, ADASEL displays all occurrences, beginning with the first occurrence in the range.

Note: The use of more than 191 MU fields or PE groups in a file must be explicitly allowed for a file (it is not allowed by default). This is accomplished using the ADADBS MUPEX function or the ADACMP COMPRESS `MUPEX` and `MUPECOUNT` parameters.

You cannot specify the PE name in a DISPLAY statement. To display the entire periodic group, you must specify the name of each field in the group.

If an MU is contained within a PE, both indexes (PE and MU) must be specified. In the index formats shown below, *i* and *j* are the MU indexes; *k* and *l* are the PE indexes. AB refers to a member field of a PE; MB refers to an MU that is a member field of a PE.

Index	Displays . . .
MU*i*	occurrence *i* of the MU
MU*i*-*j*	occurrences *i* through *j* of the MU
MU*i*-N	all occurrences of the MU, starting with occurrence *i*
AB*k*	field AB in occurrence *k* of the PE to which the field belongs
AB*k*-*l*	field AB in occurrences *k* through *l* of the PE
AB*k*-N	field AB in all occurrences of the PE, starting with occurrence *k*
MBk(*i*)	occurrence *i* of MB in occurrence *k* of the PE to which MB belongs

Index	Displays . . .
MB*k - l(i)*	occurrence *i* of MB in occurrences *k* through *l* of the PE
MB*k - l(i-j)*	occurrences *i* through *j* of MB in occurrences *k* through *l* of the PE
MB*k - l(i*-N)	all occurrences of MB (starting with occurrence *i*) in occurrences *k* through *l* of the PE
MB*k*-N*(i - j)*	occurrences *i* through *j* of MB in all occurrences of the PE (starting with occurrence *k* of the PE)
MB*k*-N*(i*-N)	all occurrences of MB (starting with occurrence *i*) in all occurrences of the PE (starting with occurrence *k* of the PE)

Example:

File 12 contains the following PE:

Level	Name	Descriptive Name	Format	Length	Options	Occ
1	JT	JOB TITLE	A	16	DE,MU	12
1	PA	INCOME			PE	12
2	SA	SALARY	P	6	DE,MU	7
2	BO	BONUS	P	5		

The following are valid *DISPLAY* statements for file 12:

```
SELECT NEW FROM FILE 12
   DISPLAY JT1
END
```

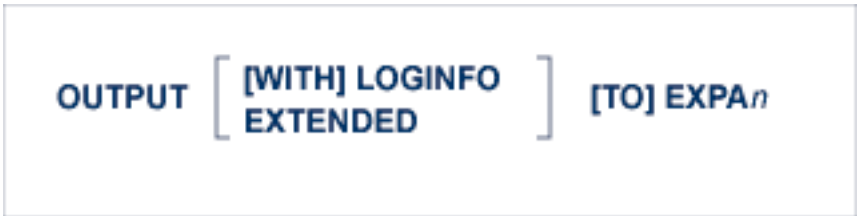
```
SELECT ALL FROM FILE 12
   DISPLAY JT1-5 SA1-5(1-N) BO1-5
END
```

```
SELECT ALL FROM FILE 12
   WITH JT3 ='Programmer' THRU 'Systems Analyst'
   DISPLAY JT3 SA3(1-N) BO3
END
```

```
SELECT UPDATED FROM FILE 12
   DISPLAY JT2-N SA2-N(1-N)
END
```

OUTPUT Instruction

The OUTPUT instruction is used to write the decompressed records from the protection log to an output data set.



Up to 20 output data sets are permitted. The output data set is specified in the EXPA*n* parameter and the DDEXPA*n*/ EXPA*n* job control statement.

Example:

Write the before-images of all updated or deleted records to data set DDEXPA1/ EXPA1:

```
SELECT BEFORE IMAGE FILE 2
   OUTPUT TO EXPA1
END
```

Output Record Format

The format of the output record depends on whether the LOGINFO or EXTENDED parameter is specified. **LOGINFO** and **EXTENDED** are used to display additional information.

Fields common to all output records are shown below. Values in parentheses are field locations when LOGINFO (bytes 32-42) or EXTENDED (bytes 64-74) are specified.

Bytes	Description
0-1	protection log record length (binary)
2-3	set to zeros (X'0000')
4-5	record image type: C'BI' before-image C'AI' after-image C'C5' user data
6-7	Adabas file number (binary)
8-9 (32-33, 64-65)	decompressed record length (including this length field and the ISN)
10-17 (34-41, 66-73)	ISN (binary) or user data from a C5 command
18 (42, 74)	beginning of the decompressed protection log data

Note: The first record in each block is preceded by the two-byte block length and two bytes of nulls or blanks.

The fields of the protection log record are provided in the order, length, and format in which they are defined in the file's FDT. Alphanumeric fields that are longer than the length defined

in the FDT are truncated. Numeric fields that are longer than the length defined in the FDT cause ADASEL to end abnormally.

MUs and PEs are preceded by a one-byte binary field containing the number of occurrences.

Variable-length fields have a default length of zero and are preceded by a one-byte field containing the length of the value (including the length field).

If a field defined with the NC suppression option contains a null value, the null value is decompressed by ADASEL to an empty value (blanks or zeros, depending on the field's format). This type of NC field null processing applies only to ADASEL.

LOGINFO

When LOGINFO is specified, the following additional information is included in each record:

Bytes	Description
8-15	ID of the user who added, deleted, or updated the record
16-19	low-order four bytes of the TID of the user who added, deleted, or updated the record (from the communications ID; TP monitor users only)
20-23	Data Storage RABN where the record was stored (binary)
24-27	data protection block number for the record (binary)
28-31	timestamp of update (binary; high-order four store-clock (STCK) bytes)

EXTENDED

When EXTENDED is specified, the following additional information is included in each record:

Bytes	Description	
8-15	ID of the user (ETID) who added, deleted, or updated the record	
16-23	low-order eight bytes of the terminal ID of the user who added, deleted, or updated the record (from the communications ID; TP monitor users only)	
24-27	Data Storage RABN where the record was stored (binary)	
28-31	data protection block number for the record (binary)	
32-35	timestamp of update (binary; high-order four store-clock (STCK) bytes)	
36	backout indicator:	
	C'B'	record is a result of a backout
	C' '	normal record
37	reserved	
38-41	transaction number	
42-63	reserved	

Output Data Set Designation

The EXPA*n* parameter identifies the output data set. The value of *n* must match the value in the DDEXPA*n*/ EXPA*n* JCL statement. Valid output data set numbers are 1-20 with no leading zeros:

Valid statement	`OUTPUT TO EXPA3`
Invalid statement	`OUTPUT TO EXPA03`

The same rule applies to the DD/EXPAn JCL statement.

Example:

Select all records for file 1. Write decompressed records in which the BA field contains "SMITH" or "SMYTH" to DDEXPA1/ EXPA1. Write all others to DDEXPA2/ EXPA2:

```
SELECT ALL RECORDS FROM FILE 1
IF BA ='SMITH' OR BA ='SMYTH'
   THEN OUTPUT TO EXPA1
ELSE
   OUTPUT TO EXPA2
END
```

NEWPAGE and SKIP Instructions

The NEWPAGE and SKIP instructions control page formatting:

- NEWPAGE forces a page eject before displaying the next line; and
- SKIP prints the specified number of blank lines before displaying the next line of data.

Example:

```
SELECT ALL RECORDS FROM FILE 1
   WITH BA EQUAL 'SMITH' THRU 'SMYTH'
IF BA CHANGES THEN DO
   NEWPAGE
   DISPLAY 'NEW NAME' BA BB BC
   DOEND
ELSE DO
   SKIP 2 LINES
   DISPLAY BA BB BC
   DOEND
END
```

164 Overriding ADASEL Defaults with Global Parameters

ADASEL global parameters override default table and buffer sizes. Overrides are in effect only for the ADASEL run in which the SET GLOBALS statement is specified.

If used, SET GLOBALS must be the first ADASEL input statement. Comment statements can precede it. The statement has the following syntax:

```
SET GLOBALS { global-parameter = value } ...
```

No spaces are permitted between the parameter name, the equal sign, and the value. However, at least one space must separate parameters. Special characters are not permitted as separators. If multiple lines are used, the SET GLOBALS keyword must be repeated on each line. The first non-blank character string that does not begin with a parameter name terminates the SET GLOBALS statement. Thus, trailing comments are not permitted.

ADASEL provides the following global parameters. Default values are underscored.

Parameter	Description
`LST={len | 12000 ↩ }`	Length of the statement table, which is used to store the translated ADASEL statements. Depending on its complexity, a statement is translated into one or more segments. Each segment is 44 bytes plus a value length. For example: `IF BA EQ 'SMITH'...` requires 49 bytes: 44 bytes plus 5 bytes for "SMITH". The default table size (12 ,000 bytes) handles approximately 200 segments. If the table size is exceeded, a SEL003 error occurs.

`NCFLD={n | 10}`
 	
`NCUPD={n | 10}`
	The maximum count of "*field-name* CHANGES" statements allowed in the selection query, and the maximum number of parallel updates during the original session. When a statement includes a CHANGES criterion, ADASEL uses a change pool with `NCFLD * NCUPD` entries to track changed field values. If this pool is too small,

Parameter	Description
	a SEL060 error occurs. In this case, it is necessary to increase one or both of the parameters and then rerun ADASEL.
NF={*n* | 20}	The maximum number of files that can be processed during a single ADASEL run. NF is used to allocate space for the FDT for each file processed. A SEL014 error occurs if the NF value is exceeded. This value is *not* related to the maximum number of output files (DDEXPA*n*/ EXPA*n*); although more than 20 files can be processed during an ADASEL run, a maximum of 20 output files can be written.
NIF={*n* | 20}	Number of nested IF levels permitted.
NOUSERABEND	If specified, ADASEL terminates with condition code 20 instead of a user abend 034 after an error is encountered.

NV={*n* | 100}	Number of field values. NV is used to allocate a table for the evaluation of field values. One entry is required for every field specified in the statements (including duplications). For example, the following statement requires two entries even though the same Adabas field name is used:
	`IF BA ='SMITH'`
	`  THEN OUTPUT TO EXPA1`
	`ELSE IF BA ='SMYTH'`
	`  THEN OUTPUT TO EXPA2`

LPV={*n* | 0}	Length of the PE-value table used in the evaluation of field values for a PE. Normally, ADASEL uses an estimated number of PE occurrences to compute the table size. If the table size is insufficient, a SEL047 error occurs; you can increase the table size using the global LPV parameter as indicated on the screen.
PS={*n* | 60}	The page size parameter is used to alter the number of lines printed before a new page is started. The minimum page size is 2; the maximum is 999.
LS={*n* | 80}	The line size parameter is used to alter the number of printed columns. If an output line is longer than the line size, the line is truncated at the nearest blank. The rest of the line is continued on the next output line, beginning in Column 1. The minimum line size is 1; the maximum is 132.

Example

```
SET GLOBALS LST=15000 NF=15
SET GLOBALS LS=132
```

165 JCL/JCS Requirements and Examples

- BS2000 1062
- z/OS 1063
- z/VM 1064
- VSE 1065

This section describes the job control information required to run ADASEL with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Protection log	DDSIIN	tape/ disk	Sequential log
Selected data	DDEXPAn	tape/ disk	Output by ADASEL
Associator	DDASSORn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADASEL parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADASEL messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADASEL JCL Example (BS2000)

In SDF Format:

```
/.ADASEL LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A S E L ALL FUNCTIONS
/REMARK *
/DELETE-FILE SEL.AUS
/SET-JOB-STEP
/CREATE-FILE SEL.AUS,PUB(SPACE=(48,48))
/SET-JOB-STEP
/ASS-SYSLST L.SEL
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO
/SET-FILE-LINK DDSIIN,ADAyyyyy.SIBA
/SET-FILE-LINK DDEXPA1,SEL.AUS
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADASEL,DB=yyyyy,IDTNAME=ADABAS5B
SELECT ALL FROM FILE 11
DISPLAY AA BB BA BC CA CC
END
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADASEL LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A S E L ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.SEL
/FILE ADA.ASSO ,LINK=DDASSOR1
/FILE ADA.MOD ,LINK=DDLIB
/FILE ADAyyyyy.SIBA ,LINK=DDSIIN
/FILE SEL.AUS ,LINK=DDEXPA1 ,SPACE=(48,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADASEL,DB=yyyyy,IDTNAME=ADABAS5B
SELECT ALL FROM FILE 11
DISPLAY AA BB BA BC CA CC
END
/LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Protection log	DDSIIN	tape/ disk	Sequential log
Selected data	DDEXPAn	tape/ disk	Output by ADASEL
Associator	DDASSORn	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADASEL parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADASEL messages	DDDRUCK	printer	*Messages and Codes*

ADASEL JCL Example (z/OS)

```
//ADASEL    JOB
//*
//*   ADASEL:
//*      SELECT PROTECTION DATA
//*
//SEL       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD          <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1    <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1    <=== DATA
```

```
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1    <=== WORK
//DDSIIN    DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.PLOG5     <=== OUTPUT ADARES
//*                                                            PLCOPY
//DDEXPA1   DD   DISP=(,CATLG),DSN=EXAMPLE.DByyyyy.EXPA1, <= OUTPUT ADASEL
//               SPACE=(TRK,(10,5),RLSE),VOL=SER=vvvvvv,UNIT=uuuu
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADASEL,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *

*
*    SELECT UPDATES FOR FILE NUMBER 1
*
SELECT ALL FROM FILE 1
  OUTPUT TO EXPA1
END
/*
```

Refer to ADASEL in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Protection log	DDSIIN	tape/ disk	Sequential log
Selected data	DDEXPAn	tape/ disk	
Associator	DDASSORn	disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADASEL parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADASEL messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDSIIN,DSN=ADABASVv.SIBA,UNIT=181,VOL=SIBAXX
DATADEF DDEXPA1,DSN=ADABASVv.EXPA1,MODE=A
DATADEF DDPRINT,DSN=ADASEL.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADASEL.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNSEL.CONTROL,MODE=A
```

```
DATADEF DDKARTE,DSN=ADASEL.CONTROL,MODE=A
LOAD ADARUN ( START
```

Contents of RUNSEL CONTROL A1

```
ADARUN PROG=ADASEL,DEVICE=dddd,DB=yyyyy
```

Contents of ADASEL CONTROL A1

```
SELECT ALL FROM FILE 11
   DISPLAY ISN
END
```

VSE

File	File Name	Storage	Logical Unit	More Information
Protection log	SIIN	tape disk	SYS010 *	Sequential log
Selected data	EXPA1-20	tape disk	SYS011-SYS030 *	Output by ADASEL
Associator	ASSORn	disk	*	
ADARUN parameters	SYSRDR CARD	reader/ tape/ disk		*Operations*
ADASEL parameters	SYSIPT	reader		
ADARUN messages	SYSLST	printer		*Messages and Codes*
ADASEL messages	SYS009	printer		*Messages and Codes*

* *Any programmer logical unit may be specified.*

ADASEL JCS Example (VSE)

```
* $$ JOB JNM=ADASEL,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADASEL
*       SELECT PROTECTION DATA
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// PAUSE MOUNT LOAD INPUT PLOG FILE ON TAPE cuu
// ASSGN SYS010,TAPE
// TLBL SIIN,'EXAMPLE.DByyyyy.PLOG5'
// DLBL EXPA1,'EXAMPLE.ADAyyyyy.EXPA1'
// EXTENT SYS015,,,,ssss,nnnn
```

```
// ASSGN SYS015,DISK,VOL=vvvvvv,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADASEL,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
*    SELECT UPDATES FOR FILE NUMBER 1
*
SELECT ALL FROM FILE 1
  OUTPUT TO EXPA1

END
/*
/&
* $$ EOJ
```

Refer to member ADASEL.X for this example.

XXI ADAULD Utility: Unload Files

This chapter covers the following topics:

Functional Overview
UNLOAD FILE: Unload Specified File
ADAULD Input Processing
ADAULD Output Processing
ADAULD User Exit 9
JCL/JCS Requirements and Examples

166 Functional Overview

The ADAULD utility unloads an Adabas file. Adabas files are unloaded to:

- permit the data to be processed by a non-Adabas program. In this case, the file must also be decompressed after unloading using the DECOMPRESS function of the ADACMP utility.
- create one or more test files, all of which contain the same data. This procedure requires that a file be unloaded, and then reloaded as a test file having a different file number.
- change the field definition table (FDT). This requires that the file be unloaded, decompressed, compressed using the modified field definitions, and reloaded. If the ADADBS utility is used to add field definitions to a file, the file does not need to be unloaded first.

The sequence in which the records are unloaded may be

physical	the order in which they are physically positioned within Data Storage.
logical	a sequence controlled by the values of a user-specified descriptor.
ISN	ascending ISN sequence.

Selection criteria (SELCRIT/SELVAL) are optionally used to indicate a subset of records to be unloaded:

- If no such criterion is provided, all records are unloaded in physical sequence.
- If a selection criterion is specified but no sort sequence (SORTSEQ), the specified records are unloaded in ISN sequence.
- If both a selection criterion and a sort sequence are provided, the selected records are sorted in the Work pool area of the nucleus and are unloaded in the specified sort sequence.
- If no records that match the selection criteria are found, ADAULD creates a file containing only the FDT and issues condition code 4 in register 15.

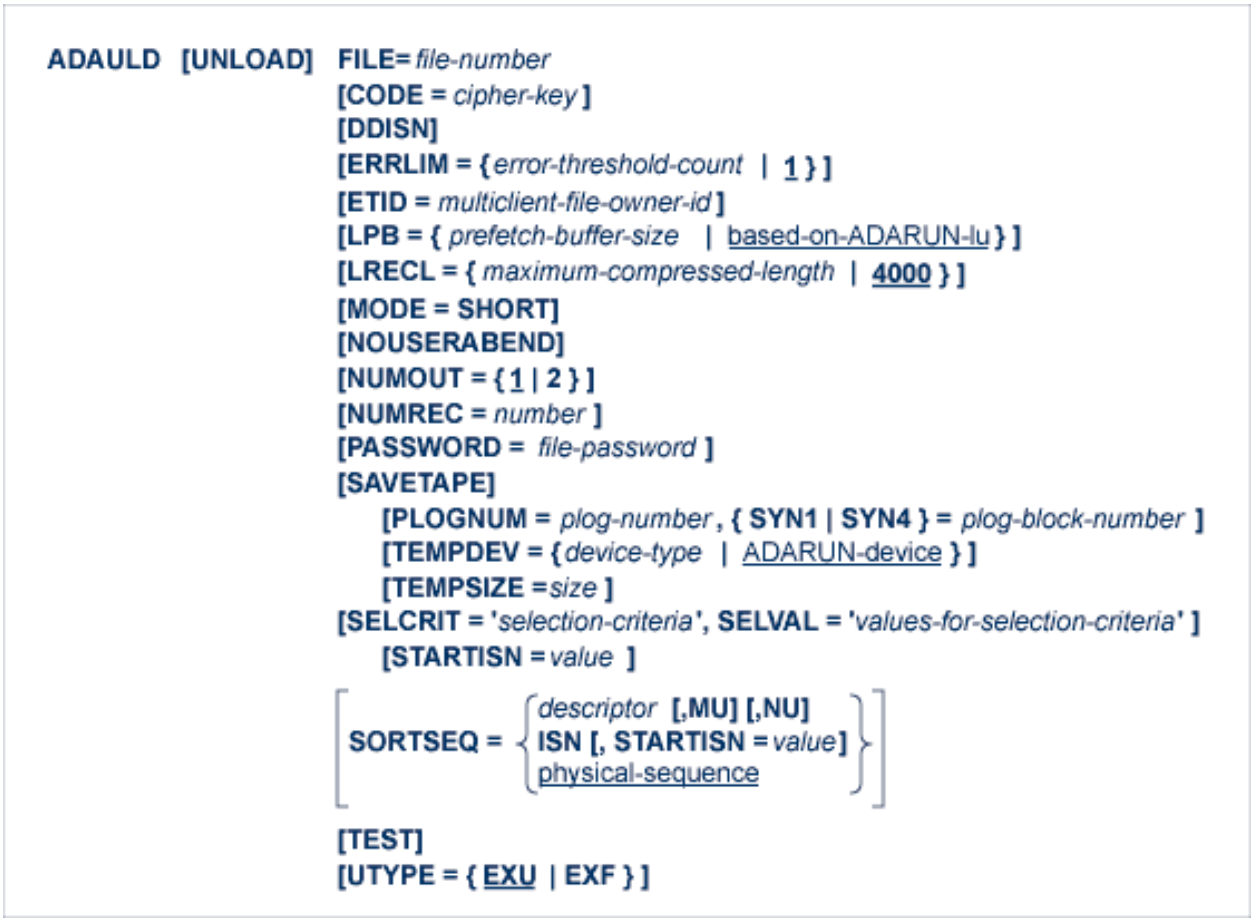
The unloaded record output is in compressed format. The output records have the same format as the records produced by the ADACMP utility.

When using the MODE=SHORT option, descriptor entries (which are required to create the normal index and upper index for the file) are omitted during the unload process. This reduces the time required for unloading. Note, however, that output created using MODE=SHORT has a different FDT from the same file unloaded without MODE=SHORT, since all descriptor information is removed.

Note: An interrupted ADAULD UNLOAD FILE run must be reexecuted from the beginning.

167 UNLOAD FILE: Unload Specified File

- Essential Parameter 1072
- Optional Parameters and Subparameters 1072
- Examples 1078



This chapter describes the syntax and parameters of the UNLOAD FILE function.

Essential Parameter

FILE

FILE specifies the number of the file to be unloaded. Neither the checkpoint file nor the security file can be unloaded.

Optional Parameters and Subparameters

CODE: Cipher Code

If the file to be unloaded is ciphered, CODE *must* supply the appropriate cipher code.

DDISN: Create DD/ISN Output File of Unloaded ISNs

Specifying the DDISN parameter instructs ADAULD to write the list of unloaded ISNs to the sequential output file DD/ISN. DD/ISN is structured so that it can be used as input to ADALOD UPDATE for the purpose of deleting the unloaded records.

If the DDISN keyword is specified

- but the DD/ISN file is missing in the JCL, ADAULD terminates with error 081.
- and SORTSEQ specifies a hyperdescriptor or descriptor that refers to a multiple-value field, ADAULD terminates with error 133 because the DD/ISN may contain duplicate ISNs.

ERRLIM: Error Threshold

ERRLIM sets the maximum number of nucleus response codes accepted by ADAULD before operation terminates. The default setting is one, which means that the first error terminates ADAULD with error 124.

The ERRLIM value may be set higher than one to tolerate conditions that occur intermittently such as response code 255 (all attached buffers allocated). In this case, the utility terminates with return code 8 and no user abend. The output file of ADAULD can be used, although records may be missing depending on the nucleus response code returned.

ETID: Multiclient File Owner ID

When unloading multiclient files, the ETID parameter can be used to restrict UNLOAD processing to only the records owned by the specified user. If the ETID parameter is omitted, all records are unloaded.

If the SELCRIT/SELVAL parameters are specified for a multiclient file, the ETID parameter *must* also be specified.

LPB: Prefetch Buffer Size

LPB specifies the size of the internal prefetch buffer. The maximum value is 32767 bytes.

By default, ADAULD attempts to make the prefetch buffer as large as possible to achieve the best performance. The LPB parameter gives the user the option of making the prefetch buffer smaller. This might be advisable, for example, if heavy use of prefetching causes ADAULD to consume too much nucleus resource relative to other users.

The default value depends on the length of the intermediate user buffer set by the ADARUN LU parameter. ADAULD subtracts the space required to accommodate the Adabas control information (108 bytes) and the specified maximum compressed record length (LRECL) from the LU value to determine the default LPB value. The result must be equal to or less than the maximum value allowed for LPB; that is, 32767 bytes.

The default value for LU is set to 65535 bytes, the maximum size, to accommodate the record buffer of utilities such as ADAULD that need the nucleus. If the LU value is too small, ADAULD may reduce the specified value for the LPB parameter.

LRECL: Maximum Compressed Record Length

LRECL specifies, in bytes, the maximum compressed record length (including DVT) to be returned.

This length is used as an Adabas record buffer length. If this value is too small, a response code 53 occurs. The default is 4000 bytes; the maximum allowed is 32760 bytes.

MODE=SHORT: Exclude Descriptor Information

This parameter indicates whether the descriptor information used to build the normal index and upper index are to be included in the output.

If MODE=SHORT is specified, no descriptor information will be unloaded, and all descriptor information is stripped from the field definition table (FDT) when it is written to the output data set.

If the output is to be used as direct input to the ADALOD utility, the file will have no descriptors.

In the case of superdescriptors, MODE=SHORT unloads them as superfields. If the output is used as direct input to ADALOD, the loaded file will have superfields.

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMOUT: Number of Output Files

NUMOUT specifies the number of output files to be produced. If the number is greater than one, user exit 9 must be used to control DD/OUT1 or DD/OUT2 output file selection. For additional information, see the *Adabas DBA Reference* documentation. Permitted values are 1 (default) and 2.

NUMREC: Number of Records to Be Unloaded

NUMREC limits the number of records to be unloaded. No limit will be in effect if the parameter is omitted.

PASSWORD: File Password

The PASSWORD parameter must be specified if the file to be unloaded is password-protected.

PLOGNUM: Protection Log Number

When SAVETAPE is specified and an online save tape is to be used as input to ADAULD, the corresponding protection log is expected as a DD/PLOG sequential input data set.

If an online save tape created using ADASAV version 5.1 is to be used, the additional parameters PLOGNUM and SYN1 or SYN4 must be specified:

- PLOGNUM specifies the number of the nucleus protection log used while the ADASAV SAVE operation was active; and
- SYN1 or SYN4 specifies the block number containing the SYN1 or SYN4 checkpoint at which the corresponding ADASAV SAVE operation began.

For online save tapes created using ADASAV version 5.2 or above, this information is included on the tape. You can specify PLOGNUM or SYN1 or SYN4 to override the tape information.

SAVETAPE

SAVETAPE is used to unload a file from a save tape. This is useful when moving a file from a save tape with one block size to a database with another, or when using a file from a save tape in one or another test environment.

If an online save tape is used, the TEMPDEV parameter must also be specified. If the online save tape was created using ADASAV version 5.1, the parameters PLOGNUM and SYN1 or SYN4 must also be specified. PLOGNUM and SYN1 or SYN4 may be specified for online save tapes created using ADASAV version 5.2 or above to override the information included on the tape.

For more information, see the section **Processing a Save Tape as Input**.

The SORTSEQ and SELCRIT parameters may not be used with SAVETAPE.

The ETID parameter may not be used with SAVETAPE. User exit 9 must be used to select records for a particular client of a multiclient file. For more information, see the section **ADAULD User Exit 9**.

If the file to be unloaded from the save tape is ciphered, the CODE parameter must be specified as usual.

Note: Special SAVETAPE functions are available for use with the Adabas Delta Save Facility. For more information, see the *Adabas Delta Save Facility* documentation.

SELCRIT: Selection Criterion

The SELCRIT parameter may be used to restrict the unloaded records to those which meet the selection criterion provided. The selection criterion must be provided using the search buffer syntax, as described in the *Adabas Command Reference* documentation.

For multiple criteria, you can specify each criterion with a separate ADAULD SELCRIT statement, as follows:

```
ADAULD SELCRIT ='AA, 20, A, D,'
ADAULD SELCRIT ='AB, 10, A.'
```

ADAULD concatenates this to:

```
'AA, 20, A, D, AB, 10, A.'
```

The values that correspond to the selection criterion must be provided using the SELVAL parameter.

SELVAL: Values for Selection Criteria

SELVAL specifies the values corresponding to the selection criteria specified with the SELCRIT parameter. The value formats are the same as those used for the Value Buffer, as described in the *Adabas Command Reference* documentation.

Values can be on multiple lines. Packed decimal or binary values can be in hexadecimal format, as shown in the following example:

```
SELVAL='PARIS   '
SELVAL=X'00149C'
SELVAL='AB100'
```

SORTSEQ: Unload Sequence

SORTSEQ specifies the sorting sequence for unloaded ISNs. If SORTSEQ is not specified, ISNs are unloaded in physical sequence.

If a descriptor name is specified, the records are unloaded in the ascending logical sequence of the descriptor values. You can specify the name of a descriptor, subdescriptor, super-descriptor, or hyperdescriptor. *Do not* refer to a field in a periodic group.

- MU *must* be specified if the descriptor name refers to a multiple-value field. In this case, the same record is unloaded once for each different value for the descriptor in the record in ascending value order. If MU is not specified (the default), ADAULD rejects MU descriptors and issues an error message.
- NU *must* be specified if the descriptor name refers to a field defined with the null suppression (NU) option. In this case, records of the descriptor that contain null values are not unloaded. If NU is not specified (the default), ADAULD rejects NU descriptors.

Note: Even when the descriptor field is not null suppressed, the record is *not* represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.

If SORTSEQ=ISN is specified, the records are unloaded in ascending ISN sequence.

If both SELCRIT/SELVAL and SORTSEQ are specified, the records are sorted in the Work pool area of the nucleus. Therefore, the ADARUN LS and LWP session parameters must provide enough space; see the *Adabas Operations* documentation for descriptions of the LS and LWP parameters.

STARTISN: Starting ISN

STARTISN is used with the SELCRIT/SELVAL and SORTSEQ parameters to restrict the unloaded records according to ISN. Specifying STARTISN alone is not allowed.

- Specifying STARTISN with SELCRIT/SELVAL causes all records with ISNs equal to or greater than the STARTISN-specified value *and* with field contents satisfying the SELCRIT/SELVAL criterion to be unloaded in ascending ISN sequence by descriptor name.
- Specifying STARTISN with SORTSEQ=ISN unloads all records beginning with the STARTISN-specified record in ISN sequence.

SYN1|SYN4: Starting Block Number

When SAVETAPE is specified and an online save tape is to be used as input to ADAULD, the corresponding protection log is expected as a DD/PLOG sequential input data set.

If an online save tape created using ADASAV version 5.1 is to be used, the additional parameters PLOGNUM and SYN1 or SYN4 must be specified:

- PLOGNUM specifies the number of the nucleus protection log used while the ADASAV SAVE operation was active; and
- SYN1 or SYN4 specifies the block number containing the SYN1 or SYN4 checkpoint at which the corresponding ADASAV SAVE operation began.

For online save tapes created using ADASAV version 5.2 or above, this information is included on the tape. You can specify PLOGNUM or SYN1 or SYN4 to override the tape information.

TEMPDEV: Temporary Storage Device Type

When SAVETAPE is specified and an online save tape is to be used as input to ADAULD, a temp data set is used to store intermediate data during processing. The TEMPDEV parameter indicates the device type to be used for the temp data set. This parameter is required only if the device type to be used is different from the standard device type assigned to Temp by the ADARUN DEVICE parameter.

The block size of the temp data set must be at least as large as the largest Data Storage block size of the file to be unloaded, plus 16 bytes.

TEMPSIZE: Temporary Storage Size

TEMPSIZE specifies the size of the temp data set for the file. The size can be either in cylinders or blocks (followed by a "B").

The temp data set must be large enough to store all Data Storage blocks from the protection log. In the worst case scenario, it must have as many blocks as the file has Data Storage blocks but need not be larger than the PLOG data set. If the temp data set is too small, ADAULD error-136 (temp data set too small) is returned.

TEST: Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UTYPE: User Type

The user type to be in effect for the unload process.

- If EXU (the default) is specified, the file cannot be updated, but other users can read the file.
- If EXF is specified, only ADAULD can use the file; no other users can read or write the file.

Examples

Example 1:

```
ADAULD FILE=6
```

File 6 is to be unloaded. The records are to be unloaded in the sequence in which they are physically positioned in Data Storage.

Example 2:

```
ADAULD FILE=6,SORTSEQ=AA
```

File 6 is to be unloaded. The values for the descriptor AA are to be used to control the sequence in which the records are to be unloaded.

Example 3:

```
ADAULD FILE=6,SORTSEQ=ISN
```

File 6 is to be unloaded. The records are to be unloaded in ascending ISN sequence.

Example 4:

```
ADAULD FILE=6,SORTSEQ=ISN,STARTISN=10000
```

File 6 is to be unloaded. The records are to be unloaded in ascending ISN sequence. Only records which have an ISN equal or greater than 10000 are to be unloaded.

Example 5:

```
ADAULD FILE=6,SORTSEQ=AB,MODE=SHORT
```

File 6 is to be unloaded. The values for the descriptor AB are to be used to control the sequence in which the records are to be unloaded. The entries used to create the normal index and upper index are not to be unloaded. All descriptor information is removed from the field definition table (FDT) in the output.

Example 6:

```
ADAULD FILE=6,SELCRIT='AA,1,S,AA,2.',SELVAL='AMM'
```

File 6 is to be unloaded. Only records with AA=A through MM are to be unloaded. The records are returned in ISN sequence.

Example 7:

```
ADAULD FILE=6,UTYPE=EXF
```

File 6 is to be unloaded. The user type is indicated as EXF which locks the file during unload processing, preventing other users from reading or writing the file.

168 ADAULD Input Processing

- Processing a Save Tape as Input 1082

ADAULD is used to unload an Adabas file from

- a database; or
- a save tape (if the SAVETAPE keyword is specified).

Processing a Save Tape as Input

If a save tape is used as input, a DD/SAVE sequential file is expected. Database or file save tapes created online and offline are acceptable. The save tape must have been created using ADASAV version 5.1 or above.

The ADARUN DBID specified for the ADAULD run must match the DBID found on the save tape.

If the file has hyperdescriptors defined, the corresponding hyperexits must be specified in the ADARUN parameters for ADAULD. If the hyperexit routines are no longer available, the file must be unloaded with MODE=SHORT specified. See the *Adabas DBA Reference* documentation for more information about hyperexits.

For an online save tape:

- the corresponding protection log is expected as a DD/PLOG sequential input data set.
- a temp (DD/TEMPR1) data set is required as intermediate storage for the Data Storage blocks on the protection log. The TEMPSIZE and TEMPDEV parameters must be specified.

If an online save tape created using ADASAV version 5.1 is to be used, the additional parameters PLOGNUM and SYN1 or SYN4 must be specified:

- PLOGNUM specifies the number of the nucleus protection log used while the ADASAV SAVE operation was active; and
- SYN1 or SYN4 specifies the block number containing the SYN1 or SYN4 checkpoint at which the corresponding ADASAV SAVE operation began.

For online save tapes created using ADASAV version 5.2 or above, this information is included on the tape. You can specify PLOGNUM or SYN1 or SYN4 to override the tape information.

The ADAULD utility protocol on DD/DRUCK displays a short header indicating the kind of save tape encountered, when it was created, the version of ADASAV used to create it, the database ID found on the save tape, and for online save tapes, the session number of the corresponding protection log and the block number of the SYN1/SYN4 checkpoint:

```
A D A R E P    Vv.r   SMs     DBID = nnnnn  STARTED           yyyy-mm-dd   hh:mm:ss ↩


PARAMETERS:
-----------

ADAULD UNLOAD FILE=3, SAVETAPE

************************************************************
*                                                          *
*  UNLOAD FROM            ONLINE DATABASE SAVE             *
*  CREATED AT             yyyy-mm-dd  hh:mm:ss             *
*  BY ADASAV VERSION      V vr                             *
*  DBID                   nnnnn                            *
*  DSID                   1 / 0 /   yyyy-mm-dd  hh:mm:ss   *
*  PLOG SESSION NR        17                               *
*  SYN1 BLOCK NR          137                              *
*                                                          *
************************************************************
```

ADAULD first reads the file control block (FCB) and file definition table (FDT) from the save tape. Then:

- for offline save tapes, ADAULD scans the tape to find the file's Data Storage RABNs, extracts the Data Storage records, and for each Data Storage record, generates the descriptor values according to the FDT.
- for online save tapes, ADAULD scans the protection log and copies the latest version of each Data Storage block of the relevant file to the temp data set. The location of a Data Storage block on the temp data set is maintained in a directory in main memory. Then, ADAULD scans the save tape for Data Storage blocks of the file. If more recent versions of Data Storage blocks exist on the temp data set, they are actually unloaded to DD/OUT1 or DD/OUT2. Note that in this case, two parallel tape units are required: concatenating the save tape and the protection log as for ADASAV RESTONL is not possible.

After opening the DD/SAVE and DD/PLOG input data sets, ADAULD cross-checks to ensure that the input tapes are correct. If an invalid save tape is encountered, ADAULD terminates and displays error-134 (invalid save tape supplied). If an invalid protection log tape is encountered, ADAULD terminates with error-135 (invalid protection log supplied).

169 ADAULD Output Processing

ADAULD unloads the records in the specified sequence. The unloaded records are written to one or both of two sequential data sets: DD/OUT1 and DD/OUT2. Writing to these output data sets is controlled by user exit 9.

The records output are identical in format to the output produced by the ADACMP utility unless the MODE=SHORT option is used, in which case the descriptor entries required for the normal index and upper index are omitted and the descriptor information is removed from the Adabas FDT. The ISN of the record immediately precedes the compressed data record, and is provided as a four-byte binary number.

Specifying the DDISN parameter instructs ADAULD to write the list of unloaded ISNs to a sequential output file DD/ISN. Only one DD/ISN file is created, containing the superset of ISNs written to either or both of DD/OUT1 and DD/OUT2. ISNs that are rejected by user exit 9 are not written to DD/ISN.

DD/ISN is structured so that it can be used as input to ADALOD UPDATE for the purpose of deleting the unloaded records.

The number of ISNs written to DD/ISN is displayed in the ADAULD statistics on the DD/DRUCK utility protocol:

```
A D A U L D  Statistics
-----------------------

Number of Output Data Sets        =  1
Number of Requested Records       =  16777215
STARTISN                          =  0
Options                           =  DVT
Unload Sequence                   =  PHYS SEQ

Number of Records Read            =         307
Number of Records Written         =         307
Number of Record Segments Read    =         777
Number of Record Segments Written =         777
Records Written to DDOUT1         =         307
Records Written to DDOUT2         =           0
Records Rejected by USEREXIT-9    =           0
Number of ISNS Written to DDISN   =         307
```

If the DDISN parameter is specified, the number of ISNs written to DD/ISN should always be the number of records read minus the number of records rejected by user exit 9.

The ISNs on the DD/ISN file are ISNs as visible to applications; that is, the internal ISN as stored in a Data Storage record plus MINISN-1.

170 ADAULD User Exit 9

User exit 9 is called (when present) for each record selected before writing the record to the output data set. The user exit is supplied with the record address, and returns an action code as follows:

1	Write record to DD/OUT1;
2	Write record to DD/OUT2;
3	Write record to DD/OUT1 and DD/OUT2;
I	Ignore this record.

The above data sets must have the same block size. See the *Adabas DBA Reference* documentation for more information about user exits.

171 JCL/JCS Requirements and Examples

- BS2000 1090
- z/OS 1092
- z/VM 1094
- VSE 1096

This section describes the job control information required to run ADAULD with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

BS2000

Data Set	Link Name	Storage	More Information
Unloaded data	DDOUT1	tape/ disk	Output by ADAULD (see note)
Unloaded data	DDOUT2	tape/ disk	Output by ADAULD (see note)
Unloaded ISNs	DDISN	tape/ disk	Required with DDISN
Save tape	DDSAVE	tape/ disk	Required with SAVETAPE
Sequential PLOG	DDPLOG	tape/ disk	Required for online save tapes
Temp area	DDTEMPR1	disk	Required for online save tapes
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADAULD parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAULD messages	SYSLST/ DDDRUCK		*Messages and Codes*

Note: DDOUT1 and DDOUT2 must have the same block size; otherwise, an ADAULD error will occur. DDOUT2 is required only if NUMOUT=2 is specified.

ADAULD JCL Examples (BS2000)

Unload from Database

In SDF Format:

```
/.ADAULD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A U L D NON-SAVETAPE FUNCTIONS
 /REMARK *
 /DELETE-FILE ADAyyyyy.OUT1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.OUT1,PUB(SPACE=(480,48))
 /SET-JOB-STEP
 /ASS-SYSLST L.ULD
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDOUT1,ADAyyyyy.OUT1
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
```

```
ADARUN PROG=ADAULD,DB=yyyyy,IDTNAME=ADABAS5B
ADAULD FILE=1,SORTSEQ=AA
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAULD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A U L D NON-SAVETAPE FUNCTIONS
 /REMARK *
 /SYSFILE SYSLST=L.ULD
 /FILE ADA.MOD ,LINK=DDLIB
 /FILE ADAyyyyy.OUT1 ,LINK=DDOUT1 ,SPACE=(480,48)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAULD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAULD FILE=1,SORTSEQ=AA
 /LOGOFF NOSPOOL
```

Unload from Offline Save Tape

In SDF Format:

```
/.ADAULD LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A U L D SAVETAPE FUNCTION
 /REMARK *
 /DELETE-FILE ADAyyyyy.OUT1
 /SET-JOB-STEP
 /CREATE-FILE ADAyyyyy.OUT1,PUB(SPACE=(480,48))
 /SET-JOB-STEP
 /ASS-SYSLST L.ULD
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDOUT1,ADAyyyyy.OUT1
 /SET-FILE-LINK DDSAVE,ADAyyyyy.SAVE
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAULD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAULD FILE=1,SAVETAPE
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAULD LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A U L D SAVETAPE FUNCTION
 /REMARK *
 /SYSFILE SYSLST=L.ULD
 /FILE ADA.MOD ,LINK=DDLIB
 /FILE ADAyyyyy.OUT1 ,LINK=DDOUT1 ,SPACE=(480,48)
 /FILE ADAyyyyy.SAVE ,LINK=DDSAVE
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAULD,DB=yyyyy,IDTNAME=ADABAS5B
 ADAULD FILE=1,SAVETAPE
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Unloaded data	DDOUT1	tape/ disk	Output by ADAULD (see note)
Unloaded data	DDOUT2	tape/ disk	Output by ADAULD (see note)
Unloaded ISNs	DDISN	tape/ disk	Required with DDISN
Save tape	DDSAVE	tape/ disk	Required with SAVETAPE
Sequential PLOG	DDPLOG	tape/ disk	Required for online save tapes
Temp area	DDTEMPR1	disk	Required for online save tapes
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
ADAULD messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations*
ADAULD parameters	DDKARTE	reader	

Note: DDOUT1 and DDOUT2 must have the same block size; otherwise, an ADAULD error will occur. DDOUT2 is required only if NUMOUT=2 is specified.

ADAULD JCL Examples (z/OS)

Unload a File

```
//ADAULD    JOB
//*
//*   ADAULD:
//*      UNLOAD A FILE
//*
//ULD       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X

//DDOUT1    DD
DISP=(,CATLG),DSN=EXAMPLE.DByyyyy.OUT1,UNIT=DISK, <===
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDCARD    DD   *
ADARUN PROG=ADAULD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAULD FILE=1,SORTSEQ=AA

00000100
/*
```

Refer to ADAULD in the JOBS data set for this example.

Unload a File from Save Tape Created Offline

```
//ADAULDS   JOB
//*
//*   ADAULD:
//*      UNLOAD A FILE  FROM AN OFFLINE SAVE TAPE
//*
//ULD       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDSAVE    DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.SAVE   <=== SAVE DATASET
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
```

```
//SYSUDUMP  DD   SYSOUT=X
//DDOUT1    DD   DISP=(,CATLG),DSN=EXAMPLE.DByyyyy.OUT1,UNIT=DISK, <===
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDCARD    DD   *
ADARUN PROG=ADAULD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAULD FILE=1,SAVETAPE

00000100
/*
```

Refer to ADAULDS in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Unloaded data	DDOUT1	tape/ disk	Output by ADAULD (see note)
Unloaded data	DDOUT2	tape/ disk	Output by ADAULD (see note)
Unloaded ISNs	DDISN	tape/ disk	Required with DDISN
Save tape	DDSAVE	tape/ disk	Required with SAVETAPE
Sequential PLOG	DDPLOG	tape/ disk	Required for online save tapes
Temp area	DDTEMPR1	disk	Required for online save tapes
Recovery log (RLOG)	DDRLOGR1	disk	Required for ADARAI
ADAULD messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAULD parameters	SYSIPT	disk/ terminal/ reader	

Note: DDOUT1 and DDOUT2 must have the same block size; otherwise, an ADAULD error will occur. DDOUT2 is required only if NUMOUT=2 is specified.

Examples (z/VM)

Unload from Database

```
DATADEF DDOUT1,DSN=ADABASVv.ULD1,UNIT=181,VOL=ULDF1
DATADEF DDPRINT,DSN=ADAULD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAULD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNULD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAULD.CONTROL,MODE=A
ADARUN
```

Contents of RUNULD CONTROL A1:

```
ADARUN PROG=ADAULD,DEVICE=dddd,DB=yyyyy
```

Contents of ADAULD CONTROL A1:

```
ADAULD FILE=1,SORTSEQ=AA
```

Unload from Offline Save Tape

```
DATADEF DDOUT1,DSN=ADABASVv.ULD1,UNIT=181,VOL=ULDF1
DATADEF DDSAVE,DSN=ADABASVv.ULD1,UNIT=181,VOL=ULDF1
DATADEF DDPRINT,DSN=ADAULD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAULD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNULD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAULD.CONTROL,MODE=A
ADARUN
```

Contents of RUNULD CONTROL A1:

```
ADARUN PROG=ADAULD,DEVICE=dddd,DB=yyyyy
```

Contents of ADAULD CONTROL A1:

```
ADAULD FILE=1,SAVETAPE
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Unloaded data	OUT1	tape disk	SYS010 see note 1	Output by ADAULD (see note 2)
Unloaded data	OUT2	tape disk	SYS011 see note 1	Output by ADAULD (see note 2)
Unloaded ISNs	ISN	tape disk	SYS012 see note 1	Required with DDISN
Save tape	SAVE	tape disk	SYS013 see note 1	Required with SAVETAPE
Sequential PLOG	PLOG	tape disk	SYS014 see note 1	Required for online save tapes
Temp area	TEMPR1	disk	see note 1	Required for online save tapes
Recovery log (RLOG)	RLOGR1	disk	see note 1	Required for ADARAI
Messages	SYSLST	printer		*Messages and Codes*
ADARUN parameters	SYSRDR CRD	reader/ tape/ disk		*Operations*
ADAULD parameters	SYSIPT	reader		

Notes:

1. Any programmer logical unit can be used.
2. OUT1 and OUT2 must have the same block size; otherwise, an ADAULD error will occur. OUT2 is required only if NUMOUT=2 is specified.

ADAULD JCS Examples (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

Unload a File from a Database

```
* $$ JOB JNM=ADAULD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D

// JOB ADAULD
*       UNLOAD A FILE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS004,DISK,VOL=vvvvvv,SHR
// DLBL OUT1,'EXAMPLE.ADA99.OUT1'
// EXTENT SYS004,,,,ssss,nnnn
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAULD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAULD FILE=1,SORTSEQ=AA
/*
/&
* $$ EOJ
```

Refer to member ADAULD.X for this example.

Unload a File from Save Tape Created Offline

```
* $$ JOB JNM=ADAULDS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAULDS
*       UNLOAD A FILE FROM AN OFFLINE SAVE TAPE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS004,DISK,VOL=vvvvvv,SHR
// DLBL OUT1,'EXAMPLE.ADA99.OUT1'
// EXTENT SYS004,,,,ssss,nnnn
// ASSGN SYS013,TAPE
// PAUSE MOUNT LOAD SAVE FILE ON TAPE cuu
// TLBL SAVE,'EXAMPLE.DByyyyy.SAVE'
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAULD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAULD FILE=1,SAVETAPE
/*
/&
* $$ EOJ
```

Refer to member ADAULDS.X for this example.

XXII ADAVAL Utility: Validate the Database

This chapter covers the following topics:

Functional Overview
VALIDATE: Validate Data Storage and Associator
Example of ADAVAL Output
JCL/JCS Requirements and Examples

172 Functional Overview

The ADAVAL utility validates any or all files within an Adabas database except the checkpoint and security files.

ADAVAL compares the actual descriptor values contained in the records in Data Storage with the corresponding values stored in the Associator to ensure that the Associator and Data Storage are synchronized, and that there are no values missing from the Associator.

Before running ADAVAL, the consistency of the inverted lists should be checked with the ADAICK utility.

Note: If ADAICK has been run and errors occurred, *do not* run ADAVAL until the cause of the ADAICK error has been corrected. This ADAVAL run restriction applies for any ADAICK error except ADAICK WARNING-163.

The Adabas nucleus must be running when executing ADAVAL. ADAVAL assigns EXF (exclusive use) status to all files to be validated, making them unavailable to other utilities or users. If ADAVAL specifies a file currently in use, an error message is issued and operation stops. ADAVAL returns condition code 4 if any errors are found.

ADAVAL prints a list of all fields compared and the ISNs rejected during validation on SYSOUT (DD/DRUCK). The normal ADAVAL output is shown under **Example of ADAVAL Output**.

If desired, rejected ISNs can also be output to a sequential data set (DD/FEHL). The first record on DD/FEHL is always as follows:

Bytes	Description
0-1	Record length in binary format (example: X'0012')
2-3	Set to zero (example: X'0000')
4-9	Program ID (example: C'ADAVAL')
10-13	Four-byte packed Julian date in format, *YYYYDDDF* ("F" = B'1111')
14-17	Four-byte packed time in format, *hhmmssth* (*t* = tenths of a second, *h* = hundredths of a second)

All remaining DD/FEHL records have the following format (items shown with an asterisk (*) are also in the normal SYSOUT and DD/DRUCK output):

Bytes	Description
0-1	Record length in binary format (example: X'0012')
2-3	Set to zero (example: X'0000')
4-5*	Adabas file number in binary format
6*	Flag byte: C'-' A value is missing C'+' A value is incorrect
7	Set to zero
8-11	ISN in binary format
12-13*	Descriptor name as stored in the field definition table (FDT)
14*	Descriptor value length in binary format
15, on*	Descriptor value

173 VALIDATE: Validate Data Storage and Associator

- Essential Parameters 1104
- Optional Parameters 1105

```
ADAVAL  VALIDATE  FILE=file-list
                  SORTSIZE= size
                  TEMPSIZE= size
                  [CODE = cipher-key ]
                  [DESCRIPTOR = 'descriptor-list ' ]
                  [LPB = { prefetch-buffer-length  |  based-on-ADARUN-lu } ]
                  [LRECL = { record-buffer-length  |  4000 } ]
                  [LWP = {work-pool-size  |  1048576 } ]
                  [NOUSERABEND]
                  [PASSWORD =  password ]
                  [SORTDEV = {device-type  |  ADARUN-device } ]
                  [TEMPDEV = {device-type  |  ADARUN-device } ]
```

The VALIDATE function validates the contents of the Data Storage against the values in the Associator. This is done by issuing commands to create a DVT that is validated against each corresponding value in the indices.

Note: ADAVAL VALIDATE cannot be performed on the checkpoint or security files.

Essential Parameters

FILE: Files to Be Validated

FILE specifies a one or more Adabas file numbers and/or file ranges. A maximum of 1000 files may be specified.

Continuation for a file list is as follows:

```
ADAVAL  VALIDATE  FILE=1-10,15
ADAVAL            FILE=13,31-35
```

ADAVAL will concatenate the file list for each specification of the FILE parameter.

SORTSIZE: Sort Area Size

SORTSIZE specifies the number of blocks or cylinders available for the sort data set. If specifying blocks, the value must be followed by a "B" (for example, "2000B"). A block value is automatically rounded up to the next full cylinder. See the *Adabas DBA Reference* documentation for information about estimating the SORTSIZE value.

TEMPSIZE: Temporary Storage Area Size

TEMPSIZE specifies the number of blocks or cylinders available for the temp data set. If specifying blocks, the value must be followed by a "B" (for example, "2000B"). A block value is

automatically rounded up to the next full cylinder. See the section *LOAD File Space Allocation* in the *ADALOD* description for information about estimating the TEMPSIZE value.

Optional Parameters

CODE: Cipher Code

The CODE parameter is required if the file or file(s) being validated are enciphered.

DESCRIPTOR: List of Descriptors to Validate

The DESCRIPTOR parameter restricts validation processing to one descriptor field, providing a way to limit the validation run in cases where that the Associator is very large or there is a need to evaluate a specific descriptor. If DESCRIPTOR is not specified, ADAVAL validates all qualifying descriptor fields.

The following is an example of DESCRIPTOR use:

```
ADAVAL  VALIDATE
FILE=5,DESCRIPTOR='AA,CC,BB'
```

LPB: Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the current ADARUN LU value. ADAVAL VALIDATE may reduce the LPB value below that specified if the LU value is too small.

LRECL: Maximum Descriptor Value

LRECL specifies the maximum length of all descriptor values in any record of the file being validated. This length is used by ADAVAL to create a temporary record buffer. If the LRECL value is too small, response code 53 occurs when an oversized record is found. The default for LRECL is 4000 bytes; the maximum length allowed is 32760 bytes.

LWP: Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADAVAL run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the Sort data set, a smaller LWP value should be specified when validating relatively small files.

The minimum work pool size depends on the Sort data set's device type:

Sort Device	Minimum LWP	Minimum LWP
	Bytes	Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

NOUSERABEND: Termination without Abend

When an error is encountered while the function is running, the utility prints an error message and terminates with user abend 34 (with a dump) or user abend 35 (without a dump).

If NOUSERABEND is specified, the utility will *not* abend after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD: Files Password

This parameter is required if the file or file(s) to be validated are password-protected.

SORTDEV: Sort Device Type

The SORTDEV parameter indicates the device type to be used for the sort data set that ADAVAL uses to sort descriptor values (the sort data set size is specified with SORTSIZE). This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEMPDEV: Temporary Storage Device Type

The TEMPDEV parameter indicates the device type to be used for the temp data set that ADAVAL uses to store intermediate data. The data set size is specified with the TEMPSIZE parameter. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

174 Example of ADAVAL Output

ADAVAL output provides a SYSOUT (DD/DRUCK) table listing, by file and descriptor, of all data storage and Associator entries and their status. The following is an example of ADAVAL VALIDATE output:

```
FILE  DE  F      ISN    DE-VALUE
-------------------------------------------------------------
  1   AA    *** NO INCONSISTENCIES ***
  1   BA  -        35  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       173  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       471  07C6D935 C5D4C1D5          *.FREEMAN*

  1   BA  -       534  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       597  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       622  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       658  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       717  07C6D935 C5D4C1D5          *.FREEMAN*
  1   BA  -       152  05D4C5E8 C5D9              *.MEYER*
  1   BA  +       153  05D4C5E8 C5D9              *.MEYER*
  1   BB    *** NO INCONSISTENCIES ***
  1   CA    *** NO INCONSISTENCIES ***
  1   CB    *** NO INCONSISTENCIES ***
  1   CC    *** NO INCONSISTENCIES ***
  1   CD    *** NO INCONSISTENCIES ***
  1   PA    *** NO INCONSISTENCIES ***
```

where

- In the F (flag) column, a dash (--) indicates that an inverted list entry is missing for the specified Data Storage descriptor; and a plus symbol (+) indicates that the inverted list entry in the Associator is incorrect.
- The DE-VALUE column provides the compressed descriptor value, first in hexadecimal and then in alphanumeric.

Note: The "*** NO INCONSISTENCIES ***" entry occurs for every successful descriptor validation.

175 JCL/JCS Requirements and Examples

- ASSO-, DATA-, and Work Data Sets 1110
- Collation with User Exit 1110
- Sorting Large Files 1111
- BS2000 1111
- z/OS 1112
- z/VM 1113
- VSE 1114

This section describes the job control information required to run ADAVAL with BS2000, z/OS, z/VM, and VSE systems and shows examples of each of the job streams.

ASSO-, DATA-, and Work Data Sets

The ASSO, DATA, and Work data sets need not be specified if Adabas is run in multiuser mode (ADARUN MODE=MULTI), because they are not opened by ADAVAL. ADAVAL receives the information concerning the database by special Adabas commands when the database is active. However, if the database is not active, ADAVAL will have problems.

However, if Adabas is run in single user mode (ADARUN MODE=SINGLE), the ASSO, DATA, and Work data sets must be specified.

Collation with User Exit

If a collation user exit is to be used during ADAVAL execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding subsystem (UES), the format of the collation descriptor user exit parameter is:

```
ADARUN CDXnn= exit-name
```

where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01-08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference* documentation for more information.

Sorting Large Files

When sorting large files, performance can be improved if either the sort data set occupies two volumes or two sort data sets are specified. Both data sets must be on the same device type (SORTDEV parameter), and each must be exactly half the size specified by the SORTSIZE parameter.

BS2000

Data Set	Link Name	Storage	More Information
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	Split the sort area across two volumes when using large files (see note)
Temp area	DDTEMPR1	disk	
ADARUN parameters	SYSDTA/DDCARD		*Operations*
ADAVAL parameters	SYSDTA/DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAVAL messages	SYSLST/ DDDRUCK		*Messages and Codes*
Rejected data	DDFEHL	tape/disk	

ADAVAL JCL Example (BS2000)

In SDF Format:

```
/.ADAVAL LOGON
 /MODIFY-TEST-OPTIONS DUMP=YES
 /REMARK *
 /REMARK * A D A V A L ALL FUNCTIONS
 /REMARK *
 /DELETE-FILE VAL.FEHL
 /SET-JOB-STEP
 /CREATE-FILE VAL.FEHL,PUB(SPACE=(48,48))
 /SET-JOB-STEP
 /ASS-SYSLST L.VAL
 /ASS-SYSDTA *SYSCMD
 /SET-FILE-LINK DDLIB,ADAvrs.MOD
 /SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
 /SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
 /SET-FILE-LINK DDFEHL1,VAL.FEHL
 /START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
 ADARUN PROG=ADAVAL,DB=yyyyy,IDTNAME=ADABAS5B
```

```
 ADAVAL VALIDATE FILE=30,SORTSIZE=3,TEMPSIZE=5
 /LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAVAL LOGON
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * A D A V A L ALL FUNCTIONS
 /REMARK *
 /SYSFILE SYSLST=L.VAL
 /FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
 /FILE ADAyyyyy.SORT ,LINK=DDSORTR1
 /FILE ADA.MOD,LINK=DDLIB
 /FILE VAL.FEHL,LINK=DDFEHL,SPACE=(48,48)
 /EXEC (ADARUN,ADA.MOD)
 ADARUN PROG=ADAVAL,DB=yyyyy,IDTNAME=ADABAS5B
 ADAVAL VALIDATE FILE=30,SORTSIZE=3,TEMPSIZE=5
 /LOGOFF NOSPOOL
```

z/OS

Data Set	DD Name	Storage	More Information
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	Split the sort area across two volumes when using large files (see note)
Temp area	DDTEMPR1	disk	
ADARUN parameters	DDCARD	reader	*Operations*
ADAVAL parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAVAL messages	DDDRUCK	printer	*Messages and Codes*
Rejected data	DDFEHL	tape/disk	

ADAVAL JCL Example (z/OS)

```
//ADAVAL    JOB
//*
//*   ADAVAL: VALIDATE DATA BASE
//*
//VAL      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <=== SORT
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <=== TEMP
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAVAL,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*

//DDKARTE   DD   *
ADAVAL VALIDATE FILE=1,TEMPSIZE=ttt,SORTSIZE=sss
/*
```

Refer to ADAVAL in the JOBS data set for this example.

z/VM

Data Set	DD Name	Storage	More Information
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	Split the sort area across two volumes when using large files (see note)
Temp area	DDTEMPR1	disk	
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations*
ADAVAL parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAVAL messages	DDDRUCK	disk/ terminal/ printer	
Rejected data	DDFEHL	tape/disk	

Example (z/VM)

```
DATADEF DDPRINT,DSN=ADAVAL,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAVAL.DDDRUCK,MODE=A
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDCARD,DSN=RUNVAL.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAVAL.CONTROL,MODE=A
DATADEF DDFEHL,DSN=FILE030.VALERR,MODE=A
ADARUN
```

Contents of RUNVAL CONTROL A1:

```
ADARUN PROG=ADAVAL,DEVICE=dddd,DB=yyyyy
```

Contents of ADAVAL CONTROL A1:

```
ADAVAL VALIDATE FILE=30,SORTSIZE=3,TEMPSIZE=5
```

VSE

File	File Name	Storage	Logical Unit	More Information
Sort area	SORTR1	disk		
Sort area	SORTR2	disk		When using large files, split the sort area across two volumes (see note)
Temp area	TEMPR1	disk*		
ADARUN parameters	- CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAVAL parameters	-	reader	SYSIPT	*Utilities*
ADARUN messages	-	printer	SYSLST	*Messages and Codes*
ADAVAL messages	-	printer	SYS009	*Messages and Codes*
Rejected data	FEHL	tape disk	SYS014 *	

** Any programmer logical unit can be used.*

ADAVAL JCS Example (VSE)

See **Library and File Procedures for VSE Examples** for descriptions of the VSE procedures (PROCs).

```
* $$ JOB JNM=ADAVAL,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAVAL
*       VALIDATE DATABASE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAVAL,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAVAL VALIDATE FILE=1,TEMPSIZE=ttt,SORTSIZE=sss
/*
/&
* $$ EOJ
```

Refer to member ADAVAL.X for this example

XXIII ADAWRK Utility: Work Area Recovery Reports

The ADAWRK utility can be used to produce reports from records in the autorestart area of Work part 1. This information can be used when the database autostart fails and the database will not come up. The data on the ADAWRK reports can help you determine whether:

- You should run a restore/regenerate (ADASAV RESTORE utility function followed by the ADARES REGENERATE utility function) of the database, which can be time-consuming.
- Excluding specific files from the autorestart with AREXCLUDE and restore/regenerate only these single files would be beneficial.
- The database can be quickly repaired so it can be started and functional more quickly.

This chapter covers the following topics:

Functional Overview
Utility Syntax
Report Descriptions
JCL/JCS Requirements and Examples

176 Functional Overview

When a database autostart fails and the database will not start up, you need to know what can be done to get the database back up and running quickly, with a minimum amount of lost data and with enough information to retrieve any lost updates. The ADAWRK utility can help you make this determination. It can produce the following reports:

- The Summary report is produced by default and provides an overview of the data in the autorestart area of Work part 1.
- The File report is an optional report that provides a breakdown of the data in the autorestart area of Work part 1 by file.
- The Transaction report is an optional report that provides a breakdown of the data in the autorestart area of Work part 1 by transaction.

Checkpoint records and associated data found within the autorestart area of Work part 1 can also be printed. Samples and more detailed descriptions of all reports and checkpoint records are provided elsewhere in this chapter.

The ADAWRK utility will only report on transactions that may need to be corrected as part of the autorestart processing logic.

You can filter all of the Work part 1 autorestart area records processed and reports produced in an ADAWRK run by communication ID, ETID, user ID, and file number.

For information on Adabas database autorestart processing, read *Recovery/Restart Design*, in *Adabas DBA Tasks Manual*

177 Utility Syntax

This is the syntax of the ADAWRK utility. Sample JCL using this syntax is provided elsewhere in this chapter.

```
ADAWRK  [ABEND34]
        [CHECKPOINT = {YES | NO }]
        [CMID = id1 [, id2 ] ... [, id24 ] ]
        [ETID = etid1 [, etid2 ] ... [, etid32 ] ]
        [FILES = fn1 [, fn2 ] ... [, fn64 ] ]
        [FORCE = {YES | NO }]
        [LWP = nnnK | 1024K ]
        [NOPPT]
        [NOUSERABEND]
        [REPORTFILE = {YES | NO }]
        [SUMMARY = {NO | YES }]
        [TEST]
        [TIMEZONE = {MACHINE | LOCAL | {+ | -} nn } ]
        [TRANSACTIONS = { NO | YES | DETAIL | FULL} ]
        [USERID = id1 [, id2 ] ... [, id24 ] ]
```

ADAWRK can be specified alone, without any parameters, to produce a summary report (SUMMARY=YES is the default). You can optionally customize the reports produced by ADAWRK by adding other parameter values. Each parameter is described here:

ABEND34

Use the ABEND34 parameter to change a user abend 35 to a user abend 34 when an ADAWRK utility error occurs. This ensures that a dump is produced when the utility terminates abnormally.

CHECKPOINT

Use the CHECKPOINT parameter to indicate whether or not checkpoints found in the autorestart area of Work part 1 should be printed in the ADAWRK report output. Valid values are "YES" and "NO". A value of "YES" indicates that checkpoint records found should be printed; a value of "NO" indicates that they should not. The default is "NO".

CMID

Use the CMID parameter to specify up to 24 32-byte communication IDs in hexadecimal format. Only Work part 1 autorestart area records with communication IDs equal to the values specified on the CMID parameter will be processed by the ADAWRK utility and printed on its reports.

ETID

Use the ETID parameter to specify up to 32 ETIDs in character format. ETIDs must be one to eight bytes long. When ETIDs are specified, only Work part 1 autorestart area records for those ETIDs are processed by the ADAWRK utility and printed on its reports.

FILES

Use the FILES parameter to specify up to 64 file numbers that should be included in the report. Only Work part 1 autorestart area records for files listed in the FILES parameter will be processed by the ADAWRK utility and printed on its reports. However, if the FILES parameter is not specified, all files in the database will be processed by default.

You can specify a range of file numbers for this parameter if needed. For example, FILES=2-20 indicates that all files with file numbers between and including 2 and 20 should be processed by the utility.

FORCE

Use the FORCE parameter to indicate how ADAWRK processing should proceed when inconsistencies in the autorestart area of Work part 1 are encountered. Valid values are "YES" and "NO". A value of "YES" indicates that the ADAWRK utility should continue to attempt to interpret the data, without abending; a value of "NO" indicates that inconsistencies in the autorestart area of Work part 1 will result in the termination of the utility with an appropriate message and abend.

LWP

Use the LWP parameter to specify the size of the work pool used internally by the ADAWRK utility. Valid values are in the range 100K - 1048576K (or 1 Gb). The default is 1024K (1Mb). The LWP must be specified in kilobyte units and if the "K" is not present in the specification, an error will result. For example, LWP=500K is a valid specification, but LWP=500 is not.

NOPPT

Use the NOPPT parameter to indicate that the Associator (ASSO) data set for the database should not be opened by the utility. If you specify this parameter, the ASSO data set will not

be opened. When this parameter is not specified, and if the ASSO data set is provided to the utility, the utility will use the PPT.

This parameter provides a workaround in situations where the Work data sets are available, but the Associator is not.

NOUSERABEND

Use the NOUSERABEND parameter to indicate that the utility should terminate with a return code of 20 if a user abend occurs. If you specify this parameter, a return code of 20 will be produced instead of abending. If you do not specify this parameter, the user abend will occur if necessary.

REPORTFILE

Use the REPORTFILE parameter to indicate whether or not the File Statistics report should be printed. Valid values are "YES" and "NO". "NO" indicates that the File Statistics report should *not* be printed; "YES" indicates that the File Statistics report *should* be printed.

The default is "NO".

When the CMID, FILES, USERID, or ETID parameters are specified with the REPORTFILE parameter, only updates that satisfy all of the criteria specified by all of the parameters is included in the File Statistics report.

SUMMARY

Use the SUMMARY parameter to indicate whether or not the Summary report should be printed. Valid values are "YES" and "NO". "NO" indicates that the Summary report should *not* be printed; "YES" indicates that the Summary report *should* be printed.

The default is "YES".

When the CMID, FILES, USERID, or ETID parameters are specified with the SUMMARY parameter, only updates that satisfy all of the criteria specified by all of the parameters is included in the Summary report.

TEST

Use the TEST parameter to test the validity of the parameters you have specified for the ADAWRK utility. When you specify the TEST parameter, no reports are produced.

TIMEZONE

Use the TIMEZONE parameter to convert the time values to a specific time zone prior to producing the ADAWRK reports. Valid values for the TIMEZONE parameter are LOCAL, MACHINE, and an hour difference specification in the format +|- *nn* (where *nn* is a valid value from 0 to 23).

Parameter Value	Description
LOCAL	Times are adjusted by the local adjustment value found on the machine. This is the default.
MACHINE	Times are printed as they are found.
-23 to +23	Times are adjusted backward (if a minus sign is specified) or forward (if a plus sign is specified) by the number of hours specified.

TRANSACTIONS

Use the TRANSACTIONS parameter to indicate whether or not the Transaction report should be printed. Valid values are "YES", "NO", "DETAIL", and "FULL":

Parameter Value	Description
NO	The Transaction report is *not* printed. This is the default.
YES	The Transaction report is printed.
DETAIL	Additional details about transaction data found in the autorestart area of Work part 1 are printed, including the files and ISN numbers that have been updated by each transaction.
FULL	All data storage and inverted list (descriptor value table DVT) data associated with each update in a transaction is printed.

When the CMID, FILES, USERID, or ETID parameters are specified with the TRANSACTIONS parameter, only updates that satisfy all of the criteria specified by all of the parameters is included in the Transaction report.

USERID

Use the USERID parameter to specify up to 24 user ID values that should be used to filter the Work part 1 autorestart area records processed by the utility. User IDs are stored in the last eight bytes of the communication ID in a record. Only records with communication IDs whose last eight bytes match the user IDs listed in the USERID parameter will be processed by the ADAWRK utility and printed on its reports.

User IDs must be specified as one to eight bytes long and in character or hexadecimal format. If you specify user IDs in character format, you may use an asterisk (*) as a wildcard character; user IDs specified in hexadecimal cannot use wildcards (the specification of an asterisk in a hexadecimal user ID will be converted to X'5C' in ADAWRK processing).

The asterisk wildcard character in character user IDs must be specified at the end of a user ID. For example, USERID=ABC* would cause the ADAWRK utility to process all Work part 1 autorestart area records with user ID values beginning with the letters "ABC". However, if USERID=*ABC were specified, the asterisk wildcard would be ignored and the ADAWRK utility would process all Work part 1 autorestart are records with a user ID value of "ABC".

178 Report Descriptions

- Environment Report 1126
- Summary Report 1126
- File Report 1128
- Transaction Report 1131
- Checkpoint Record Reporting 1150

This chapter describes the possible reports produced by the ADAWRK utility.

All reports are affected by the settings of the filter parameters CMID, ETID, FILES, and USERID. When these parameters are specified in an ADAWRK run, ADAWRK will only process records and produce reports from Work part 1 autorestart area records that meet the criteria specified by the filter parameters.

Environment Report

Here is a sample of the Environment report. This page is always printed -- regardless of the ADAWRK parameters specified. This report identifies the ADAWRK parameters used to produce the report as well as the Work data sets used for the report.

```
A D A W R K   V8.1  SM8   DBID = 15002  Started           2007-05-24  18:57:14



Parameters:
-----------


ADAWRK NOPPT
ADAWRK FORCE=YES
ADAWRK FILES=1-4,5,10
ADAWRK TRANSACTIONS=FULL
ADAWRK REPORTFILE=YES




Work datasets provided in JCL.
Database ID on Work is 15,002.

The following Work datasets were used:
    -    ADABAS.ADAWRK.TC1.WORK               NUCID:     0
```

Summary Report

The Summary report is triggered by the ADAWRK SUMMARY parameter. When SUMMARY=YES (the default) is specified, a Summary report is produced that provides an overview of the data in the autorestart area of Work part 1. This report consists of a number of sections, as follows:

- An overview of what was processed and any environmental information that can be gleaned from the provided input

- An overview of the files that will be modified as part of an autorestart and some details about what parts of the file may be affected
- The total number of system checkpoint records in the autorestart area of Work part 1 (checkpoints with a name starting with the letters "SYN")
- The total number of user checkpoint records in the autorestart area of Work part 1
- The total number of ET transactions that may need to be redone (including a subtotal of those that were backed out)
- For ET transactions that may need to be redone, the number of insert, update, and delete operations that were performed on the file
- For ET transactions that may need to be redone, the total number of data modifications to the file (total of inserts, updates, and deletes), the total number of data record updates, the total number of update commands with associated descriptor value table (inverted list) records, and the total number of descriptor value table (inverted list) updates
- The total number of incomplete transactions for the file
- For incomplete transactions, the number of insert, update, and delete operations that were performed on the file
- For incomplete transactions, the total number of data modifications to the file (total of inserts, updates, and deletes), the total number of data record updates, the total number of update commands with associated descriptor value table (inverted list) records, and the total number of descriptor value table (inverted list) updates

The following is a sample of a Summary report produced by ADAWRK:

```
*******************************************************************************
*                          A D A W R K  Summary Report                        *
*******************************************************************************


Work Dataset: ADABAS.ADAWRK.TC1.WORK                NUCID:     0

I Item                     I Work RABN   I      Blocks I Date/Time           I
I--------------------------I-------------I-------------I---------------------I
I LP value                 I             I       1,000 I                     I
I Last written block       I           6 I             I 2007-05-08 12:30:54 I
I Autobackout endpoint     I           4 I           2 I 2007-05-08 12:30:53 I
I Backward repair endpoint I           5 I           1 I                     I
I--------------------------I-------------I-------------I---------------------I

Summary of potential inconsistencies:
-------------------------------------
"Index Structure"....: Index structures may be inconsistent.
"Index Values".......: Index values may be incorrect.
"Address Converter"..: Address converter may be incorrect.
"Data Contents"......: Data contents may be incorrect according to application.
```

```
I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      1 I           I           I           I           I
I      2 I           I           I           I           I
I      3 I           I           I           I           I
I      4 I           I           I           I           I
I      5 I           I           I           I           I
I--------I-----------I-----------I-----------I-----------I
I     10 I           I           I           I           I
I--------I-----------I-----------I-----------I-----------I


The following checkpoints were encountered in the area of interest:

Total number of system checkpoints:      0


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       0
              Backed out...........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0

Incomplete transactions that must be backed out during autorestart processing:
         Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0
```

File Report

The File report is triggered by the ADAWRK REPORTFILE parameter. When REPORTFILE=YES is specified, a File report is produced that breaks down the data in the autorestart area of Work part 1 by file. For each file, the following information can be provided in a File report:

- Information similar to that provided in the Summary report
- The total number of ET transactions that may need to be redone (including a subtotal of those that were backed out)

- For ET transactions that may need to be redone, the number of insert, update, and delete operations that were performed on the file
- For ET transactions that may need to be redone, the total number of data modifications to the file (total of inserts, updates, and deletes), the total number of data record updates, the total number of update commands with associated descriptor value table (inverted list) records, and the total number of descriptor value table (inverted list) updates
- The total number of incomplete transactions for the file
- For incomplete transactions, the number of insert, update, and delete operations that were performed on the file
- For incomplete transactions, the total number of data modifications to the file (total of inserts, updates, and deletes), the total number of data record updates, the total number of update commands with associated descriptor value table (inverted list) records, and the total number of descriptor value table (inverted list) updates

The following is a sample of a File report:

```
*****************************************************************************
*                          A D A W R K  File Report                         *
*****************************************************************************


I---------------------------------------------------------------------------I
I                                File      1                                I
I---------------------------------------------------------------------------I

Oldest activity in Work RABN          54 at 2007-06-28 00:54:38  by NUCID 0      ↩

Latest activity in Work RABN          54 at 2007-06-28 00:54:38  by NUCID 0



I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      1 I           I           I     *     I     *     I
I--------I-----------I-----------I-----------I-----------I


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       1
              Backed out...........................:       0
        Total modification commands................:       1
              Inserts (N1).........................:       1
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       1
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0
```

```
Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).....................:       0
              Updates (A1).....................:       0
              Deletes (E1).....................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0


I-------------------------------------------------------------------------------I
I                                File      2                                    I
I-------------------------------------------------------------------------------I

Latest activity in Work RABN           0 at                      by NUCID 0
Oldest activity in Work RABN           0 at                      by NUCID 0


I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      2 I           I           I           I           I
I--------I-----------I-----------I-----------I-----------I


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       0
              Backed out.......................:       0
        Total modification commands................:       0
              Inserts (N1).....................:       0
              Updates (A1).....................:       0
              Deletes (E1).....................:       0
        Total data records modified................:       1
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0

Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).....................:       0
              Updates (A1).....................:       0
              Deletes (E1).....................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0
```

Transaction Report

The Transaction report is triggered by the ADAWRK TRANSACTIONS parameter. When TRANSACTIONS=YES, TRANSACTIONS=DETAIL, or TRANSACTIONS=FULL are specified, a Transaction report is produced that breaks down the data in the autorestart area of Work part 1 by transaction. For each transaction, the following information can be provided in a Transaction report:

- The communication ID for the transaction
- Whether the transaction was still open, committed, or backed out
- The number of inserts, updates, and deletes performed by the transaction
- The total number of modifications performed by the transaction (total of inserts, updates, and deletes)
- The total number of update commands with associated descriptor value table (inverted list) records
- Whether the ET data is associated with the transaction and any ETID is associated with the transaction
- If TRANSACTIONS=DETAIL is specified, each file updated by the transaction and the ISN on each file that was updated by the transaction
- If TRANSACTIONS=FULL is specified, the Data Storage image(s) associated with each update for the transaction and the descriptor value table (inverted list) data associated with each update for the transaction

The following is a portion of a Transaction report:

```
*****************************************************************************
*                          A D A W R K  Transaction Report                  *
*****************************************************************************


I---------------------------------------------------------------------------I
I                            New Communication ID                           I
I---------------------------------------------------------------------------I
Communication ID:  ' .~..o           8  U001U001'
                  x'0004A10E2096000040404040404040000F8EE80E4F0F0F1E4F0F0F1'


This transaction was committed (ET).
ETID: None
ET data was provided.
First protection data in Work RABN           4 written at 2007-05-08 12:30:53.
Last  protection data in Work RABN           0 written at                    .
```

```
        Total modification commands.................:       1
              Inserts (N1)......................:       1
              Updates (A1)......................:       0
              Deletes (E1)......................:       0
        Total data records modified.................:       1
        Total modifications with descriptor updates..:       0
        Total descriptor updates....................:       0

File     1  ISN           11 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000B09C1 F8C1F800 00000103  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 0A                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000001                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 0A                 'A9A9A   .'

File     1  ISN           12 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000C09C1 F8C1F800 00000203  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 09                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000002                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 09                 'A9A9A   .'

File     1  ISN           13 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000D09C1 F8C1F800 00000303  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 08                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.
```

```
Field name A8 image: AI data:
0000 C1F8C1F8 00000003                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 08                 'A9A9A   .'

File     1  ISN           14 (internal) Inserted

After Image Data Storage in Work RABN          4 written at 12:30:53
0000 00210000 000E09C1 F8C1F800 00000403  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 07                                   '.'

Related descriptor value updates in Work RABN          4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000004                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 07                 'A9A9A   .'

File     1  ISN           15 (internal) Inserted

After Image Data Storage in Work RABN          4 written at 12:30:53
0000 00210000 000F09C1 F8C1F800 00000503  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 06                                   '.'

Related descriptor value updates in Work RABN          4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000005                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 06                 'A9A9A   .'

File     1  ISN           16 (internal) Inserted

After Image Data Storage in Work RABN          4 written at 12:30:53
0000 00210000 001009C1 F8C1F800 00000603  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 05                                   '.'

Related descriptor value updates in Work RABN          4 written at 12:30:53

This record contains 2 descriptor updates.
```

```
Field name A8 image: AI data:
0000 C1F8C1F8 00000006                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 05                 'A9A9A   .'

File     1  ISN            17 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 001109C1 F8C1F800 00000703  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 04                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000007                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 04                 'A9A9A   .'

File     1  ISN            18 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 001209C1 F8C1F800 00000803  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 03                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000008                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 03                 'A9A9A   .'

File     1  ISN            19 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 001309C1 F8C1F800 00000903  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 02                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
```

```
0000 C1F8C1F8 00000009                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 02                 'A9A9A   .'

File     1  ISN           20 (internal) Inserted

After Image Data Storage in Work RABN          4 written at 12:30:53
0000 00210000 001409C1 F8C1F800 00000A03  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 01                                   '.'

Related descriptor value updates in Work RABN          4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 0000000A                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 01                 'A9A9A   .'

File     1  ISN           10 (internal) Updated

Related descriptor value updates in Work RABN          4 written at 12:30:53
0000 00210000 000A09C1 F8C1F800 00000A03  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 01                                   '.'

After Image Data Storage in Work RABN          4 written at 12:30:53
0000 00210000 000A09E4 F8E4F800 00000103  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 06                                   '.'

Related descriptor value updates in Work RABN          4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 0000000A                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 01                 'A9A9A   .'

File     1  ISN           11 (internal) Updated

Related descriptor value updates in Work RABN          4 written at 12:30:53
0000 00210000 000B09C1 F8C1F800 00000103  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 0A                                   '.'

After Image Data Storage in Work RABN          4 written at 12:30:53
```

```
0000 00210000 000B09E4 F8E4F800 00000203  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 05                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000001                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 0A                 'A9A9A   .'

File     1  ISN            12 (internal) Updated

Related descriptor value updates in Work RABN           4 written at 12:30:53
0000 00210000 000C09C1 F8C1F800 00000203  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 09                                   '.'

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000C09E4 F8E4F800 00000303  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 04                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000002                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 09                 'A9A9A   .'

File     1  ISN            13 (internal) Updated

Related descriptor value updates in Work RABN           4 written at 12:30:53
0000 00210000 000D09C1 F8C1F800 00000303  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 08                                   '.'

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000D09E4 F8E4F800 00000403  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 03                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.
```

```
Field name A8 image: BI data:
0000 C1F8C1F8 00000003                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 08                 'A9A9A   .'

File     1  ISN           14 (internal) Updated

Related descriptor value updates in Work RABN           4 written at 12:30:53
0000 00210000 000E09C1 F8C1F800 00000403  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 07                                   '.'

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000E09E4 F8E4F800 00000503  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 02                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000004                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 07                 'A9A9A   .'

File     1  ISN           15 (internal) Updated

Related descriptor value updates in Work RABN           5 written at 12:30:54
0000 00210000 000F09C1 F8C1F800 00000503  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 06                                   '.'

After Image Data Storage in Work RABN           5 written at 12:30:54
0000 00210000 000F09E4 F8E4F800 00000603  ' .   ..U8U8   ..'
0010 E4F205E4 F4E4F40A E4F9E4F9 C1000000  'U2.U4U4.U9U9A   '
0020 01                                   '.'

Related descriptor value updates in Work RABN           5 written at 12:30:54

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000005                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 06                 'A9A9A   .'

File     1  ISN           16 (internal) Deleted
```

```
Related descriptor value updates in Work RABN           5 written at 12:30:54
0000 00210000 001009C1 F8C1F800 00000603  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 05                                   '.'

Related descriptor value updates in Work RABN           5 written at 12:30:54

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000006                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 05                 'A9A9A   .'

File     1  ISN            17 (internal) Deleted

Related descriptor value updates in Work RABN           5 written at 12:30:54
0000 00210000 001109C1 F8C1F800 00000703  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 04                                   '.'

Related descriptor value updates in Work RABN           5 written at 12:30:54

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000007                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 04                 'A9A9A   .'

File     1  ISN            18 (internal) Deleted

Related descriptor value updates in Work RABN           5 written at 12:30:54
0000 00210000 001209C1 F8C1F800 00000803  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 03                                   '.'

Related descriptor value updates in Work RABN           5 written at 12:30:54

This record contains 2 descriptor updates.

Field name A8 image: BI data:
0000 C1F8C1F8 00000008                    'A8A8   .'

Field name A9 image: BI data:
0000 C1F9C1F9 C1000000 03                 'A9A9A   .'

Transaction was committed (ET)
```

```
I--------------------------------------------------------------------------------I
I                            New Communication ID                                I
I--------------------------------------------------------------------------------I
Communication ID:  ' .~..o           8  U001U001'
                  x'0004A10E2096000040404040404040000F8EE80E4F0F0F1E4F0F0F1'


This transaction was committed (ET).
ETID: None
ET data was provided.
First protection data in Work RABN           4 written at 2007-05-08 12:30:53.
Last  protection data in Work RABN           0 written at                     .

        Total modification commands..................:       0
              Inserts (N1)......................:       0
              Updates (A1)......................:       0
              Deletes (E1)......................:       0
        Total data records modified..................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates.....................:       0

File     1  ISN            11 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000B09C1 F8C1F800 00000103  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
0020 0A                                   '.'

Related descriptor value updates in Work RABN           4 written at 12:30:53

This record contains 2 descriptor updates.

Field name A8 image: AI data:
0000 C1F8C1F8 00000001                    'A8A8   .'

Field name A9 image: AI data:
0000 C1F9C1F9 C1000000 0A                 'A9A9A   .'

File     1  ISN            12 (internal) Inserted

After Image Data Storage in Work RABN           4 written at 12:30:53
0000 00210000 000C09C1 F8C1F800 00000203  ' .   ..A8A8   ..'
0010 C1F205C1 F4C1F40A C1F9C1F9 C1000000  'A2.A4A4.A9A9A   '
.
. (etc)
.
.I--------------------------------------------------------------------------------I
I                            New Communication ID                                I
I--------------------------------------------------------------------------------I
Communication ID:  ' .~..o           .T.A001    '
                  x'0005A10E2096000040404040404040000FAE380C1F0F0F100000000'
```

```
This transaction was committed (ET).
ETID: None
ET data was not provided.
First protection data in Work RABN           2 written at 2007-07-13 19:19:23
Last  protection data in Work RABN           5 written at 2007-07-13 19:19:23

        Total modification commands..................:       1
              Inserts (N1)......................:       1
              Updates (A1)......................:       0
              Deletes (E1)......................:       0
        Total data records modified..................:       4
        Total modifications with descriptor updates..:       0
        Total descriptor updates.....................:       0

--- File    34  ISN             1 (internal) Inserted
 .. This segment ISN =            1 (primary)
  . Next segment ISN =            1 (secondary)

#1: After Image Data Storage in Work RABN           2 written <19:19:23
0000 13090000 00000000 00010081 00000001  '..       . a   .'
0010 00000001 6480FCD6 C3C3E4D9 C5D5C3C5  '   ... OCCURENCE'
0020 F0F0F100 00000A40 40404040 40404040  '001   .         '
.
.
.
12F0 40404040 4040C5D5 C4D6C3C3 E4D9C5D5  '      ENDOCCUREN'
1300 C3C5F0F3 F1000000 0A                 'CE031   .'

--- File    34  ISN             1 (internal) Inserted
 .. This segment ISN =            1 (secondary)
  . Next segment ISN =            2 (secondary)

#2: After Image Data Storage in Work RABN           3 written <19:19:23
0000 138F0000 00000000 00010041 00000002  '..       . .   .'
0010 00000001 80FC4040 40404040 40C5D5C4  '   ...       END'
0020 D6C3C3E4 D9C5D5C3 C5F0F3F2 0000000A  'OCCURENCE032   .'
.
.
.
1370 40404040 40404040 404040C5 D5C411D6  '           END.O'
1380 C3C3E4D9 C5D5C3C5 F0F6F200 00000A    'OCCURENCE062   .'

--- File    34  ISN             1 (internal) Inserted
 .. This segment ISN =            2 (secondary)
  . Next segment ISN =            3 (secondary)

#3: After Image Data Storage in Work RABN           4 written <19:19:23
0000 12F70000 00000000 00020041 00000003  '.7       . .   .'
0010 00000001 75404040 40404040 40404040  '   ..            '
0020 40404040 40404040 40404040 40404040  '                 '
.
.
```

```
.
12E0 40404040 40404040 40404040 40404040  '               '
12F0 40404040 C5D5C4                      '    END'

--- File    34  ISN             1 (internal) Inserted
 .. This segment ISN =            3 (secondary)
  . Last segment

#4: After Image Data Storage in Work RABN           5 written <19:19:23
0000 05100000 00000000 00030041 00000000  '..        . .    '
0010 00000001 80FCD6C3 C3E4D9C5 D5C3C5F0  '   ...OCCURENCE0'
0020 F9F30000 000A4040 40404040 40404040  '93   .          '
.
.
.
04F0 40404040 40404040 40404040 40404040  '               '
0500 40404040 40404040 40404040 40C5D5C4  '             END'

Transaction was committed (ET).
.
.
.
(etc.)
```

The following sample of a Transaction Report shows how the report appears for an EXU user (U002 in the report).

```
*******************************************************************************
*                          A D A W R K  Transaction Report                    *
*******************************************************************************


I-----------------------------------------------------------------------------I
I                            New Communication ID                             I
I-----------------------------------------------------------------------------I

Communication ID:  '????hhhhA.Qb2.??    U001    '
                  x'777777778888888C11FD882F23857460000000E4F0F0F100000000'

This transaction was committed (ET).
ETID: None
ET data was not provided.
First protection data in Work RABN           2 written at 2007-08-30 12:32:13
Last  protection data in Work RABN           2 written at 2007-08-30 12:32:13

        Total modification commands...................:       1
              Inserts (N1)....................:       1
              Updates (A1)....................:       0
              Deletes (E1)....................:       0
        Total data records modified...................:       1
        Total modifications with descriptor updates..:       0
```

```
        Total descriptor updates.....................:       0

--- File    19  ISN         1,067 (internal) Inserted

#1: After Image Data Storage in Work RABN           2 written <12:32:13
0000 00280000 042BC802 C905C11F A2DE05C2  ' .  ..H.I.A.s?.B'
0010 E4C6C6C3 09404040 40404000 00024009  'UFFC.        . .'
0020 E4E2C1D1 C8E3E7C7                    'USAJHTXG'

Transaction was committed (ET).


I-------------------------------------------------------------------------------I
I                            New Communication ID                               I
I-------------------------------------------------------------------------------I

Communication ID:  ' .~..o           9??U001    '
                  x'0005A10E2096000040404040404040000F9B880E4F0F0F100000000'

This transaction was committed (ET).
ETID: 'U001    '
ET data was not provided.
First protection data in Work RABN           2 written at 2007-08-30 12:32:13
Last  protection data in Work RABN           2 written at 2007-08-30 12:32:13

        Total modification commands..................:      10
              Inserts (N1).....................:      10
              Updates (A1).....................:       0
              Deletes (E1).....................:       0
        Total data records modified..................:      10
        Total modifications with descriptor updates..:      10
        Total descriptor updates.....................:      10

--- File     4  ISN            11 (internal) Inserted

#2: After Image Data Storage in Work RABN           2 written <12:32:13
0000 000F0000 000B09E4 D3F0F100 000000    ' .   ..UL01    '

#3: Descriptor value updates in Work RABN           2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000000                    'UL01    '

--- File     4  ISN            12 (internal) Inserted

#4: After Image Data Storage in Work RABN           2 written <12:32:13
0000 000F0000 000C09E4 D3F0F100 000001    ' .   ..UL01   .'

#5: Descriptor value updates in Work RABN           2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000001                    'UL01   .'

--- File     4  ISN            13 (internal) Inserted
```

```
#6: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 000D09E4 D3F0F100 000002    ' .   ..UL01   .'

#7: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000002                   'UL01   .'

--- File     4  ISN           14 (internal) Inserted

#8: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 000E09E4 D3F0F100 000003    ' .   ..UL01   .'

#9: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000003                   'UL01   .'

--- File     4  ISN           15 (internal) Inserted

#10: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 000F09E4 D3F0F100 000004    ' .   ..UL01   .'

#11: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000004                   'UL01   .'

--- File     4  ISN           16 (internal) Inserted

#12: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 001009E4 D3F0F100 000005    ' .   ..UL01   .'

#13: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000005                   'UL01   .'

--- File     4  ISN           17 (internal) Inserted

#14: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 001109E4 D3F0F100 000006    ' .   ..UL01   .'

#15: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000006                   'UL01   .'

--- File     4  ISN           18 (internal) Inserted

#16: After Image Data Storage in Work RABN          2 written <12:32:13
0000 000F0000 001209E4 D3F0F100 000007    ' .   ..UL01   .'

#17: Descriptor value updates in Work RABN          2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000007                   'UL01   .'
```

```
--- File     4  ISN            19 (internal) Inserted

#18: After Image Data Storage in Work RABN           2 written <12:32:13
0000 000F0000 001309E4 D3F0F100 000008    ' .   ..UL01   .'

#19: Descriptor value updates in Work RABN           2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000008                   'UL01   .'

--- File     4  ISN            20 (internal) Inserted

#20: After Image Data Storage in Work RABN           2 written <12:32:13
0000 000F0000 001409E4 D3F0F100 000009    ' .   ..UL01   .'

#21: Descriptor value updates in Work RABN           2 written <12:32:13
  Value for descriptor A8 inserted:
0000 E4D3F0F1 00000009                   'UL01   .'

Transaction was committed (ET).


I-------------------------------------------------------------------------------I
I                            New Communication ID                               I
I-------------------------------------------------------------------------------I

Communication ID:  ' .~..o           9??U002    '
                 x'0005A10E2096000040404040404040000F9B880E4F0F0F200000000'

These updates were performed by an EXU user.
First protection data in Work RABN           1 written at 2007-08-30 12:32:08
Last  protection data in Work RABN           1 written at 2007-08-30 12:32:08

        Total modification commands..................:      10
              Inserts (N1)......................:      10
              Updates (A1)......................:       0
              Deletes (E1)......................:       0
        Total data records modified..................:      10
        Total modifications with descriptor updates..:      10
        Total descriptor updates.....................:      10

--- File     5  ISN             1 (internal) Inserted

#22: After Image Data Storage in Work RABN           1 written <12:32:08
0000 000F0000 000109E4 F0F0F200 000000    ' .   ..U002    '

#23: Descriptor value updates in Work RABN           1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000000                   'U002    '

--- File     5  ISN             2 (internal) Inserted
```

```
#24: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000209E4 F0F0F200 000001    ' .   ..U002   .'

#25: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000001                   'U002   .'

--- File     5  ISN             3 (internal) Inserted

#26: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000309E4 F0F0F200 000002    ' .   ..U002   .'

#27: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000002                   'U002   .'

--- File     5  ISN             4 (internal) Inserted

#28: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000409E4 F0F0F200 000003    ' .   ..U002   .'

#29: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000003                   'U002   .'

--- File     5  ISN             5 (internal) Inserted

#30: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000509E4 F0F0F200 000004    ' .   ..U002   .'

#31: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000004                   'U002   .'

--- File     5  ISN             6 (internal) Inserted

#32: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000609E4 F0F0F200 000005    ' .   ..U002   .'

#33: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000005                   'U002   .'

--- File     5  ISN             7 (internal) Inserted

#34: After Image Data Storage in Work RABN          1 written <12:32:08
0000 000F0000 000709E4 F0F0F200 000006    ' .   ..U002   .'

#35: Descriptor value updates in Work RABN          1 written <12:32:08
  Value for descriptor A8 inserted:
0000 E4F0F0F2 00000006                   'U002   .'
```

```
 --- File     5  ISN             8 (internal) Inserted

 #36: After Image Data Storage in Work RABN           1 written <12:32:08
 0000 000F0000 000809E4 F0F0F200 000007    ' .   ..U002   .'

 #37: Descriptor value updates in Work RABN           1 written <12:32:08
   Value for descriptor A8 inserted:
 0000 E4F0F0F2 00000007                   'U002   .'

 --- File     5  ISN             9 (internal) Inserted

 #38: After Image Data Storage in Work RABN           1 written <12:32:08
 0000 000F0000 000909E4 F0F0F200 000008    ' .   ..U002   .'

 #39: Descriptor value updates in Work RABN           1 written <12:32:08
   Value for descriptor A8 inserted:
 0000 E4F0F0F2 00000008                   'U002   .'

 --- File     5  ISN            10 (internal) Inserted

 #40: After Image Data Storage in Work RABN           1 written <12:32:08
 0000 000F0000 000A09E4 F0F0F200 000009    ' .   ..U002   .'

 #41: Descriptor value updates in Work RABN           1 written <12:32:08
   Value for descriptor A8 inserted:
 0000 E4F0F0F2 00000009                   'U002   .'

1
 Summary of communication IDs with recovery data:
 ------------------------------------------------

 Communication ID:  '????hhhhA.Qb2.??    U001    '
                   x'7777777788888888C11FD882F238574600000000E4F0F0F100000000'

 Communication ID:  ' .~..o           9??U002    '
                   x'0005A10E2096000040404040404040000F9B880E4F0F0F200000000'

 Communication ID:  ' .~..o           9??U001    '
                   x'0005A10E2096000040404040404040000F9B880E4F0F0F100000000'
1
 ********************************************************************************
 *                          A D A W R K  Summary Report                         *
 ********************************************************************************


 Work Dataset: RD.DB035.WORKR1

 I Item                     I   Work RABN I      Blocks I Date/Time           I
 I--------------------------I-------------I-------------I---------------------I
 I LP value                 I             I         800 I                     I
 I Last written block       I           2 I             I 2007-08-30 12:32:13 I
 I Autobackout endpoint     I           1 I           2 I 2007-08-30 12:32:08 I
```

```
I Backward repair endpoint I           1 I           2 I 2007-08-30 12:32:08 I
I--------------------------I------------I-------------I---------------------I


Summary of potential inconsistencies:
-------------------------------------
"Index Structure"....: Index structures may be inconsistent.
"Index Values".......: Index values may be incorrect.
"Address Converter"..: Address converter may be incorrect.
"Data Contents"......: Data contents may be incorrect according to application.

I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      4 I           I           I           I     *     I
I      5 I     *     I     *     I     *     I     *     I
I     19 I           I           I           I     *     I
I--------I-----------I-----------I-----------I-----------I


The following checkpoints were encountered in the area of interest:

Total number of system checkpoints:      5


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       3
              Backed out...........................:       0
        Total modification commands................:      21
              Inserts (N1).........................:      21
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:      21
        Total modifications with descriptor updates..:      20
        Total descriptor updates...................:      20

Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0
1
*******************************************************************************
*                          A D A W R K  File Report                           *
*******************************************************************************


I-------------------------------------------------------------------------------I
```

```
I                                 File      4                                 I
I-------------------------------------------------------------------------------I

Oldest activity in Work RABN           2 at 2007-08-30 12:32:13
Latest activity in Work RABN           2 at 2007-08-30 12:32:13


I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      4 I           I           I           I     *     I
I--------I-----------I-----------I-----------I-----------I


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       1
              Backed out...........................:       0
        Total modification commands................:      10
              Inserts (N1).........................:      10
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:      10
        Total modifications with descriptor updates..:      10
        Total descriptor updates...................:      10

Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0


I-------------------------------------------------------------------------------I
I                                 File      5                                 I
I-------------------------------------------------------------------------------I

Oldest activity in Work RABN           1 at 2007-08-30 12:32:08
Latest activity in Work RABN           1 at 2007-08-30 12:32:08


I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I      5 I     *     I     *     I     *     I     *     I
I--------I-----------I-----------I-----------I-----------I


Completed transactions that will be re-done during autorestart processing:
```

```
        Total transactions.........................:       1
              Backed out...........................:       0
        Total modification commands................:      10
              Inserts (N1).........................:      10
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:      10
        Total modifications with descriptor updates..:      10
        Total descriptor updates...................:      10

Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       0
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0


I-------------------------------------------------------------------------------I
I                                File     19                                    I
I-------------------------------------------------------------------------------I

Oldest activity in Work RABN           2 at 2007-08-30 12:32:13
Latest activity in Work RABN           2 at 2007-08-30 12:32:13


I  File  I   Index   I  Index    I  Address  I   Data    I
I Number I Structure I  Values   I Converter I Contents  I
I--------I-----------I-----------I-----------I-----------I
I     19 I           I           I           I     *     I
I--------I-----------I-----------I-----------I-----------I


Completed transactions that will be re-done during autorestart processing:
        Total transactions.........................:       1
              Backed out...........................:       0
        Total modification commands................:       1
              Inserts (N1).........................:       1
              Updates (A1).........................:       0
              Deletes (E1).........................:       0
        Total data records modified................:       1
        Total modifications with descriptor updates..:       0
        Total descriptor updates...................:       0

Incomplete transactions that must be backed out during autorestart processing:
        Total transactions.........................:       0
        Total modification commands................:       0
              Inserts (N1).........................:       0
              Updates (A1).........................:       0
```

```
              Deletes (E1).....................:       0
         Total data records modified..................:       0
         Total modifications with descriptor updates..:       0
         Total descriptor updates.....................:       0


 A D A W R K  Terminated normally                          2007-08-30  12:33:39
```

Checkpoint Record Reporting

Checkpoint records and associated data found within the autorestart area of Work part 1 can be printed by the ADAWRK utility and are triggered by the CHECKPOINT parameter. If CHECKPOINT=YES is specified, checkpoint records such as the following are printed in the output. If a Transaction report is also requested, these checkpoint records are interspersed within the Transaction report.

```
A D A W R K   V8.1  SM8   DBID = 15003  Started           2007-01-10  18:34:01
                                                                              ↩

Work RABN           1 written at hh:mm:ss  System Checkpoint
 written by jobname='cccccccc'
 Checkpoint Name=nnnn Type=tt xxxxxxxxx
                                                                              ↩

Work RABN           1 written at hh:mm:ss  System Checkpoint
 written by jobname='cccccccc'
 Checkpoint Name=nnnn Type=tt xxxxxxxxx
                                                                              ↩

Work RABN           1 written at hh:mm:ss  System Checkpoint
 written by jobname='cccccccc'
 Checkpoint Name=nnnn Type=tt xxxxxxxxx
```

179 JCL/JCS Requirements and Examples

- z/OS 1152
- VSE 1154
- z/VM 1155
- BS2000 1156

This section describes the job control information required to run ADAWRK on z/OS systems and shows examples of z/OS job streams.

z/OS

Data Set	DD Name	Storage	More Information
Associator	DDASSOR*n*	disk	Not required if the NOPPT parameter is specified
Work	DDWORKR*n*	disk	Multiple work data sets must be provided for a cluster nucleus
ADARUN parameters	DDCARD	reader	*Operations*
ADAWRK parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAWRK messages	DDDRUCK	printer	*Messages and Codes*

ADAWRK JCL Examples (z/OS)

The following example produces a Summary report. Refer to member ADAWRKJ in the JOBS data set for this example.

```
//ADAWRK    JOB
//*
//*   ADAWRK: ALL FUNCTIONS
//*
//WRK      EXEC  PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAWRK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAWRK    SUMMARY=YES
/*
```

The following example will produce a Summary and Transaction report. The Transaction report will provide a full report of all of the transactions that are on the Work data sets provided. Refer to member ADAWRKT in the JOBS data set for this example.

```
//ADAWRK    JOB
//*
//*   ADAWRK: PRINT ALL RELEVANT TRANSACTIONS ON WORK
//*
//WRK      EXEC  PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAWRK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAWRK    TRANSACTIONS=FULL
/*
```

The following example produces a Summary, Transaction, and File report, including a full report of all transactions that are on the Work data sets. However only Work data set records for files 1, 2, 3, 4, 5, and 10 will be processed and reported by the utility. In addition, note the absence of an Associator data set in the sample JCL -- this is due to the NOPPT parameter which specifies that no PPT should be used for the run, and therefore no Associator data set is required.

```
//ADAWRK    JOB
//*
//*   ADAWRK: PRINT ALL RELEVANT TRANSACTIONS ON WORK
//*
//WRK      EXEC  PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD      <=== ADABAS LOAD
//*
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAWRK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
 ADAWRK NOPPT
 ADAWRK FORCE=YES
 ADAWRK FILES=1-4,5,10
 ADAWRK TRANSACTIONS=FULL
 ADAWRK REPORTFILE=YES
/*
```

VSE

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSOR *n*	disk	*	Not required if the NOPPT parameter is specified
Work	WORKR*n*	disk	*	Multiple work data sets must be provided for a cluster nucleus
ADARUN parameters		reader tape disk	SYSRDR SYS000 *	*Operations*
ADAWRK parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADAWRK messages		printer	SYS009	*Messages and Codes*

** Any programmer logical unit may be used.*

ADAWRK JCL Examples (VSE)

The following example produces a Summary report. Refer to member ADAWRKJ.X for this example.

```
* $$ JOB JNM=ADAWRKJ,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAWRKJ     EXECUTE THE ADABAS UTILITY ***WRK***
// OPTION LOG,PARTDUMP
*
* ********************************************************************
*       SAMPLE JOB STREAM TO USE THE ADABAS UTILITY ADAWRK
*       PRINT SUMMARY REPORT
* ********************************************************************
// EXEC PROC=ADAVVLIB                                          <======
// EXEC PROC=ADAVVFIL                                          <======
*
* ********************************************************************
*     DON'T FORGET TO CUSTOMIZE PARAMETERS OF ADABAS UTILITY
* ********************************************************************
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAWRK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy   <======
/*
ADAWRK    SUMMARY=YES                                           <======
/*
/&
* $$ EOJ
```

The following example will produce a full Transaction report. The Transaction report will provide a full report of all of the transactions that are on the Work data sets provided. Refer to member ADAWRKT.X for this example.

```
* $$ JOB JNM=ADAWRKT,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAWRKT     EXECUTE THE ADABAS UTILITY ***WRK***
// OPTION LOG,PARTDUMP
*
* ****************************************************************
*       SAMPLE JOB STREAM TO USE THE ADABAS UTILITY ADAWRK
*       PRINT ALL RELEVANT TRANSACTIONS ON WORK
* ****************************************************************
// EXEC PROC=ADAVVLIB                                          <======
// EXEC PROC=ADAVVFIL                                          <======
*
* ****************************************************************
*     DON'T FORGET TO CUSTOMIZE PARAMETERS OF ADABAS UTILITY
* ****************************************************************
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAWRK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy   <======
/*
ADAWRK    TRANSACTIONS=FULL                                     <======
/*
/&
* $$ EOJ
```

z/VM

File	Symbolic Name	Storage	More Information
Associator	DDASSOR *n*	disk	Not required if the NOPPT parameter is specified
Work	DDWORKR*n*	disk	
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations*
ADAWRK parameters	DDKARTE	disk/ terminal/ reader	
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADAWRK messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*

ADAWRK JCL Examples (z/VM)

The following example produces an ADAWRK Summary report. Refer to member ADAWRK supplied on the distribution tape for this example.

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADAWRK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADAWRK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNWRK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAWRK.CONTROL,MODE=A
ADARUN
```

Based on this sample JCL, the ADARUN parameters used for the Summary report are stored in the RUNWRK.CONTROL data set (DDCARD symbolic name). Refer to member RUNWRK supplied on the distribution tape for an example:

```
ADARUN  PROG=ADAWRK,DEVICE=dddd,DBID=yyyyy
```

Based on this sample JCL, the ADAWRK parameters used for the Summary report are stored in the ADAWRK.CONTROL data set (DDKARTE symbolic name). Refer to member WRKSUMM on the distribution tape for an example:

```
ADAWRK  SUMMARY=YES
```

Refer to member WRKTRNS on the distribution tape for an example of the ADAWRK parameters used for a Transaction report.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Work	DDWORKR1 DDWORKR4	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations*
ADALOD parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		ADALOD report, see also *Messages and Codes*
ADALOD messages	SYSLST/ DDDRUCK		*Messages and Codes*

ADAWRK JCL Examples (BS2000)

```
/BEGIN-PROC C
/MOD-TEST DUMP=YES
/REMARK *
/REMARK *    ADAWRK: ALL FUNCTIONS
/REMARK *
/ASS-SYSLST L.WRKJ
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK   DDLIB,ADABAS.MOD
/SET-FILE-LINK    DDASSOR1,ADA99.ASSO
/SET-FILE-LINK    DDWORKR1,ADA99.WORK
/START-PROG *M(E=ADARUN,L=ADABAS.MOD),RUN-MODE=ADV(A-L=YES)
ADARUN PROG=ADAWRK,DB=99,MODE=MULTI
ADAWRK    SUMMARY=YES
/ASS-SYSDTA *PRIM
/ASS-SYSLST *PRIM
/END-PROC
```

XXIV ADAZAP Utility: Display or Modify Asso, Data, and Work Data Sets

This chapter covers the following topics:

Functional Overview
ADAZAP Syntax
JCL/JCS Requirements and Examples

180 Functional Overview

The ADAZAP utility can be executed only when the Adabas nucleus is inactive.

ADAZAP is used to display in hexadecimal format and optionally to change the contents of the ASSO, DATA, or WORK data sets.

Because a significant element of risk is involved in modifying physical database blocks, the use of this utility is restricted. Software AG will provide the mastercode necessary to run the ADAZAP utility only on written request to the individuals at each customer site who are authorized to receive it.

In addition, Software AG strongly recommends that you use your external security system to protect ADAZAP just as you protect other ZAP programs.

Software AG also recommends that a current save tape be available before running ADAZAP. If an error occurs during ADAZAP execution, it may be necessary to restore the affected file or database.

If the data is successfully altered, a SYNP 3F checkpoint record is written containing the REP and VER data to provide an audit trail of any changes that have been made.

A version of ADAZAP running with different syntax was unofficially distributed with previous releases of Adabas. No documentation was or is provided for this earlier version and it was and is not supported.

181 ADAZAP Syntax

- Essential Parameters 1164
- Optional Parameters 1164
- Examples 1165

```
ADAZAP  MCODE = master-code
        { ASSO | DATA | WORK }
        [LENGTH = length-of-data ]
        [OFFSET = { offset-from-RABN-start | 0 } ]
        [RABN = { rabn-number | 1 } ]
        [REP = replace-data ]
        [VER = verify-data ]
```

This chapter describes the syntax and parameters of the ADAZAP utility.

Essential Parameters

MCODE
For security purposes, a mastercode is required to run the ADAZAP utility. Software AG provides the 8-byte mastercode on written request to authorized individuals.

ASSO | DATA | WORK
It is necessary to specify the physical data set you wish to display or modify.

Optional Parameters

LENGTH
The length of the data to be displayed. LENGTH cannot be specified if VER is specified, and the reverse.

The minimum number of bytes displayed is 16 since the lower address is rounded down to a 16-byte boundary and the upper address is rounded up to a 16-byte boundary.

OFFSET
This is the offset from the start of the block. The value must be smaller than or equal to the length of a block; that is, it must fall within the block. The default value is zero.

RABN
The relative Adabas block number (RABN) that is to be displayed or altered. The default is '1'.

REP
The replace data, which must be less than or equal to the verify data specified in the VER parameter. Up to 128 bytes of hexadecimal data may be specified.

VER

The verify data, which must be at least as long as the replace data. Up to 128 bytes of hexadecimal data may be specified.

Examples

Example 1:

```
ADAZAP MCODE=master-code
       ADAZAP ASSO OFFSET=X'10',LENGTH=16
```

The default RABN=1 is used. ADAZAP displays the database name.

Example 2:

```
ADAZAP MCODE=master-code
ADAZAP WORK OFFSET=X'10'
ADAZAP      VER=X'C1C2'
ADAZAP      REP=X'C2C1'
```

The default RABN=1 is used. ADAZAP alters data in the Work data set.

182 JCL/JCS Requirements and Examples

- BS2000 1168
- z/OS 1169
- z/VM 1170
- VSE 1171

Below are sample jobs to use the ADAZAP utility. They can be used to change the contents of a specific Adabas RABN in DATA, ASSO, or WORK.

- Specify the RABN, the offset, and the values to be replaced in hexadecimal.
- To obtain the master password, contact your local support center.

Important: This utility must be used carefully. Any misuse may lead to serious problems.

BS2000

Data Set	Link Name	Storage	More Information
Associator	DDASSOR*n*	disk	required if ASSO is being zapped
Data Storage	DDDATAR*n*	disk	required if DATA is being zapped
Work	DDWORKR*n*	disk	required if WORK is being zapped
ADARUN parameters	DDCARD	reader	*Operations*
ADAZAP parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAZAP messages	DDDRUCK	printer	*Messages and Codes*

ADAZAP JCL Example (BS2000)

In SDF Format:

```
/BEGIN-PROC A
/REMA
/REMA SAMPLE JCL FOR ADAZAP
/REMA
/ASS-SYSOUT L.ADAZAP.OUT
/ASS-SYSLST L.ADAZAP.LST
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADABAS.ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,DByyyyyy.ASSOR1,SUP=DISK(SHARE-UPD=YES)
/SET-FILE-LINK DDDATAR1,DByyyyyy.DATAR1,SUP=DISK(SHARE-UPD=YES)
/SET-FILE-LINK DDWORKR1,DByyyyyy.WORKR1,SUP=DISK(SHARE-UPD=YES)
/STA-PROG *M(ADABAS.ADAvrs.MOD,ADARUN),RUN-MODE=*ADV(ALT=Y)
ADARUN PROG=ADAZAP,DB=yyyyyy
ADAZAP MCODE=xxxxxxxxxx                        <<--- MASTER PASSWORD
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16     <=== DISPLAY ASSO RABN 1
ADAZAP ASSO
RABN=1,OFFSET=X'10',VER=X'C1C2',REP=X'C2C1'
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16     <=== DISPLAY ASSO RABN 1
/SET-JOB-STEP
```

```
/ASS-SYSDTA *PRIM
/ASS-SYSLST *PRIM
/ASS-SYSOUT *PRIM
/END-PROC
```

In ISP Format:

```
/.ADAZAP PROC
 /OPTION MSG=FH,DUMP=YES
 /REMARK *
 /REMARK * SAMPLE JCL FOR ADAZAP
 /REMARK *
 /SYSFILE SYSLST=L.ZAP
 /SYSFILE SYSDTA=(SYSCMD)
 /FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
 /FILE ADAyyyyy.SORT ,LINK=DDSORTR1
 /FILE ADAyyyyy.ASSO ,LINK=DDASSOR1
 /EXEC (ADARUN,ADA.MOD)
 ADAZAP MCODE=xxxxxxxxxx <<    MASTER PASSWORD
 ADAZAP ASSO RABN=1,OFFSET=X 10 ,LENGTH=16 <===DISPLAY ASSO RABN 1
 ADAZAP ASSO RABN=1,OFFSET=X 10 ,VER=X C1C2 ,REP=X C2C1
 ADAZAP ASSO RABN=1,OFFSET=X 10 ,LENGTH=16 <===DISPLAY ASSO RABN 1
 /STEP
 /SYSFILE SYSDTA=()
 /SYSFILE SYSLST=()
 /ENDP
```

z/OS

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	required if ASSO is being zapped
Data Storage	DDDATARn	disk	required if DATA is being zapped
Work	DDWORKRn	disk	required if WORK is being zapped
ADARUN parameters	DDCARD	reader	*Operations*
ADAZAP parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAZAP messages	DDDRUCK	printer	*Messages and Codes*

Example (z/OS)

```
//ADAZAP    JOB
//*
//ZAP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.ADAvrs.LOAD       <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAZAP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
//DDKARTE  DD   *
  ADAZAP MCODE=mmmmmmmm                              <<--- MASTER PASSWORD        ↩

  ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16          <=== DISPLAY ASSO RABN 1
  ADAZAP ASSO
RABN=1,OFFSET=X'10',VER='C1C2',REP=X'C2C1'
  ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16          <=== DISPLAY ASSO RABN 1
/*
```

Refer to ADAZAP in the JOBS data set for this example.

z/VM

Data Set	Link Name	Storage	More Information
Associator	DDASSORn	disk	required if ASSO is being zapped
Data Storage	DDDATARn	disk	required if DATA is being zapped
Work	DDWORKRn	disk	required if WORK is being zapped
ADARUN parameters	DDCARD	reader	*Operations*
ADAZAP parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAZAP messages	DDDRUCK	printer	*Messages and Codes*

Example (z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1    <=== ASSO
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1    <=== DATA
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1    <=== WORK
DATADEF DDPRINT,DSN=ADAZAP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADAZAP.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNZAP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAZAP.CONTROL,MODE=A
ADARUN
```

Contents of RUNREP CONTROL A1:

```
ADARUN PROG=ADAZAP,DEVICE=dddd:
```

Contents of ADAREP CONTROL A1:

```
ADAZAP MCODE=mmmmmmmm                                  <<--- MASTER PASSWORD
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16                <===DISPLAY ASSO RABN 1
ADAZAP ASSO RABN=1,OFFSET=X'10',VER='C1C2',REP=X'C2C1'
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16                <===DISPLAY ASSO RABN 1    ↩
```

VSE

Data Set	Link Name	Storage	More Information
Associator	ASSOR*n*	disk	required if ASSO is being zapped
Data Storage	DATAR*n*	disk	required if DATA is being zapped
Work	WORKR*n*	disk	required if WORK is being zapped
ADARUN parameters	CARD	reader	*Operations*
ADAZAP parameters	KARTE	reader	
ADARUN messages	PRINT	printer	*Messages and Codes*
ADAZAP messages	DRUCK	printer	*Messages and Codes*

Example (VSE)

```
* $$ JOB JNM=ADAZAP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAZAP
*
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAZAP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAZAP MCODE=MMMMMMMM                        <<--- MASTER PASSWORD
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16    <===DISPLAY ASSO RABN 1
ADAZAP ASSO
RABN=1,OFFSET=X'10',VER='C1C2',REP=X'C2C1'
ADAZAP ASSO RABN=1,OFFSET=X'10',LENGTH=16    <===DISPLAY ASSO RABN 1
/*
/&
* $$ EOJ
```

Refer to member ADAZAP.X for this example.

See *Library and File Procedures for VSE Examples* for descriptions of the VSE procedures (PROCs).

XXV AppendicesAppendicesAppendix

- A Appendices 1175
- B Appendices 1189
- C Appendix 1193

A Appendices

- Sequential File Table 1176
- Operating System Dependencies 1178

This document describes Adabas sequential files.

Sequential File Table

This section summarizes the sequential files used by the Adabas utilities. Explanations of the table heading and contents are in the text following the table.

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
ADACDC	DD/SIIN	10		x		Yes
ADACMP	DD/AUSBA DD/EBAND DD/FEHL	12 10 14	x x	x		Yes
ADACNV	DD/FILEA	10	x			
ADALOD	DD/EBAND DD/FILEA DD/ISN DD/OLD	10 12 16 14	x x	x x x	Yes	Yes Yes
ADAMER	DD/EBAND	10		x		
ADAORD	DD/FILEA	10	x	x	Yes	
ADAPLP	DD/PLOG	14		x		Yes
ADARAI	DD/OUT	10	x			
ADAREP	DD/SAVE DD/PLOG	10 11		x x		Yes Yes
ADARES	DD/BACK DD/SIAUS1 DD/SIAUS2 DD/SIIN	20 21 22 20	x x	x x		Yes Yes
ADASAV	DD/DEL1 DD/DEL2 DD/DEL3 DD/DEL4	31 32 33 34		x x x x		Yes Yes Yes Yes
	DD/DEL5 DD/DEL6 DD/DEL7 DD/DEL8	35 36 37 38		x x x x		Yes Yes Yes Yes
	DD/DUAL1 DD/DUAL2 DD/DUAL3 DD/DUAL4	21 22 23 24	x x x x			

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
	DD/DUAL5	25	x			
	DD/DUAL6	26	x			
	DD/DUAL7	27	x			
	DD/DUAL8	28	x			
	DD/FULL	30		x		Yes
	DD/PLOG	10		x		Yes
	DD/REST1	11		x		Yes
	DD/REST2	12		x		
	DD/REST3	13		x		
	DD/REST4	14		x		
	DD/REST5	15		x		
	DD/REST6	16		x		
	DD/REST7	17		x		
	DD/REST8	18		x		
	DD/SAVE1	11	x			
	DD/SAVE2	12	x			
	DD/SAVE3	13	x			
	DD/SAVE4	14	x			
	DD/SAVE5	15	x			
	DD/SAVE6	16	x			
	DD/SAVE7	17	x			
	DD/SAVE8	18	x			
ADASEL	DD/EXPA1	11	x			
	DD/EXPA2	12	x			
	DD/EXPA3	13	x			
	DD/EXPA4	14	x			
	DD/EXPA5	15	x			
	DD/EXPA6	16	x			
	DD/EXPA7	17	x			
	DD/EXPA8	18	x			
	DD/EXPA9	19	x			
	DD/EXPA10	20	x			
	DD/EXPA11	21	x			
	DD/EXPA12	22	x			
	DD/EXPA13	23	x			
	DD/EXPA14	24	x			
	DD/EXPA15	25	x			
	DD/EXPA16	26	x			
	DD/EXPA17	27	x			
	DD/EXPA18	28	x			
	DD/EXPA19	29	x			
	DD/EXPA20	30	x			

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
	DD/SIIN	10		x		Yes
ADAULD	DD/OUT1	10	x	x	Yes	Yes
	DD/OUT2	11	x	x	Yes	Yes
	DD/ISN	12	x	x	Yes	Yes
	DD/SAVE	13		x		Yes
	DD/PLOG	14				
	DD/FULL	30				
	DD/DEL1-8	31-38				
ADAVAL	DD/FEHL	14	x		Yes	

Files that are both output and input are first written and then read by the indicated program. BS2000, z/VM, z/OS, and OS-compatible files have names starting with "DD" (DDSIIN, DDFEHL, etc.); VSE file names are without "DD".

Operating System Dependencies

The following sections describe characteristics of file and device definition by operating system.

BS2000 Systems

Note: This discussion uses SPF format. In ISP format:

SPF Format	ISP Format
BUFF-LEN	BLKSIZE defined by BLKSIZE=(STD,16)
REC-FORM	RECFM
REC-SIZE	RECSIZE
SET-FILE-LINK	FILE

The LINK name by which a file is referenced is determined as follows:

- The characters DD are prefixed to the file name to form the LINK name.
- If files for which the column "Concatenation" contains "Yes" are on tape, they may be concatenated as follows: the first file is read using the indicated LINK name; at the first end-of-file, 01 is appended to the LINK name; and, if there is a /SET-FILE-LINK (in ISP format /FILE) statement for that LINK name, reading continues.
- Each subsequent end-of-file adds 1 to the LINK name, and as long as there is a /SET-FILE-LINK (in ISP format /FILE) statement for that LINK name, reading continues through a maximum of

99. For LINK names longer than six characters, the excess characters will be overlaid with the file number increment (e.g., DDEBAND becomes DDEBAN01).

- BS2000 does not support the backward reading of multivolume tape files; therefore, all volumes of the ADARES DDBACK file must be specified in the reverse order in which they were written on /SET-FILE-LINK (in ISP format /FILE) statements using the LINK names DDBACK, DDBACK01, DDBACK02, and so on.

The BUFF-LEN of a sequential file is determined as follows:

1. The BUFF-LEN is obtained from the /SET-FILE-LINK statement or the data set's catalog entry, if present.
2. If the BUFF-LEN cannot be obtained from the /SET-FILE-LINK statement and/or catalog, the value of the ADARUN QBLKSIZE parameter is used, if specified.
3. Otherwise, the BUFF-LEN depends on the device type as follows:

Tape:	32760
Disk:	32768 (BUFF-LEN=(STD,16))

The REC-SIZE and REC-FORM should be as follows:

Tape:	REC-SIZE = BUFF-LEN - 4; REC-FORM = V;
Disk:	REC-SIZE = BUFF-LEN - 20; RECFORM = V;
Input:	Obtained from the /SET-FILE-LINK statement or the data set's catalog entry.

Note: Do not specify REC-FORM, REC-SIZE, or BUFF-LEN for input data sets unless the TAPE data set contains no REC-FORM, REC-SIZE, or BUFF-LEN values in HDR2.

The SPACE parameter for primary and secondary allocations must specify a multiple of three (3) times the number of PAM blocks specified in the BUFF-LEN parameter. Otherwise, I/O errors will occur. For the default /CREATE-FILE ...,PUB(SPACE(48,48)) and /SET-FILE-LINK ...,BUFF-LEN=STD(16) (in ISP format, BLKSIZE=(STD,16), SPACE=(48,48)) is the smallest valid value.

The portions of the DDDRUCK and DDPRINT data sets already written to disk can be accessed during either a regular nucleus or utility session for reading. This includes the following BS2000 read accesses:

- SHOW-FILE
- @READ data set
- /COPY-FILE (in ISP format, /COPY)

Concatenation of Sequential Input Files for BS2000

For using more than one data set as input medium to an ADABAS utility, some operating systems (such as z/OS) provide a concatenation feature.

For BS2000 this feature is simulated by adding /SET-FILE-LINK (in ISP format, /FILE) statements with modified LINK names created from the original and a two-digit increment (ranging from 01 to 99):

```
/SET-FILE-LINK DDTEST,firstfile
/SET-FILE-LINK DDTEST01,secondfile
/SET-FILE-LINK DDTEST02,thirdfile
...
/SET-FILE-LINK DDTEST99,lastfile
```

In ISP format:

```
/FILE firstfile ,LINK=DDTEST
/FILE secondfile,LINK=DDTEST01
/FILE thirdfile ,LINK=DDTEST02
...
/FILE lastfile ,LINK=DDTEST99
```

For those original LINK names that are 7 or 8 characters long, the incremental number occupies the 7th and 8th position. For example:

```
/SET-FILE-LINK DDEBAND,firstfile
/SET-FILE-LINK DDEBAND01,lastfile
```

In ISP format:

```
/FILE firstfile ,LINK=DDEBAND
/FILE secondfile,LINK=DDEBAN01
```

When processing input files that have the concatenation option at end-of-file of one input file, a check is made to determine whether a /SET-FILE-LINK (in ISP format, /FILE) statement exists for the next data set. If none exists, the sequential GET call returns EOF; otherwise, the data set currently open is closed, and an open is tried for the next file.

Files concatenated in this way must have the same file characteristics (block size, record format and record size).

This concatenation feature applies also to files that are processed backwards. The order of the LINK names is the reverse of the creation order. For example, ADARES with DDBACK:

```
/SET-FILE-LINK DDBACK,lastfile
/SET-FILE-LINK DDBACK01,filebeforelast
/...
/SET-FILE-LINK DDBACKnn,firstfile
```

In ISP format:

```
/FILE lastfile ,LINK=DDBACK
/FILE filebeforelast,LINK=DDBACK01
/....
/FILE firstfile ,LINK=DDBACKnn
```

Note that this feature can also be used to process a multivolume file backwards, if each volume is specified with a separate /SET-FILE-LINK (in ISP format, /FILE) statement.

The following list is of LINK names/utilities with the concatenation option:

DDDELn (where n = 1-8)	ADASAV
DDEBAND	ADACMP ADALOD ADAMER
DDFULL	ADASAV
DDISN	ADALOD
DDPLOG	ADAPLP ADASAV
DDBACK	ADARES
DDSIIN	ADARES ADASEL
DDREST1	ADASAV (LINK names used are DDREST1, DDREST01, DDREST02, and so on.)

Example for Use of the Concatenation Feature with ADARES

During the last nucleus session, three protection log files were produced with ADARES PLCOPY named F1, F2, F3.

When backing out the session to a specific point, use the following /SET-FILE-LINK (in ISP format, /FILE) statements for the ADARES BACKOUT function:

```
/SET-FILE-LINK DDBACK,F3
/SET-FILE-LINK DDBACK01,F2
/SET-FILE-LINK DDBACK02,F1
```

In ISP format:

```
/FILE F3,LINK=DDBACK
/FILE F2,LINK=DDBACK01
/FILE F1,LINK=DDBACK02
```

To regenerate the database from the protection log that was produced during the session, use the following /SET-FILE-LINK (in ISP format, /FILE) statements for the ADARES REGENERATE function:

```
/SET-FILE-LINK DDSIIN,F1
/SET-FILE-LINK DDSIIN01,F2
/SET-FILE-LINK DDSIIN02,F3
```

In ISP format:

```
/FILE F1,LINK=DDSIIN
/FILE F2,LINK=DDSIIN01
/FILE F3,LINK=DDSIIN02
```

Control Statement Read Procedure in Version 11.2 (OSD 2.0)

With BS2000 version 11.2 (OSD 2.0), the SYSIPT system file is no longer available. Beginning with version 5.3.3, ADABAS can read all control statements from the SYSDTA system file.

When running on BS2000 Versions 10.0 or 11.0, the SYSIPT assignment can still be used; however, Software AG recommends adapting all ADABAS utility and Entire Net-Work job control to indicate the SYSDTA system file before migrating to BS2000 version 11.2 (OSD 2.0).

ADARUN TAPEREL: Tape Release Option

The ADARUN parameter TAPEREL is required to perform the tape handling control for utilities that access files on tape. See the *ADABAS Operations* documentation for more information.

z/OS Systems

The DDNAME is formed by prefixing the characters DD to the file name.

To allow utilities to access data set information after closing, the DD statement for sequential data sets used in utilities should not contain FREE=CLOSE.

The BLKSIZE of a sequential file is determined as follows:

- If the column, "BLKSIZE by device" specifies "Yes" for a file, the default BLKSIZE depends on the device type as follows:

Tape:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column "BLKSIZE by device" does *not* specify "Yes" for a file, the file's BLKSIZE is obtained from the DD statement or data set label, if present. It must be present for any input file.
- If the column "BLKSIZE by device" does *not* specify "Yes" for a file *and* the BLKSIZE cannot be obtained from the DD statement or data set label, the value of the ADARUN QBLKSIZE parameter is used, if specified.

Except for ADACMP EBAND, the RECFM and LRECL of all sequential files are VB and BLKSIZE-4, respectively. For ADACMP EBAND, RECFM and LRECL must be available from the DD statement and/or data set label.

If the DCB BUFNO parameter is not provided on the DD statement, the operating system default will be used.

z/VM Systems

The DATADEF name is formed by prefixing the characters DD to the file name.

The BLKSIZE of a sequential file is determined as follows:

- If the column, "BLKSIZE by device" specifies "Yes" for a file, the BLKSIZE depends on the device type as follows:

Tape:	32760
FBA disk:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column "BLKSIZE by device" does *not* specify "Yes" for a file, the file's BLKSIZE is obtained from the DD statement or data set label, if present. It must be present for any input file.
- If the column "BLKSIZE by device" does *not* specify "Yes" for a file *and* the BLKSIZE cannot be obtained from the DD statement or data set label, the value of the ADARUN QBLKSIZE parameter is used, if specified.

For all sequential files except ADACMP EBAND, the RECFM is VB and LRECL is (BLKSIZE - 4). For ADACMP EBAND, RECFM and LRECL must be available from the DATADEF statement and/or data set label.

VSE Systems

The following items determine how a file is referenced by the utilities running under VSE:

- The file name is used as the filename on the DLBL or TLBL statement.
- If files for which the column "Concatenation" contains "Yes" are on tape, they may be concatenated as follows:
  - The file is first read using the indicated file name.
  - At the first end-of-file, "01" is appended to the file name and, if there is a TLBL statement for that filename, reading continues.
  - At each subsequent end-of-file, 1 is added to the file name and reading continues as long as there is a TLBL statement for that filename, up through a maximum of 99.
- Since VSE does not support reading multivolume tape files backward, each volume of the ADARES BACK file must be specified in reverse order from the way it was written on TLBL statements using the filenames BACK, BACK01, BACK02, and so on.

Any programmer logical unit may be used for sequential files on disk. The *VSE Tape SYS* number must be used for sequential files on tape; any or all of these numbers may be changed using procedures defined in the ADABAS Installation documentation.

The BLKSIZE of a sequential file is determined as follows:

- If the column "BLKSIZE by device" specifies "Yes" for a file, the BLKSIZE depends on the device type as follows:

Tape:	32760
FBA disk:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column "BLKSIZE by device" does *not* specify "Yes" for a file, the value of the ADARUN QBLKSIZE parameter is used, if specified.

For ADACMP EBAND, this BLKSIZE is checked and may then be changed to an actual BLKSIZE, depending on the RECFM and LRECL parameters as specified on ADACMP control cards, as follows:

If RECFM= ...	then the actual BLKSIZE= ...
F	LRECL.
FB	BLKSIZE/LRECL*LRECL, where the remainder of the division is discarded before the multiplication.
U	LRECL, which must not be greater than BLKSIZE.
V	LRECL+4, which must not be greater than BLKSIZE.
VB	BLKSIZE, which must not be less than LRECL+4.

The RECFORM of all sequential files except ADACMP EBAND is VARBLK. For ADACMP EBAND, it is provided by the RECFM parameter of a control statement.

To distinguish whether VSE message 4140D refers to the first or a subsequent volume of a multivolume tape file, message ADAI31 is written to the operator whenever a tape file is opened, but not at end-of-volume.

Concatenation of Sequential Input Files for VSE

In those cases where it is desired to use more than one data set as input medium for an ADABAS utility, a concatenation feature is provided by some operating systems (z/OS, for example).

For VSE, this feature is simulated by adding FILE statements with modified LINK names created from the original and a two-digit increment (ranging from 01 to 99):

```
// DLBL TEST  ,'firstfile'
// EXTENT ...
// DLBL TEST01,'secondfile'
// EXTENT ...
...
// DLBL TEST99,'lastfile'
// EXTENT ...
```

When processing input files that have the concatenation option at end-of-file (EOF) of one input file, a check is made to determine whether a FILE statement exists for the next data set. If it does not exist the Sequential Get call returns EOF; otherwise, the data set currently open is closed and an open is tried for the next file.

Files concatenated in this way must have the same file characteristics (block size, record format, and record size).

This concatenation feature applies also to files that are processed backwards. The order of the LINK names is the reverse of the creation order; for example, ADARES with BACK:

```
// DLBL BACK  ,'lastfile'
// EXTENT ...
// DLBL BACK01,'filebeforelast'
// EXTENT ...
...
// DLBL BACKnn,'firstfile'
// EXTENT ...
```

Note that this feature could also be used to process a multivolume file backwards, if each volume is specified with a separate FILE statement.

The following are the LINK names/utilities with the concatenation option:

DELn (where n=1-8)	ADASAV
EBAND	ADACMP ADALOD ADAMER
FULL	ADASAV
ISN	ADALOD
PLOG	ADAPLP ADASAV
BACK	ADARES
SIIN	ADARES ADASEL
REST1	ADASAV (LINK names used are REST1, REST101, REST102, and so on.)

Example for use of the Concatenation Feature with ADARES

During the last nucleus session, three protection log files were produced with ADARES PLCOPY named F1, F2, F3.

When deciding to back out the session to a specific point, the following FILE statements should be used for the ADARES BACKOUT function:

```
// DLBL BACK  ,'F3'
// EXTENT ...
// DLBL BACK01,'F2'
// EXTENT ...
// DLBL BACK02,'F1'
// EXTENT ...
```

To regenerate the database from the protection log that was produced during the session, the following FILE statements should be used for the ADARES REGENERATE function:

```
// DLBL SIIN  ,'F1'
// EXTENT ...
// DLBL SIIN01,'F2'
// EXTENT ...
// DLBL SIIN02,'F3'
// EXTENT ...
```

B Appendices

- Adabas Libraries (ADAVvLIB) 1190
- Adabas Files (ADAVvFIL) 1190

The VSE examples assume that the procedures for defining Adabas libraries (ADAVvLIB) and Adabas files (ADAVvFIL) have been cataloged into an accessible procedure library.

For information about cataloging these procedures, refer to the section *Catalog Procedures for Defining Libraries and the Database* in the VSE section of the Adabas Installation documentation.

Information about cataloging procedures for use with the Delta Save Facility are documented in the *Adabas Delta Save Facility* documentation.

Adabas Libraries (ADAVvLIB)

```
// PROC
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ *
* LIBRARY DEFINITIONS AND CHAINING FOR ADABAS                       *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ *
// SETPARM VERS=vrs                  <- CURRENT VERSION
// SETPARM ADALIB=SAGLIB             <- SAG PRODUCT LIBRARY
// SETPARM ADASUB=ADA&VERS           <- ADABAS SUBLIBRARY
// DLBL SAGLIB,'SAG.PRODUCT.LIBRARY'
// EXTENT ,vvvvvv
// LIBDEF *,SEARCH=&ADALIB..&ADASUB,TEMP
// LIBDEF PHASE,CATALOG=&ADALIB..&ADASUB,TEMP
// ASSGN SYS009,PRINTER
```

where

vrs	is the Adabas version, revision, and system maintenance (SM) level
vvvvvv	is the programmer logical unit assigned

Adabas Files (ADAVvFIL)

```
// ASSGN SYS031,dddd,VOL=ADA001,SHR
// ASSGN SYS032,dddd,VOL=ADA002,SHR
// ASSGN SYS033,dddd,VOL=ADA003,SHR
// ASSGN SYS034,dddd,VOL=ADA004,SHR
// DLBL ASSOR1,'EXAMPLE.ADAyyyyy.ASSOR1',99/365,DA
// EXTENT SYS031,ADA001,,,15,1500
// DLBL DATAR1,'EXAMPLE.ADAyyyyy.DATAR1',99/365,DA
// EXTENT SYS032,ADA002,,,15,3000
// DLBL WORKR1,'EXAMPLE.ADAyyyyy.WORKR1',99/365,DA
// EXTENT SYS033,ADA003,,,15,600
// DLBL PLOGR1,'EXAMPLE.ADAyyyyy.PLOGR1',99/365,DA
```

```
// EXTENT SYS034,ADA004,,,15,600
// DLBL PLOGR2,'EXAMPLE.ADAyyyyy.PLOGR2',99/365,DA
// EXTENT SYS034,ADA004,,,615,600
// DLBL CLOGR1,'EXAMPLE.ADAyyyyy.CLOGR1',99/365,DA
// EXTENT SYS034,ADA004,,,1215,750
// DLBL CLOGR2,'EXAMPLE.ADAyyyyy.CLOGR2',99/365,DA
// EXTENT SYS034,ADA004,,,1965,750
// DLBL TEMPR1,'EXAMPLE.ADAyyyyy.TEMPR1',99/365,DA
// EXTENT SYS032,ADA002,,,3015,1500
// DLBL SORTR1,'EXAMPLE.ADAyyyyy.SORTR1',99/365,DA
// EXTENT SYS033,ADA003,,,615,375
// EXTENT SYS034,ADA004,,,2715,375
// DLBL RLOGR1,'EXAMPLE.ADAyyyyy.RLOGR1',99/365,DA
// EXTENT SYS033,ADA003,,,990,150
```

C Appendix

Here is the FDT of the Personnel demo file distributed with Adabas. Other demo files distributed with Adabas include an Employees demo file, a Vehicle demo file, and a Miscellaneous demo file. The Personnel demo file includes data that makes use of the expanded features provided in Adabas 8, such as large object support. Associated with the Personnel demo file is a new LOB demo file, containing the LOB data referenced by the Personnel demo file. During installation, the LOB demo file is loaded in the same job used to load the Personnel demo file.

Note: The Personnel demo file must be installed on a UES-enabled database because it includes wide-character format (W) fields.

```
1,AO                          personnel-data
2,AA,8,A,DE,UQ                personnel-id
2,AB                          id-data
3,AC,4,F,DE                   personnel-no !UQ taken!
3,AD,8,B,NU                   id-card
3,AE,0,A,LA,NU,NV,NB          signature
1,BO                          full-name
2,BA,40,W,NU                  first-name
2,BB,40,W,NU                  middle-name
2,BC,50,W,NU,DE               name
1,CA,1,A,FI                   mar-stat
1,DA,1,A,FI                   sex
1,ES,2,B,NU                   birth
1,EA,4,P,DE,NC                birth
1,FO,PE                       full-address
2,FA,60,W,NU,MU               address-line
2,FB,40,W,DE,NU               city
2,FC,10,A,NU                  post-code
2,FD,3,A,NU                   country
2,F1                          phone-numbers
3,FE,6,A,NU                   area-code
3,FF,15,A,NU                  home-phone
3,FG,15,A,NU                  home-fax
3,FH,15,A,NU                  private-mobile
```

```
3,FI,80,A,NU,MU,DE            private-email
1,IO,PE                       business-address
2,IA,40,W,NU,MU               address-line
2,IB,40,W,DE,NU               city
2,IC,10,A,NU                  post-code
2,ID,3,A,NU                   country
2,IE,5,A,NU                   room number
2,I1                          telephone
3,IF,6,A,NU                   area-code
3,IG,15,A,NU                  business-phone
3,IH,15,A,NU                  business-fax
3,II,15,A,NU                  business-mobile
3,IJ,80,A,NU,MU,DE            business-email
1,JA,6,A,DE                   dept
1,KA,66,W,DE,NU               job-title
1,LO,PE                       income
2,LA,3,A,NU                   curr-code
2,LB,6,P,NU                   salary P9.2
2,LC,6,P,NU,MU,DE             bonus P9.2
1,MA,4,G,NU                   total income (EUR)
1,NO                          leave-date
2,NA,2,U                      leave-due
2,NB,3,U,NU                   leave-taken N2.1
1,OO,PE                       leave-booked
2,OA,8,U,NU                   leave-start
2,OB,8,U,NU                   leave-end
1,PA,3,A,DE,NU,MU             language
1,QA,7,P                      last update (*TIMX)
1,RA,0,A,LB,NU,NV,NB          picture
1,SO,PE                       documents
2,SA,80,W,NU                  document-description
2,SB,3,A,NU                   document-type
2,SC,0,A,LB,NU,NV,NB,MU       document


H1=NA(1,2),NB(1,3)
S1=JA(1,2)
S2=JA(1,6),BC(1,40)
S3=LA(1,3),LB(1,6)
```

Index

, 1126

A

ABEND34 parameter, 1122
ACTIVATE parameter, 330, 334
ADAACK utility
  ACCHECK function, 13
  BS2000 JCL, 18
  functional overview, 11
  VSE JCS, 21
  z/OS JCL, 19
  z/VM JCL, 20
Adabas
  ADASAV release support, 935
Adabas 8
  ADACNV CONVERT utility support, 185
  ADACNV REVERT utility support, 189
  ADAORD considerations, 596
  ADASAV support, 935
  valid index ranges for ADASEL IF statements, 1050, 1053
Adabas control block
  start logging
    using utility, 292
  stop logging
    using utility, 292
Adabas Delta Save Facility
  change flags, 828
  database status report, 828
  display status
    using utility, 287
Adabas file
  compressing using ADACMP COMPRESS, 83
Adabas files
  personnel file FDT, 1193
  sequential list by utility, 1175
Adabas Review
  deactivate
    using utility, 295
  local mode
    switch to using utility, 295
  setting or modifying the hub ID, 295
Adabas Statistics Facility
  using ADADBS REFRESHSTATS with, 314
ADACDC utility
  errors with ADADBS or ADALOD ISNREUSE, 36
  examples, 49
  extract file, 27
  functional overview, 25
  input data, 29
  JCL requirements and examples, 51
    BS2000, 52
    VSE, 55
    z/OS, 53
    z/VM, 54
  operating system factors, 39
    BS2000, 41
    VSE, 40
    z/OS, 40
  operation, 26
  parameters, 34
  phases of operation, 26
  primary output file, 28-29
    checkpoints, 28
  running, 33
  syntax, 34
  transaction file, 30
  user exit, 43
    calls, 46
    installing, 44
    interface description, 44
    using to update or add records, 47
ADACMP utility
  BS2000 JCL requirements and examples, 147
  collation with user exits, 147
  COMPRESS examples, 92
  COMPRESS function, 60, 83
  COMPRESS function compressed data records output, 162
  COMPRESS function output, 161
  COMPRESS function rejected data records output, 162
  COMPRESS function storage requirements report, 168
  compression with user exit 6, 146
  data compression, 78
  data verification, 78
  DECOMPRESS examples, 101
  DECOMPRESS function, 61, 95
  DECOMPRESS function output, 171
  DECOMPRESS function rejected data records output, 172
  DECOMPRESS function with multiclient files, 101
  input data requirements, 63
  input data structure, 64
  JCL requirements and examples, 145
  multiple-value field counts, 64
  periodic group count input requirements, 66
  processing, 71
  restart considerations, 81
  user exit 6, 81
  variable-length field size, 69
  VSE JCL requirements and examples, 157

z/OS JCL requirements and examples, 150
z/VM JCL requirements and examples, 155
ADACNV utility
Adabas 8 conversion support, 185
Adabas 8 reversion support, 189
CONVERT function, 183
functional overview, 179
JCL requirements and examples, 191
BS2000, 192
VSE, 197
z/OS, 195
z/VM, 196
REVERT function, 187
ADADBS ISNREUSE
errors with ADACDC utility, 36
ADADBS REPLICATION function
ACTIVATE parameter, 330
DEACTIVATE parameter, 330
DSBI parameter, 331
FILE parameter, 330
KEY parameter, 331
MODIFY parameter, 330
NOKEY parameter, 331
OFF parameter, 330
ON parameter, 330
TARGET parameter, 331
ADADBS REPTOR function
ACTIVATE parameter, 334
CLOSE parameter, 334
DBID parameter, 334
DEACTIVATE parameter, 334
description, 333
DESTINATION parameter, 335
examples, 336
FILE parameter, 335
IQUEUE parameter, 335
OPEN parameter, 334
SUBSCRIPTION parameter, 335
ADADBS utility, 333
ADD data set function, 203
ALLOCATE function, 207
CHANGE function, 211
checking syntax, 202
CVOLSER function, 215
DEALLOCATE function, 217
DECREASE function, 221
DELCP function, 225
DELETE function, 229
DSREUSE function, 233
ENCODEF function, 237
functional overview, 201
INCREASE function, 241
BS2000 procedure, 247
general procedure, 243
VSE procedure, 244
z/OS procedure, 243
z/VM procedure, 246
ISNREUSE function, 249
JCL requirements and examples
BS2000, 360
VSE, 363
z/OS, 361
z/VM, 362
MODFCB function, 253
MUPEX function, 257
NEWFIELD function, 261
ONLINVERT function, 265
ONLREORFASSO function, 269
ONLREORFDATA function, 273
ONLREORFILE function, 277
OPERCOM function, 281
PRIORITY function, 299
RECORDSPANNING function, 303
RECOVER function, 307
REFRESH function, 309
REFRESHSTATS function, 313
RELEASE function, 317
RENAME function, 321
RENUMBER function, 325
REPLICATION function, 329
RESETDIB function, 339
RESETPPT function, 343
SPANCOUNT function, 347
TRANSACTIONS function, 351
UNCOUPLE function, 355
ADADCK utility
DSCHECK function, 369
JCL requirements and examples
BS2000, 374
VSE, 377
z/OS, 375
z/VM, 376
ADADEF utility
DEFINE function, 383
functional overview, 381
checkpoint file, 382
database components, 382
JCL requirements and examples
BS2000, 402
VSE, 407
z/OS, 403
z/VM, 405
MODIFY function, 393
NEWWORK function, 397
ADAEND operator command
using utility, 283
ADAFRM utility
all functions, 414
format new RABNs, 412
formatting database components, 412
functional overview, 411
JCL requirements and examples
BS2000, 420
VSE, 425
z/OS, 422
z/VM, 423
reset data set blocks/cylinders to zeros, 412
ADAICK utility
ACCHECK function, 431
ASSOPRINT function, 435
BATCH function, 437
DATAPRINT function, 439
DSCHECK function, 441
DUMP function, 445
examples, 473
FCBPRINT function, 447
FDTPRINT function, 451
functional overview, 429

GCBPRINT function, 453
ICHECK function, 455
INT function, 457
JCL requirements and examples
BS2000, 476
VSE, 480
z/OS, 478
z/VM, 479
NIPRINT function, 459
NOBATCH function, 461
NODUMP function, 463
NOINT function, 465
PPTPRINT function, 467
UIPRINT function, 471
user exit collation, 476
ADAINV utility
COUPLE function, 485
examples, 497
functional overview, 483
INVERT function, 493
JCL requirements and examples
BS2000, 500
VSE, 506
z/OS, 503
z/VM, 504
space allocation during execution, 497
user exits
collation, 360, 500
ADALOD ISNREUSE
errors with ADACDC utility, 36
ADALOD utility
JCL requirements and examples
BS2000, 564
VSE, 572
z/OS, 568
z/VM, 570
LOAD function, 513
examples, 531
input data for, 533
space allocation for file, 533
space/statistics report, 561
storage requirements and use, 555
Temp data set requirements, 557
UPDATE function, 541
Associator updating with, 553
descriptor information generation, 553
examples, 549
input requirements, 552
mass updates to expanded files, 554
space allocation, 553
updating Associator using the LOAD function, 536
user exits
collation, 564
ADAM
load files with
using utility, 517
ADAMER utility
examples, 582
JCL requirements and examples
BS2000, 586
VSE, 589
z/OS, 587
z/VM, 588
output report, 583
syntax, 579
ADAORD utility
Adabas 8 considerations, 596
JCL requirements and examples
BS2000, 672
VSE, 683
z/OS, 677
z/VM, 680
REORASSO function, 597
REORDATA function, 605
REORDB function, 611
REORFASSO function, 623
REORFDATA function, 631
REORFILE function, 637
RESTRUCTUREDB function, 647
RESTRUCTUREF function, 653
STORE function, 659
ADAPLP utility
examples, 696
JCL requirements and examples
BS2000, 698
VSE, 706
z/OS, 701
z/VM, 704
PLOGPRI function, 691
SPLOGPRI function, 691
WORKPRI function, 691
ADAPRI utility
ASSOPRI function, 715
CLOGPRI function, 715
DATAPRI function, 715
examples, 717
JCL requirements and examples
BS2000, 720
VSE, 723
z/OS, 721
z/VM, 722
PLOGPRI function, 715
RLOGPRI function, 715
SORTPRI function, 715
TEMPPRI function, 715
WORKPRI function, 715
ADAR2E utility
for recovering BS2000 pubset members, 786
ADARAI utility
BS2000 RECOVER skeleton job control input, 776
BS2000 skeleton job control input for RECOVER function, 780
CHKDB function, 731
concepts, 728
DISABLE function, 733
file-level recovery, 760
function directory, 725
JCL requirements and examples
BS2000, 772
LIST function, 735
LIST function BS2000 output examples, 739
LIST function z/OS output examples, 740
PREPARE function, 749
RECOVER function, 753
RECOVER function input data sets, 758
RECOVER function output job stream, 758
REMOVE function, 769
restarting RECOVER function after interruption, 768

running RECOVER function, 759
skeleton job control input for RECOVER function, 764
special considerations under BS2000, 772
special considerations under VSE, 795
VSE JCL requirements and examples, 795
VSE LIST function output examples, 745
z/OS JCL requirements and examples, 788
z/OS skeleton job control input for RECOVER function, 789
z/VM JCL requirements and examples, 793
ADAREP utility
BS2000 JCL requirements and examples, 842
examples, 805
VSE JCL requirements and examples, 845
z/OS JCL requirements and examples, 843
z/VM JCL requirements and examples, 844
ADARES utility
BACKOUT DPLOG function, 867
BACKOUT function, 859
BACKOUT MPLOG function, 867
BS2000 JCL requirements and examples, 906
CLCOPY function, 875
COPY function, 879
function directory, 852
MERGE CLOG function, 883
PLCOPY function, 885
REGENERATE function, 889
REPAIR function, 899
VSE JCL requirements and examples, 925
z/OS JCL requirements and examples, 913
z/VM JCL requirements and examples, 920
ADASAV utility
Adabas release support, 935
BS2000 JCL requirements and examples, 1014
RESTONL FILES function, 943
RESTONL FMOVE function, 951
RESTONL function, 937
overview, 934
RESTONL function overview, 935
RESTONL GCB function, 963
RESTORE FILES function, 975
RESTORE FMOVE function, 981
RESTORE function, 969
RESTORE function overview, 934
RESTORE GCB function, 991
RESTPLOG function, 997
RESTPLOG function overview, 935
SAVE FILES function, 1007
SAVE function, 1001
SAVE function overview, 935
VSE JCL requirements and examples, 1029
z/OS JCL requirements and examples, 1020
z/VM JCL requirements and examples, 1025
ADASEL utility
BS2000 JCL requirements and examples, 1062
DISPLAY instruction syntax, 1051
global parameters, 1059
IF statement syntax, 1044
NEWPAGE instruction syntax, 1057
OUTPUT instruction syntax, 1054
SKIP instruction syntax, 1057
syntax, 1039
additional selection criteria, 1042
date/time, 1043
output instruction, 1051
SELECT statement, 1041
value criterion, 1046
WITH clause, 1043
valid index ranges for Adabas 8 IF statements, 1050, 1053
VSE JCL requirements and examples, 1065
z/OS JCL requirements and examples, 1063
z/VM JCL requirements and examples, 1064
ADAULD utility
BS2000 JCL requirements and examples, 1090
input processing, 1081
JCL requirements and examples
VSE, 1096
LOAD FILE function DDISN parameter, 1072
output processing, 1085
UNLOAD FILE function, 1071
user exit 9 processing, 1087
z/OS JCL requirements and examples, 1092
z/VM JCL requirements and examples, 1094
ADAVAL utility
BS2000 JCL requirements and examples, 1111
output example, 1107
rejected ISNs, 1101
VALIDATE function, 1103
VSE JCL requirements and examples, 1114
z/OS JCL requirements and examples, 1112
z/VM JCL requirements and examples, 1113
ADAWRK utility
BS2000 JCL requirements and examples, 1156
overview, 1119
reports, 1125
syntax, 1121
VSE JCL requirements and examples, 1154
z/OS JCL requirements and examples, 1152
z/VM JCL requirements and examples, 1155
ADAZAP utility
BS2000 JCL requirements and examples, 1168
VSE JCL requirements and examples, 1171
z/OS JCL requirements and examples, 1169
z/VM JCL requirements and examples, 1170
address converter
allocate an extent using ADADBS utility, 208
check against Data Storage
using utility, 13
check index against using ADAICK, 455
deallocate an extent using the ADADBS utility, 218
space allocation
using utility, 534, 553
ALOCKF operator command
using utility, 284
alphanumeric fields
no conversion option (NV), 116
Associator
add data set to
using utility, 204
check physical structure of using ADAICK, 429
coupling lists
creating, 489
decrease size of data set
using utility, 221
formatting using ADAFRM, 414
increase size of data set
using utility, 241
print blocks using ADAPRI, 713
print/dump block(s)

using utility, 435
reorder
using utility, 594
reordering for a file using ADAORD REORFASSO, 623
reordering for database using ADAORD REORASSO, 597
reset blocks/cylinders to zeros
using utility, 415
resetting PPT, 343
updating
using utility, 553
updating using ADALOD LOAD, 536
validate values against Data Storage contents using ADAVAL VALIDATE, 1103
ATM
loading system files, 515
attached buffers
command to display usage, 289

B

blank compression
NB field option, 115
bold, 3
braces ({}), 4
brackets ([]), 4
BS2000
ADAR2E utility for recovering lost pubset members, 786
additional recovery log information for, 736
special considerations for using ADARAI, 772

C

CANCEL operator command
using utility, 284
checkpoint file
define, 383
CHECKPOINT parameter, 1122
checkpoint record reporting, 1150
checkpoints
database status report, 835
delete
using utility, 225
written by Adabas nucleus/utilities, 836
choices in syntax, 4
CLOSE parameter, 334
CMID parameter, 1122
collation descriptor
define
using ADAINV, 494
defining in ADACMP COMPRESS, 124
Command log
add data set to
using utility, 204
close/switch dual
using utility, 291
copy a dual data set to a sequential data set using ADARES CLCOPY, 875
formatting using ADAFRM, 414
merging multiple, 883, 917
print blocks from multiple CLOGs using ADAPRI, 713
start logging
using utility, 292
stop logging
using utility, 292
command queue
command to display usage, 289
command queue element
display posted
using utility, 286
command queue elements
display
using utility, 288
compressing files, 83
convert database
to higher version, 183
to lower version, 187
coparameters
specifying, 7
CT
ADARUN parameter
command to override setting, 286
CT operator command
using utility, 286

D

data compression
ADACMP utility, 60, 78
fields with NC option, 118
data decompression
ADACMP utility, 61, 95
fields with NC option, 118
data definition
COLDE statement
of ADAINV INVERT, 495
COLDE statement in ADACMP COMPRESS, 124
DE - descriptor field option, 110
FI - fixed storage field options, 110
field options, 109
FIELD statement
of ADAINV INVERT, 495
FNDEF statement in ADACMP COMPRESS, 105
HYPDE statement
of ADAINV INVERT, 495
HYPDE statement in ADACMP COMPRESS, 127
LA - long alphanumeric field options, 111
LB - large object field options, 112
MU - multiple-value field option, 113
NB - blank compression field options, 115
NC - null not counted in compression/decompression, 118
NC - null not counted SQL field options, 118
NN - not null SQL field option, 120
NV - no conversion field options, 116
PE - periodic group field option, 122
PHONDE statement
of ADAINV INVERT, 495
PHONDE statement in ADACMP COMPRESS, 130
SUBDE statement
of ADAINV INVERT, 495
SUBDE statement in ADACMP COMPRESS, 132
SUBFN statement, 134
SUPDE statement
of ADAINV INVERT, 495
SUPDE statement in ADACMP COMPRESS, 135
SUPFN statement, 143
syntax using ADACMP, 103
U - null value suppression field options, 115
UQ - unique descriptor field options, 116

XI - exclude PE instance from UQ field option, 117
data integrity block
display entries
using utility, 286
reset entries in
using utility, 339
data sets
intermediate coupling storage
calculate using utility, 488
print blocks using ADAPRI, 713
Data Storage
add data set to
using utility, 204
allocate an extent using ADADBS utility, 208
check the address converter against
using utility, 13
deallocate an extent using the ADADBS utility, 218
decrease size of data set
using utility, 221
formatting using ADAFRM, 414
increase size of data set
using utility, 241
print blocks using ADAPRI, 713
print/dump block(s) using ADAICK, 439
print/dump record using ADAICK, 441
reorder
using utility, 594
reordering for a file using the ADAORD REORFDATA, 631
reordering for database using ADAORD REORDATA, 605
repair blocks
using utility, 899
resetting blocks/cylinders to zeros using ADAFRM, 415
reuse blocks
using utility, 233
space allocation
using utility, 535
validate against Associator values using ADAVAL VALIDATE, 1103
database
automate and optimize recovery using ADARAI, 728
build job stream for restore, 753
change name assigned to
using utility, 321
check active/inactive status
using utility, 731
define, 383
delete file from
using utility, 229
loading files into using ADAORD STORE, 595
modify physical blocks using ADAZAP utility, 1161
produce status report using ADAREP REPORT, 799
quiesce, 351
recovering using ADARAI RECOVER, 753
reordering Associator using ADAORD REORASSO, 597
reordering Data Storage and Associator using ADAORD REORDB, 611
reordering Data Storage using ADAORD REORDATA, 605
restore from offline source using ADASAV RESTORE, 969
restore from online source using ADASAV RESTONL, 937
restore incremental from offline source using ADASAV RESTORE GCB, 991
restore incremental from online source using ADASAV RESTONL GCB, 963
restore updates performed between two checkpoints using ADARES REGENERATE, 889
save using ADASAV SAVE, 1001
unload to sequential data set using ADAORD RESTRUCTUREDB, 595
unloading to sequential data set using ADAORD RESTRUCTUREDB, 647
DAUQ operator command
using utility, 286
DBID parameter, 334
DCQ operator command
using utility, 286
DD/JCLIN
requirement for ADARAI, 758
DDIB operator command
using utility, 286
DDSF operator command
using utility, 287
DEACTIVATE parameter, 330, 334
default parameter values, 3
descriptor
collation, 124
define
using ADAINV utility, 493
field option (DE), 110
hyperdescriptor, 127
phonetic, 130
release from descriptor status
using utility, 317
specify for file coupling
using utility, 485
subdescriptor, 132
superdescriptor, 135
unique, 116
DESTINATION parameter, 335
DFILES operator command
using utility, 287
DFILUSE operator command
using utility, 287
DHQA operator command
using utility, 287
disk volume
print Adabas extents located on
using utility, 215
DLOCKF operator command
using utility, 287
DNC operator command
using utility, 288
DNH operator command
using utility, 288
DNU operator command
using utility, 288
DONLSTAT operator command
using utility, 288
DPARM operator command
using utility, 289
DRES operator command
using utility, 289
DSBI parameter, 331
DSTAT operator command
using utility, 289
DTH operator command
using utility, 290
dump

terminate online status
using utility, 294
dump print format
activating, 445
suppressing, 463
DUQ operator command
using utility, 290
DUQA operator command
using utility, 290
DUQE operator command
utility, 290
DUUQE operator command
using utility, 290

E

ellipsis (...), 4
encodings
change
using utility, 393
ETID parameter, 1122
expanded file chain
checking uniqueness of descriptor within, 117
expanded files
load anchor file, 518
loading, 536
mass updates to, 554

F

FEOFCL operator command
using utility, 291
FEOFPL operator command
using utility, 291
field definition
syntax using ADACMP, 103
Field Definition Table
database status report, 831
Field Definition Table (FDT)
add a field to
using utility, 261
print/dump
using utility, 451
field definitions, 105, 121
fields
add
using utility, 261
change standard length of
using utility, 211
subfield, 134
superfield, 143
File Control Block (FCB)
dump/print
using utility, 447
file coupling
coupling lists, 490
lists
create using utility, 489
temporary space for
calculate using utility, 488
uncouple
using utility, 355
using utility, 485
file encoding
modify, 237
file extents
allocate
using utility, 207
deallocate
using utility, 217
print on given disk volume
using utility, 215
file information, 823
file options, 816, 826
FILE parameter, 330, 335
File report, 1128
file space allocations, 819
Files
restore updates performed between two checkpoints using ADARES REGENERATE, 889
files
activating or deactivating replication, 329
Adabas Delta Save change flags, 828
Adabas sequential list by utility, 1175
allocate space for, 533
BS2000 concatenation input of Adabas sequential files, 1180
BS2000 control statement read procedure of Adabas sequential files, 1182
BS2000 record formats of Adabas sequential files, 1178
BS2000 tape release option of Adabas sequential files, 1182
build job stream for restore, 753
change file number of
using utility, 325
change name assigned to
using utility, 321
compressing using ADACMP COMPRESS, 83
counting spanned records, 347
decompressing multiclient files, 101
delete
using utility, 229
determining names and blocks sizes of Adabas sequential, 1176
display locked, 287
display quantity of user types, 287
display total commands processed, 287
enabling or disabling record spanning, 303
Field Definition Table report, 831
information about, 823
load
using utility, 513
load into existing database using ADAORD STORE, 659
load with ADAM option, 517
loading into existing database using ADAORD STORE, 595
locking at all security levels, 291
locking for all non-utility use, 291
locking for all users except EXU or EXF users, 292
locking in advance, 284
modify parameters of
using utility, 253
option settings report, 826
personnel file FDT, 1193
recovering with ADARAI RECOVER, 760
remove advance lock
on all files, 294
on specified file, 294
reordering Associator and Data Storage using ADAORD REORFILE, 637

reordering Associator using ADAORD REORFASSO function, 623
reordering Data Storage using ADAORD REORFDATA, 631
reset to empty status
using utility, 309
restore to any RABNs from offline source using ADASAV RESTORE FMOVE, 981
restore to any RABNs from online source using ADASAV RESTONL FMOVE, 951
restore to original RABNs from offline source using ADASAV RESTORE FILES, 975
restore to original RABNs from online source using ADASAV RESTONL FILES, 943
save using ADASAV FILES, 1007
space allocation report, 829
special descriptors report, 833
specifying MU or PE limits, 257
stop users of specified
using utility, 295
types, 515
unloading to sequential data set using ADAORD RESTRUCTUREDB, 595
unloading to sequential data set using ADAORD RESTRUCTUREF, 653
unlock specified
using utility, 297
unlocking for utility use, 297
VSE concatenation of input sequential files, 1186
VSE record formats of Adabas sequential files, 1184
z/OS record formats of Adabas sequential files, 1183
z/VM record formats of Adabas sequential files, 1183
FILES parameter, 1122
fixed storage (FI)
use in ADACMP, 110
FORCE parameter, 1122
format buffer
start logging
using utility, 292
stop logging
using utility, 292
format pool
command to display usage, 289

G

general control block (GCB)
print/dump
using utility, 453
general database information, 812
general file status, 815
generation, 729
group definitions, 105

H

HALT operator command
using utility, 291
hold queue
command to display usage, 289
display count of ISNs
using utility, 288
hold queue elements
display
using utility, 287
hyperdescriptor
define
using ADAINV, 494
defining in ADACMP COMPRESS, 127

I

I/O activity
start logging
using utility, 292
stop logging
using utility, 293
IF statements
valid index ranges, 1050, 1053
indentation, 4
index
check against address converter using ADAICK, 455
space allocation
by ADAINV, 497
using utility, 533
space allocation for coupling lists, 491
interpreted print format
activating, 457
suppressing, 465
invert
start online process
using utility, 265
IQUEUE parameter, 335
ISN buffer
start logging
using utility, 292
stop logging
using utility, 293
ISNs
file of unloaded, 1072
format for specifying, 550
set to reuse
using utility, 249
italic, 3

K

KEY parameter, 331

L

large object (LB)
field option, 112
LOCKF operator command
using utility, 291
locking files
in advance, 284
LOCKU operator command
using utility, 291
LOCKX operator command
using utility, 292
LOGGING
operator command
using utility, 292
LOGxx
operator command
using utility, 292
long alphanumeric (LA)

field option, 111
lowercase, 3
LWP parameter, 1122

M

minimum keywords, 3
mixed case, 3
MODIFY parameter, 330
MU fields
setting maximum for a file, 257
multiclient files
assigning a new owner ID using ADALOD LOAD, 521
decompressing, 101
loading, 538
specify length of owner ID using ADALOD LOAD, 523
unloading using ADAULD, 1073
multiple-value fields
field option (MU), 113

N

NOKEY parameter, 331
NOLOGGING
operator command
using utility, 292
NOPPT parameter, 1122
normal font, 3
normal index
allocate an extent using ADADBS utility, 208
deallocate an extent using the ADADBS utility, 218
print/dump
using utility, 459
NOUSERABEND parameter, 1123
nucleus
display current operating status
using utility, 289
null value
not allowed (NN) field option, 120
not counted (NC) field option, 118
SQL support, 117
suppression (NU) field option, 115
null value indicator
specified in record buffer, 119

O

OFF parameter, 330
ON parameter, 330
online invert
start
using utility, 265
online process
display status of
using utility, 288
resume a suspended process
using utility, 293
stop cleanly
using utility, 293
suspend
using utility, 293
online reorder
Associator, 269
Data Storage, 273
file, 277
ONLRESUME
operator command
using utility, 293
ONLSTOP
operator command
using utility, 293
ONLSUSPEND
operator command
using utility, 293
OPEN parameter, 334
operator commands
ADADBS OPERCOM function, 281
optional syntax elements, 4

P

Parallel Participant Table (PPT)
print/dump
using utility, 467
parameters
positional values
specifying, 7
PE groups
setting maximum for a file, 257
periodic group definition
FNDEF statement in ADACMP COMPRESS, 121
periodic group definitions, 121
periodic groups
data definition option (PE), 122
phonetic descriptor
define
using ADAINV, 494
defining using ADACMP COMPRESS, 130
physical database layout, 821
PPT
resetting on Associator data set, 343
printout
set width to 132 characters using ADAICK, 437
set width to 80 characters
using utility, 461
printouts
dump print format, 445
interpreted format, 457
suppressing dump print format, 463
suppressing interpreted format, 465
procedures
for defining VSE libraries and files for examples, 1189
Protection log
close/switch dual
using utility, 291
data set types, 857
dual backout updates between two checkpoints using ADARES BACKOUT DPLOG, 867
dual copy to sequential data set using ADARES PLCOPY, 885
formatting using ADAFRM, 414
information described, 857
merging multiple, 915
multiple backout updates between two checkpoints using ADARES BACKOUT MPLOG, 867
print blocks from multiple PLOGs using ADAPRI, 713
print records from multiple PLOGs using ADAPLP, 692

print records from sequential intermediate PLOGs using ADAPLP, 692
print records from sequential PLOGs using ADAPLP, 692
restore using ADASAV RESTPLOG, 997
sequential backout updates between two checkpoints using ADARES BACKOUT, 859
use of compressed data, 857
protection log
sequential copy using ADARES COPY
copy using utility, 879
punctuation and symbols in syntax, 5

Q

quiesce database
ADADBS function, 351

R

RALOCKF operator command
using utility, 294
RALOCKFA operator command
using utility, 294
RDUMPST operator command
using utility, 294
read-only status
switch on/off
using utility, 284, 294
READONLY
operator command
using utility, 294
record buffer
null value indicator value, 119
start logging
using utility, 292
stop logging
using utility, 293
records
add/delete
using utility, 541
counting spanned, 347
enabling or disabling spanned, 303
recovery
building the job stream, 759
check before starting, 767
job stream skeleton for BS2000, 776
of lost BS2000 pubset members, 786
restarting after interruption, 768
restarting job stream, 768
skeleton job stream, 764
using ADARAI RECOVER, 753
recovery aid (see ADARAI utility)
Recovery log
additional information for BS2000, 736
checking content with ADARAI LIST, 736
deactivate using ADARAI DISABLE, 733
description, 729
disabling using ADARAI REMOVE, 769
display contents using ADARAI LIST, 735
formatting using ADAFRM, 415
initialize and start using ADARAI PREPARE, 749
preparing using ADARAI, 728
preparing with ADARAI PREPARE, 750
print blocks using ADAPRI, 713
units of recovery, 729
redo pool
command to display usage, 289
reorder Associator
start online process
using utility, 269
reorder Data Storage
start online process
using utility, 273
reorder file
start online process
using utility, 277
repeated syntax elements, 4
replication
activating or deactivating, 329
REPLICATOR parameter, 526, 667
Report
database status checkpoint information, 835
REPORTFILE parameter, 1123
Reports
database status description, 811
reports
ADAWRK utility, 1125
checkpoint record, 1150
database status file information, 823
File, 1128
Summary, 1126
Transaction, 1131
work part 1 recovery report description, 1125
REPTOR parameter, 390, 395
required syntax elements, 4
resources
display current usage
using utility, 289
statistics
command to display, 289
RESTONL function, 935
RESTPLOG function, 935
resume normal processing
ADADBS function, 351
revert database
to lower version, 187
REVIEW
operator command
using utility, 295
RLOG (see Recovery log)
RPLDBSI parameter, 527
RPLKEY parameter, 527
RPLLOAD parameter, 528, 548
RPLTARGETID parameter, 528
RPLUPDATEONLY parameter, 529, 602

S

save tape
copy using ADARES COPY utility, 879
search buffer
start logging
using utility, 292
stop logging
using utility, 293
secondary address converter
allocate an extent using ADADBS utility, 208
deallocate an extent using the ADADBS utility, 218

security pool
command to display usage, 289
session
cancel immediately
using utility, 284
display current parameters
using utility, 289
reset statistical values for
using utility, 313
stop
using utility, 291
terminate normally
using utility, 283
SLOG parameter, 529, 667
Sort data set
formatting using ADAFRM, 414
print blocks using ADAPRI, 713
space
calculating for file coupling lists using ADAINV, 490
estimation report (ADACMP), 168
for temporary file coupling, 488
recover
using utility, 307
space allocated to database components, 814
spanned records
counting, 347
enabling or disabling for a file, 303
special descriptors, 833
SQL
null representation support, 117
STOPF
operator command
using utility, 295
STOPI
operator command
using utility, 295
STOPU
operator command
using utility, 296
storage
fixed (FI), 110
subdescriptor
define
using ADAINV, 494
defining using ADACMP COMPRESS, 132
subfield
defining using ADACMP COMPRESS, 134
subparameter syntax, 4
subparameters
specifying, 7
SUBSCRIPTION parameter, 335
SUMMARY parameter, 1123
Summary report, 1126
superdescriptor
define
using ADAINV, 494
defining using ADACMP COMPRESS, 135
superfield
defining using ADACMP COMPRESS, 143
suspend normal processing
ADADBS function, 351
SYNCC
operator command
using utility, 296
Syntax conventions
indentation, 4
syntax conventions
bold, 3
braces ({}), 4
brackets ([]), 4
defaults, 3
ellipsis (...), 4
italic, 3
lowercase, 3
minimum keywords, 3
mixed case, 3
mutually exclusive choices, 4
normal font, 3
optional elements, 4
punctuation and symbols, 5
repeated elements, 4
required elements, 4
subparameters, 4
underlining, 3
uppercase, 3
vertical bars (|), 4

T

table of ISNs pool
command to display usage, 289
table of sequential commands pool
command to display usage, 289
TARGET parameter, 331
Temp data set
ADALOD requirements for, 557
formatting using ADAFRM, 415
print blocks using ADAPRI, 713
space allocation
using utility, 535
TEST parameter, 1123
threads
display status
using utility, 290
timeout control
interregion communication limit
command to override setting, 286
non-activity limit set for access-only users, 297
non-activity limit set for ET logic users, 297
non-activity limit set for exclusive control users, 297
TIMEZONE parameter, 1123
TNAA
operator command
using utility, 297
TNAE
operator command
using utility, 297
TNAX
operator command
using utility, 297
transaction
set time limit for ET logic users
using utility, 297
transaction ID (XID) pool
command to display usage, 289
transaction processing
suspend/resume, 351
Transaction report, 1131

TRANSACTIONS parameter, 1124
TSOSCAT (BS2000)
  reading members with ADAR2E utility, 786
TT
  operator command
    using utility, 297

U

underlining, 3
unique descriptor
  define
    using ADALOD, 530
  exclude PE instance, 117
  use in ADACMP, 116
unique descriptor pool
  command to display usage, 289
Universal Encoding Support (UES)
  no conversion field option (NV), 116
UNLOCKF
  operator command
    using utility, 297
UNLOCKU
  operator command
    using utility, 297
UNLOCKX
  operator command
    using utility, 297
upper index
  allocate an extent using ADADBS utility, 208
  deallocate an extent using the ADADBS utility, 218
  print/dump
    using utility, 471
uppercase, 3
user data
  start logging
    using utility, 292
  stop logging
    using utility, 293
user exits
  6, used with ADACMP utility, 81
  9, used with ADAULD, 1087
  ADACDU, 43
  hyperdescriptor, 127
user queue
  command to display usage, 289
user queue element
  display
    using utility, 286
  remove stopped, 295
    using utility, 295
user queue elements
  display all
    using utility, 290
  display for specified user
    using utility, 290
  display up to five
    using operator command, 290
  display utility
    using utility, 290
user queue file list pool
  command to display usage, 289
USERID parameter, 1124
users
  change priority
    using utility, 299
  display count of
    using utility, 288
  resynchronize all ET logic users, 296
  set non-activity time limit
    using utility, 297
  stop those timed out
    using utility, 295
  stop those using a specified file
    using utility, 295
  stop those with a specified job name
    using utility, 296
  stop user with specified ID
    using utility, 296
utilities
  ADADBS function, 333
  sequential list of files, 1175
utility control statement
  parameter values, 5
    default, 3
    value, 6
    value list, 7
    value range, 7
  rules, 5
  syntax, 2
    parameter list, 2
  syntax symbols, 3
utility-only status
  switch on/off
    using utility, 298
UTIONLY
  operator command
    using utility, 298

V

value buffer
  start logging
    using utility, 292
  stop logging
    using utility, 293
vertical bars (|), 4
VSE
  library and file procedures for examples, 1189
  special considerations for using ADARAI, 795

W

wide-character fields
  no conversion option (NV), 116
Work data set
  formatting using ADAFRM, 414
  print blocks using ADAPRI, 713
  print data protection records using ADAPLP, 692
  resetting blocks/cylinders to zeros using ADAFRM, 415
Work file
  define
    using utility, 397
work pool
  command to display usage, 289

X

XID pool
command to display usage, 289